Curso de Python

CURSO COMPLETO PARTIENDO DESDE 0

Pere Manel Verdugo Zamora
www.peremanelv.com | pereverdugo@gmail.c0m

Contenido

Capítulo 1.- Instalación del lenguaje y pasos para crear nuestro primer programa 11
Capítulo 2.- Codificación del diagrama de flujo en Python 16
Capítulo 3.- Errores sintácticos y lógicos 22
Capítulo 4.- Estructura de programación secuencial – 1 24
Capítulo 5.- Estructura de programación secuencial – 2 26
Capítulo 6.- Estructura de programación secuencial – 3 28
Capítulo 7.- Estructura de programación secuencial – 4 29
Capítulo 8.- Estructura de programación secuencial – 5 31
Capítulo 9.- Estructura de programación secuencial – 6 32
Capítulo 10.- Condicionales simples y compuestas – 1 33
Capítulo 11.- Condicionales simples y compuestas – 2 36
Capítulo 12.- Condicionales simples y compuestas – 3 38
Capítulo 13.- Condicionales simples y compuestas – 4 40
Capítulo 14.- Condicionales simples y compuestas – 5 42
Capítulo 15.- Estructuras condicionales anidadas – 1 43
Capítulo 16.- Estructuras condicionales anidadas – 2 45
Capítulo 17.- Estructuras condicionales anidadas – 3 48
Capítulo 18.- Estructuras condicionales anidadas – 4 50
Capítulo 19.- Estructuras condicionales anidadas – 5 52
Capítulo 20.- Condiciones compuestas con operadores lógicos – 1 54
Capítulo 21.- Condiciones compuestas con operadores lógicos – 2 57
Capítulo 22.- Condiciones compuestas con operadores lógicos – 3 59
Capítulo 23.- Condiciones compuestas con operadores lógicos – 4 61
Capítulo 24.- Condiciones compuestas con operadores lógicos – 5 63
Capítulo 25.- Condiciones compuestas con operadores lógicos – 6 65
Capítulo 26.- Condiciones compuestas con operadores lógicos – 7 67
Capítulo 27.- Condiciones compuestas con operadores lógicos – 8 70
Capítulo 28.- Condiciones compuestas con operadores lógicos – 9 72
Capítulo 29.- Estructura repetitiva while – 1 75
Capítulo 30.- Estructura repetitiva while – 2 80
Capítulo 31.- Estructura repetitiva while – 3 82
Capítulo 32.- Estructura repetitiva while – 4 84
Capítulo 33.- Estructura repetitiva while – 5 86
Capítulo 34.- Estructura repetitiva while – 6 88
Capítulo 35.- Estructura repetitiva while – 7 90

capítulo 36.- Estructura repetitiva while – 8 92
Capítulo 37.- Estructura repetitiva while – 9 94
Capítulo 38.- Estructura repetitiva while – 10 96
Capítulo 39.- Estructura repetitiva while – 11 100
Capítulo 40.- Estructura repetitiva for – 1 102
Capítulo 41.- Estructura repetitiva for – 2 105
Capítulo 42.- Estructura repetitiva for – 3 107
Capítulo 43.- Estructura repetitiva for – 4 108
Capítulo 44.- Estructura repetitiva for – 5 109
Capítulo 45.- Estructura repetitiva for – 6 110
Capítulo 46.- Estructura repetitiva for - 7 111
Capítulo 47.- Estructura repetitiva for – 8 112
Capítulo 48.- Estructura repetitiva for – 9 113
Capítulo 49.- Estructura repetitiva for – 10 114
Capítulo 50.- Estructura repetitiva for – 11 116
Capítulo 51.- Estructura repetitiva for – 12 118
Capítulo 52.- Estructura repetitiva for – 13 120
Capítulo 53.- Definición de comentarios en el código fuente – 1 122
Capítulo 54.- Definición de comentarios en el código fuente – 2 124
Capítulo 55.- Definición de comentarios en el código fuente – 3 125
Capítulo 56.- Variables enteras, flotantes y cadena de caracteres – 1 126
Capítulo 57.- Variables enteras, flotantes y cadena de caracteres – 2 128
Capítulo 58.- Variables enteras, flotantes y cadena de caracteres – 3 130
Capítulo 59.- Variables enteras, flotantes y cadena de caracteres – 4 131
Capítulo 60.- Procesar cadenas de caracteres – 1 132
Capítulo 61.- Procesar cadenas de caracteres – 2 134
Capítulo 62.- Procesar cadenas de caracteres – 3 135
Capítulo 63.- Procesar cadenas de caracteres – 4 137
Capítulo 64.- Procesar cadenas de caracteres – 5 138
Capítulo 65.- Procesar cadenas de caracteres – 6 139
Capítulo 66.- Estructura de datos tipo lista – 1 140
Capítulo 67.- Estructura de datos tipo lista – 2 142
Capítulo 68.- Estructura de datos tipo lista – 3 143
Capítulo 69.- Estructura de datos tipo lista – 4 144
Capítulo 70.- Estructura de datos tipo lista – 5 145
Capítulo 71.- Estructuras de datos tipo lista – 6 146

Capítulo 72.- Listas: carga por teclado de sus elementos – 1 147
Capítulo 73.- Listas: carga por teclado de sus elementos – 2 149
Capítulo 74.- Listas: carga por teclado de sus elementos – 3 150
Capítulo 75.- Listas: carga por teclado de sus elementos – 4 151
Capítulo 76.- Listas: carga por teclado de sus elementos – 5 152
Capítulo 77.- Listas: mayor y menor elemento – 1 153
Capítulo 78.- Listas: mayor y menor elemento – 2 154
Capítulo 79.- Listas: mayor y menor elemento – 3 155
Capítulo 80.- Listas: mayor y menor elemento – 4 156
Capítulo 81.- Listas paralelas – 1 157
Capítulo 82.- Listas paralelas – 2 158
Capítulo 83.- Listas paralelas – 3 159
Capítulo 84.- Listas paralelas – 4 161
Capítulo 85.- Listas: ordenamiento de sus elementos – 1 162
Capítulo 86.- Listas: ordenamiento de sus elementos – 2 164
Capítulo 87.- Listas: ordenamiento de sus elementos – 3 165
Capítulo 88.- Listas: ordenamiento de sus elementos – 4 166
Capítulo 89.- Listas: ordenamiento de sus elementos – 5 168
Capítulo 90.- Listas: ordenamiento con listas paralelas – 1 169
Capítulo 91.- Listas: ordenamiento con listas paralelas – 2 171
Capítulo 92.- Listas: componentes de tipo lista – 1 173
Capítulo 93.- Listas: componentes de tipo lista – 2 175
Capítulo 94.- Listas: componentes de tipo lista – 3 176
Capítulo 95.- Listas: componentes de tipo lista – 4 177
Capítulo 96.- Listas: componentes de tipo lista – 5 178
Capítulo 97.- Listas: componentes de tipo lista – 6 179
Capítulo 98.- Listas: carga por teclado de componentes de tipo lista – 1 180
Capítulo 99.- Listas: carga por teclado de componentes de tipo lista – 2 183
Capítulo 100.- Listas: carga por teclado de componentes de tipo lista – 3 185
Capítulo 101.- Listas: carga por teclado de componentes de tipo lista – 4 186
Capítulo 102.- Listas: carga por teclado de componentes de tipo lista – 5 188
Capítulo 103.- Listas: carga por teclado de componentes de tipo lista – 6 190
Capítulo 104.- Listas: carga por teclado de componentes de tipo lista – 7 192
Capítulo 105.- Lista: eliminación de elementos – 1 194
Capítulo 106.- Listas: eliminación de elementos – 2 196
Capítulo 107.- Listas: Eliminación de elementos – 3 197

Capítulo 108.- Listas: Eliminación de elementos – 4 198
Capítulo 109.- Concepto de funciones: Programación estructurada – 1 199
Capítulo 110.- Concepto de funciones: Programación estructurada – 2 201
Capítulo 111.- Concepto de funciones: Programación estructurada – 3 203
Capítulo 112.- Concepto de funciones: Programación estructurada – 4 204
Capítulo 113.- Funciones: parámetros – 1 205
Capítulo 114.- Funciones: parámetros – 2 207
Capítulo 115.- Funciones: parámetros – 3 208
Capítulo 116.- Funciones: parámetros – 4 209
Capítulo 117.- Funciones: parámetros – 5 210
Capítulo 118.- Funciones: retorno de datos – 1 212
Capítulo 119.- Funciones: retorno de datos – 2 213
Capítulo 120.- Funciones: retorno de datos – 3 214
Capítulo 121.- Funciones: retorno de datos – 4 215
Capítulo 122.- Funciones: retorno de datos – 5 216
Capítulo 123.- Funciones: retorno de datos – 6 217
Capítulo 124.- Funciones: retorno de datos – 7 219
Capítulo 125.- Funciones: parámetros de tipo lista – 1 220
Capítulo 126.- Funciones: parámetros de tipo lista – 2 221
Capítulo 127.- Funciones: parámetros de tipo lista – 3 222
Capítulo 128.- Funciones: parámetros de tipo lista – 4 224
Capítulo 129.- Funciones: parámetros de tipo lista – 5 225
Capítulo 130.- Funciones: retorno de una lista - 1 226
Capítulo 131.- Funciones: retorno de una lista – 2 228
Capítulo 132.- Funciones: retorno de una lista – 3 229
Capítulo 133.- Funciones: retorno de una lista – 4 230
Capítulo 134.- Funciones: retorno de una lista – 5 232
Capítulo 135.- Funciones: retorno de una lista – 6 235
Capítulo 136.- Funciones: parámetros con valor por defecto – 1 237
Capítulo 137.- Funciones. parámetros con valor por defecto – 2 238
Capítulo 138.- Funciones: llamada a la función con argumentos nombrados – 1 239
Capítulo 139.- Funciones: llamada a la función con argumentos nombrados – 2 241
Capítulo 140.- Funciones: llamada a la función con argumentos nombrados – 3 242
Capítulo 141.- Funciones: con cantidad de variables de parámetro – 1 243
Capítulo 142.- Funciones: con cantidad variables de parámetros – desempaquetado de listas – 2 244

Capítulo 143.- Funciones: con cantidad variable de parámetro – 3 245
Capítulo 144.- Estructura de datos tipo Tupla – 1 246
Capítulo 145.- Estructura de datos tipo Tupla – 2 247
Capítulo 146.- Estructura de datos tipo Tupla – 3 249
Capítulo 147.- Estructura de datos tipo Tupla – 4 250
Capítulo 148.- Estructura de datos tipo Tupla – 5 251
Capítulo 149.- Listas y tuplas anidadas – 1 253
Capítulo 150.- Listas y tuplas anidadas – 2 254
Capítulo 151.- Listas y tuplas anidadas – 3 255
Capítulo 152.- Listas y tuplas anidadas – 4 257
Capítulo 153.- Variantes de la estructura repetitiva for para recorrer tuplas y listas – 1 259
Capítulo 154.- Variantes de la estructura repetitiva for para recorrer tuplas y listas – 2 261
Capítulo 155.- Variantes de la estructura repetitiva for para recorrer tuplas y listas – 3 263
Capítulo 156.- Variantes de la estructura repetitiva for para recorrer tuplas y listas – 4 264
Capítulo 157.- Estructura de datos tipo diccionario – 1..... 266
Capítulo 158.- Estructura de datos tipo diccionario – 2..... 268
Capítulo 159.- Estructura de datos tipo diccionario – 3..... 269
Capítulo 160.- Estructura de datos tipo diccionario – 4..... 271
Capítulo 161.- Diccionarios con valores de tipo lista, tuplas y diccionarios – 1..... 272
Capítulo 162.- Diccionarios con valores de tipo lista, tuplas y diccionarios – 2..... 274
Capítulo 163.- Diccionarios con valores de tipo, lista, tuplas y diccionarios – 3..... 276
Capítulo 164.- Funciones parámetros mutables e inmutables – 1 278
Capítulo 165.- Funciones parámetros mutables e inmutables – 2 280
Capítulo 166.- Funciones parámetros mutables e inmutables – 3 281
Capítulo 167.- Funciones parámetros mutables e inmutables – 4 283
Capítulo 168.- Porciones de listas, tuplas y cadenas de caracteres – 1 285
Capítulo 169.- Porciones de listas, tuplas y cadenas de caracteres – 2 286
Capítulo 170.- Porciones de listas, tuplas y cadenas de caracteres – 3 287
Capítulo 171.- Porciones de listas, tuplas y cadenas de caracteres – 4 288
Capítulo 172.- Porciones de listas, tuplas y cadenas de caracteres – 5 289
Capítulo 173.- Índices negativos en listas, tuplas y cadenas de caracteres – 1 290
Capítulo 174.- Índices negativos en listas, tuplas y cadenas de caracteres – 2 292
Capítulo 175.- Índices negativos en listas, tuplas y cadenas de caracteres – 3 293
Capítulo 176.- Biblioteca estándar de Python – 1..... 294
Capítulo 177.- Biblioteca estándar de Python – 2..... 296
Capítulo 178.- Biblioteca estándar de Python – 3..... 297

Capítulo 179.- Biblioteca estándar de Python – 4 298
Capítulo 180.- Importar algunas funcionaliddes de un módulo de la biblioteca estándar de Python – 1 300
Capítulo 181.- Importar algunas funcionalidades de un módulo de la biblioteca estándar de Python – 2 302
Capítulo 182.- Importar algunas funcionalidades de un módulo de la biblioteca estándar de Python – 3 303
Capítulo 183.- Aplicaciones propias con varios módulos – 1 304
Capítulo 184.- Aplicaciones propias con varios módulos – 2 306
Capítulo 185.- Declaración de una clase y creación de objetos – 1 308
Capítulo 186.- Declaración de una clase y creación de objetos – 2 310
Capítulo 187.- Declaración de una clase y creación de objetos – 3 311
Capítulo 188.- Declaración de una clase y creación de objetos – 4 312
Capítulo 189.- Método __init__ de la clase – 1 314
Capítulo 190.- Método __init__ de la clase – 2 316
Capítulo 191.- Método __init__ de la clase – 3 318
Capítulo 192.- Método __init__ de la clase – 4 319
Capítulo 193.- Llamada de métodos desde otro método de la misma clase – 1 320
Capítulo 194.- Llamada de métodos desde otro método de la misma clase – 2 322
Capítulo 195.- Llamada de métodos desde otro método de la misma clase – 3 324
Capítulo 196.- Colaboración de clases – 1 329
Capítulo 197.- Colaboración de clases – 2 331
Capítulo 198.- Colaboración de clases – 3 334
Capítulo 199.- Herencia – 1 335
Capítulo 200.- Herencia – 2 339
Capítulo 201.- Herencia – 3 341
Capítulo 202.- Variables de clase – 1 344
Capítulo 203.- Variables de clase – 2 347
Capítulo 204.- Método especial __str__ - 1 348
Capítulo 205.- Método especial __str__ - 2 350
Capítulo 206.- Método especial __str__ - 3 351
Capítulo 207.- Método especial __str__ - 4 353
Capítulo 208.- Redefinición de los operadores matemáticos con objetos – 1 354
Capítulo 209.- Redefinición de los operadores matemáticos con objetos – 2 356
Capítulo 210.- Redefinición de los operadores relacionales con objetos – 1 357
Capítulo 211.- Redefinición de los operadores relacionales con objetos – 2 360

Capítulo 212.- Instalación de editor Visual Studio Code (VSCode) y de la extensión para Python 362
Capítulo 213.- pip: instalador de paquetes 376
Capítulo 214.- Interfaz gráfica de usuario: tkinter 378
Capítulo 215.- tkinter: Controles Button y Label – 1 380
Capítulo 216.- tkinter: Controles Button y Label – 2 381
Capítulo 217.- tkinter: Controles Button y Label – 3 382
Capítulo 218.- tkinter: Controles Button y Label – 4 383
Capítulo 219.- tkinter: Control Entry – 1 385
Capítulo 220.- tkinter: Control Entry – 2 386
Capítulo 221.- tkinter: Control Entry – 3 387
Capítulo 222.- tkinter: Control Entry – 4 388
Capítulo 223.- tkinter: Control Radiobutton – 1 389
Capítulo 224.- tkinter: Control Radiobutton – 2 391
Capítulo 225.- tkinter: Control Radiobutton – 3 393
Capítulo 226.- tkinter: Control Checkbutton – 1 395
Capítulo 227.- tkinter: Control Checkbutton – 2 397
Capítulo 228.- tkinter: Control Checkbutton – 3 398
Capítulo 229.- tkinter: Control Listbox – 1 400
Capítulo 230.- tkinter: Control Listbox – 2 402
Capítulo 231.- tkinter: Control Listbox – 3 404
Capítulo 232.- tkinter: Control ListBox – 4 406
Capítulo 233.- Módulo ttk – 1 407
Capítulo 234.- Módulo ttk – 2 409
Capítulo 235.- Módulo ttk – 3 411
Capítulo 236.- Módulo ttk – 4 413
Capítulo 237.- Módulo ttk – 5 415
Capítulo 238.- ttk: Combobox – 1 417
Capítulo 239.- ttk: Control Combobox – 2 419
Capítulo 240.- tkinter: Control Menu – 1 420
Capítulo 241.- tkinter: Control Menu – 2 428
Capítulo 242.- ttk: Controles Notebook y Frame 430
Capítulo 243.- ttk: Control LabelFrame – 1 433
Capítulo 244.- ttk: Control LabelFrame – 2 436
Capítulo 245.- tkinter: Layout Manager (administrador de diseño) – pack 438
Capítulo 246.- tkinter: Layout Manager (administrador de diseño) – grid 441

Capítulo 247.- tkinter: Layout Manager (administrador de diseño) – place 443
Capítulo 248.- tkinter: messagebox (ventana de mensajes) – 1 445
Capítulo 249.- tkinter: messagebox (ventana de mensajes) – 2 449
Capítulo 250.- tkinter: Ventanas de diálogo (clase Toplevel) 451
Capítulo 251.- ttk: Control Spinbox 455
Capítulo 252.- tkinter: ScrolledText (editor multilínea) 458
Capítulo 253.- tkinter: Control Canvas – 1 462
Capítulo 254.- tkinter: Control Canvas – 2 464
Capítulo 255.- tkinter: Control Canvas – 3 467
Capítulo 256.- tkinter: Control Canvas – 4 470
Capítulo 257.- tkinter: Control Canvas y la captura de eventos del mouse – 1 473
Capítulo 258.- tkinter: Control Canvas y la captura de eventos del mouse – 2 476
Capítulo 259.- Canvas: Borrar figuras mediante Ids y Tags 478
Capítulo 260.- Canvas: Desplazar una figura mediante el método move – 1 481
Capítulo 261.- Canvas: Desplazar una figura mediante el método move – 2 483
Capítulo 262.- Canvas: Mostrar una imagen – 1 485
Capítulo 263.- Canvas: Mostrar una imagen – 2 487
Capítulo 264.- Canvas: Mover una figura – 1 489
Capítulo 265.- Canvas: Mover una figura – 2 492
Capítulo 266.- Archivos de texto: Creación, escritura y lectura – 1 493
Capítulo 267.- Archivos de texto: Creación, escritura y lectura – 2 495
Capítulo 268.- Archivos de texto: Creación, escritura y lectura – 3 497
Capítulo 269.- Archivos de texto: Creación, escritura y lectura – 4 498
Capítulo 270.- Archivos de texto: Creación, escritura y lectura – 5 500
Capítulo 271.- Archivos de texto: Interfaz visual tkinter para mostrar y recuperar archivos de texto 501
Capítulo 272.- MySQL: Base de datos desde Python – 1 506
Capítulo 273.- MySQL: Base de datos desde Python – 2 515
Capítulo 274.- MySQL: Base de datos desde Python – 3 517
Capítulo 275.- MySQL: Base de datos desde Python – 4 519
Capítulo 276.- MySQL: Base de datos desde Python – 5 520
Capítulo 277.- MySQL: Interfaz visual con tkinter y acceso a la base de datos – 1 522
Capítulo 278.- MySQL: Interfaz visual con tkinter y acceso a la base de datos – 2 531
Capítulo 279.- MySQL: Otras funciones del paquete mysql connector – 1 539
Capítulo 280.- MySQL: Otra funcionalidad el paquete mysql connector – 2 540
Capítulo 281.- MySQL: Otras funcionalidades del paquete mysql connector – 3 541

Capítulo 282.- MySQL: Otras funcionalidades del paquete mysql connector – 4 542
Capítulo 283.- MySQL: Otras funcionalidades del paquete mysql connector -5 543
Capítulo 284.- Manejo de excepciones en Python..... 544
Capítulo 285.- Excepciones estándar de Python..... 546
Capítulo 286.- Instrucción finally en un bloque try..... 549
Capítulo 287.- Base de datos desde Python..... 552
Capítulo 288.- SQLite: Interfaz visual con tkinter y acceso a la base de datos – 1 555
Capítulo 289.- SQLite: Interfaz visual con tkinter y acceso a la base de datos – 2 561
Capítulo 290.- Python: f-strings para formateo de cadenas de caracteres 568
Capítulo 291.- Recursividad: Conceptos básicos – 1 572
Capítulo 292.- Recursividad: Conceptos básicos – 2 576
Capítulo 293.- Recursividad: Conceptos básicos – 3 578
Capítulo 294.- Recursividad: Problemas donde conviene aplicar la recursividad - 1 579
Capítulo 295.- Recursividad: Problemas donde conviene aplicar la recursividad - 2 580
Capítulo 296.- Recursividad: Problemas donde conviene aplicar la recursividad – 3..... 584
Capítulo 297.- Funciones de orden superior – 1 591
Capítulo 298.- Funciones de orden superior – 2 593
Capítulo 299.- Funciones lambda en Python – 1..... 594
Capítulo 300.- Funciones lambda en Python – 2..... 595
Capítulo 301.- Funciones lambda en Python – 3..... 597
Capítulo 302.- Comprensión de listas en Python – 1 598
Capítulo 303.- Comprensión de listas en Python – 2 600
Capítulo 304.- Comprensión de listas en Python - 3 601
Capítulo 305.- Estructura de datos tipo conjunto: set – 1 603
Capítulo 306.- Estructura de datos tipo conjunto: set – 2 606
Capítulo 307.- Estructura de datos tipo conjunto: set – 3 609
Capítulo 308.- PostgresSQL: Base de datos desde Python..... 610
Capítulo 309.- PostgeSQL: Interfaz visual con tkinter y acceso a la base de datos – 1..... 617
Capítulo 310.- PostgreSQL: Interfaz visual con tkinter y acceso a la base de datos – 2 622
Capítulo 311.- Módulos y paquetes de Python..... 629
Capítulo 312.- Acceso a Internet mediante el paquete urllib – 1 632
Capítulo 313.- Acceso a Internet mediante el paquete utllib – 2 636
Capítulo 314.- Procesar archivos con formato JSON en Python módulo json – 1 638
Capítulo 315.- Procesar archivos con formato JSON en Python módulo json – 2 644
Capítulo 316.- Procesar archivos con formato JSON en Python módulo json – 3 646
Capítulo 317.- Procesar archivos con formato JSON en Python módulo json – 4 650

Capítulo 318.- Procesar archivos con formato JSON en Python módulo json – 5 652
Capítulo 319.- Serialización y deserialización en Python (módulo json)..... 655

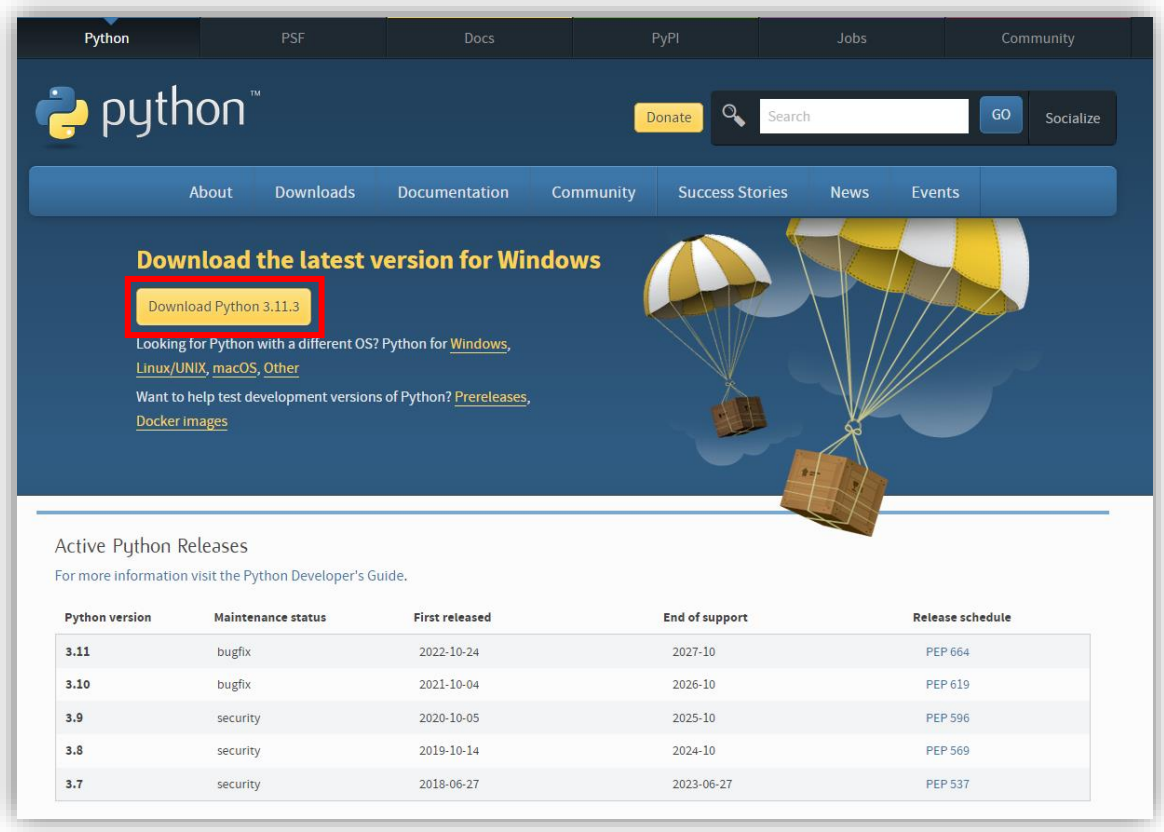
Realizando este tutorial resumen de Diego Moiseet de Espanes he aprendido mucho sobre Python, espero que este tutorial pueda serles de mucha ayuda.

Para acceder a los tutoriales de Diego Moisset de Espanes puedes escanear el código QR o acceder a la siguiente URL:
https://www.youtube.com/playlist?list=PLIygiKpYTC_6wtOmJe3pbqbWbvlopYZoD

Capítulo 1.- Instalación del lenguaje y pasos para crear nuestro primer programa

Lo primero que vamos a hacer es descargar Python para ello accederemos al siguiente enlace:

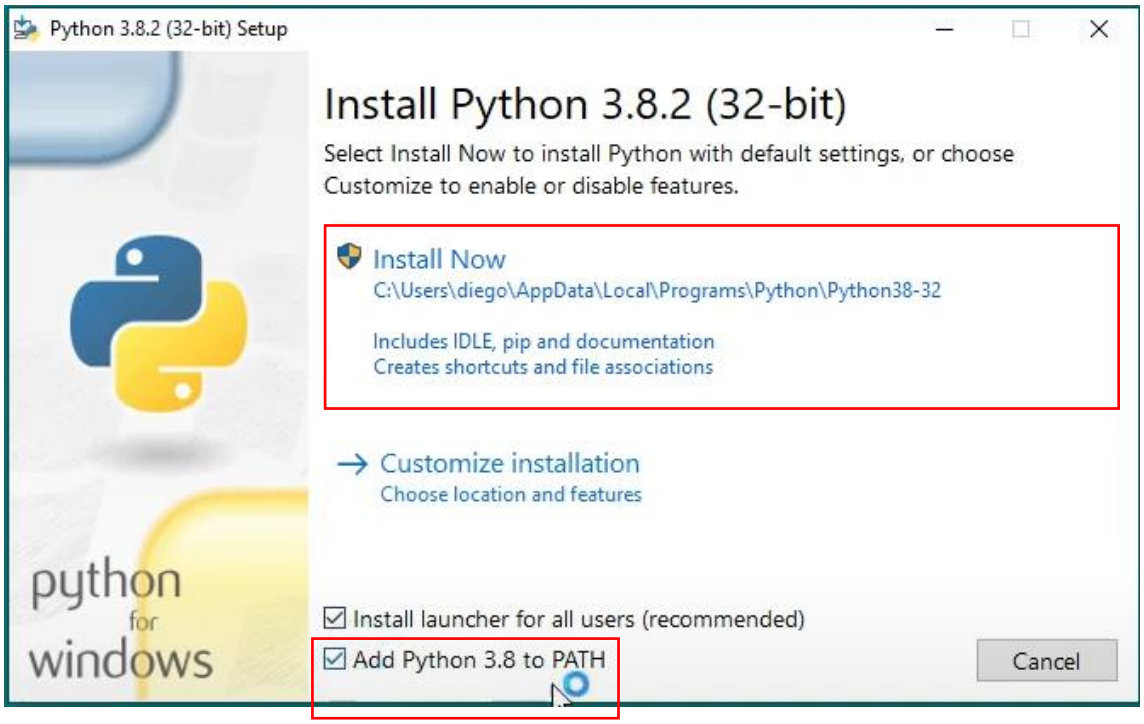
https://www.python.org/downloads/



Y procederemos a su descarga.

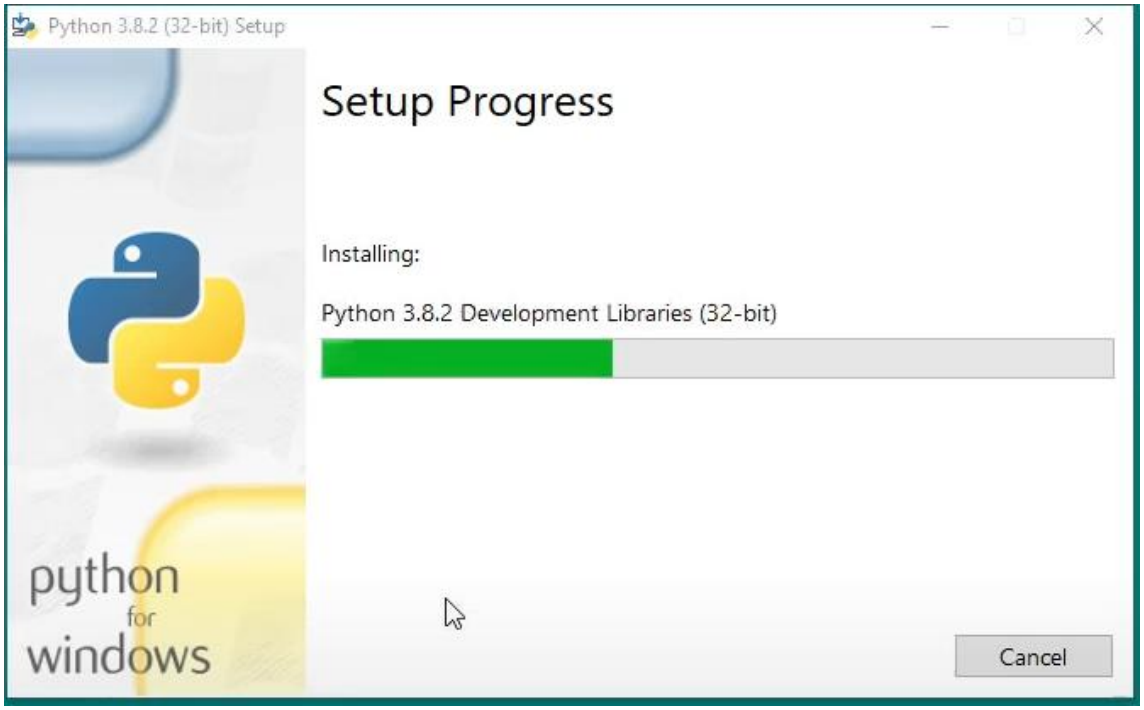
A continuación procederemos a su instalación:

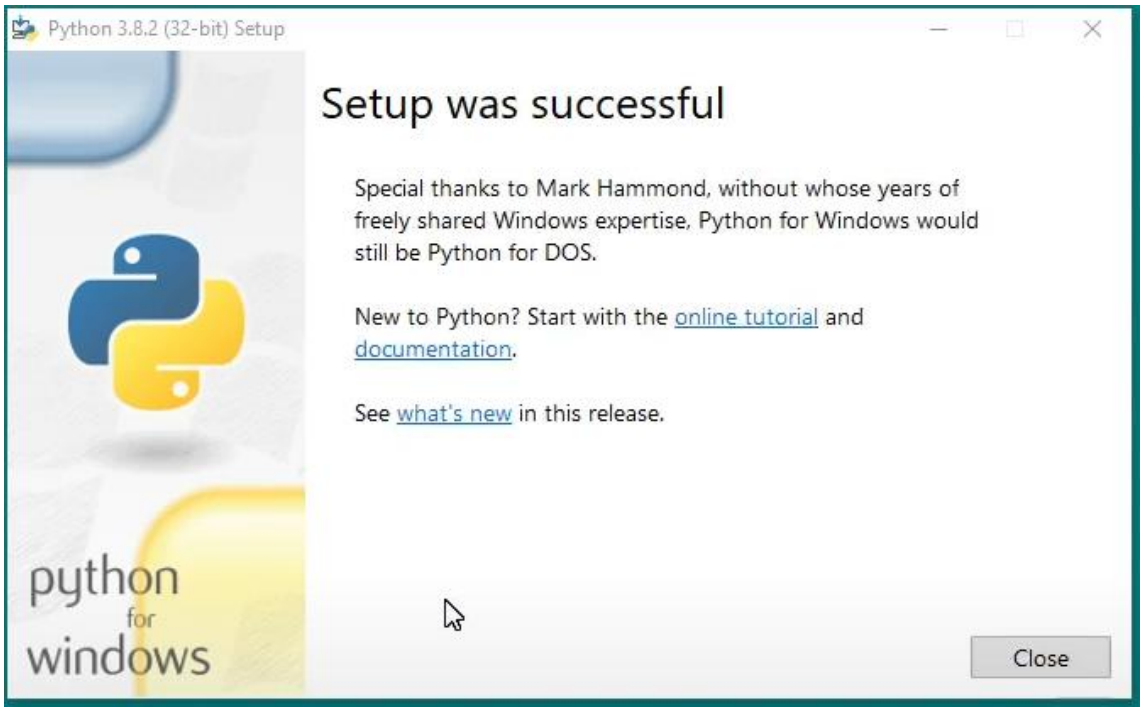
Lo único que tenemos que tener en cuenta es activar la opción Add Pyton..



Desde este modo tendremos acceso a Python desde cualquier directorio donde nos encontremos.

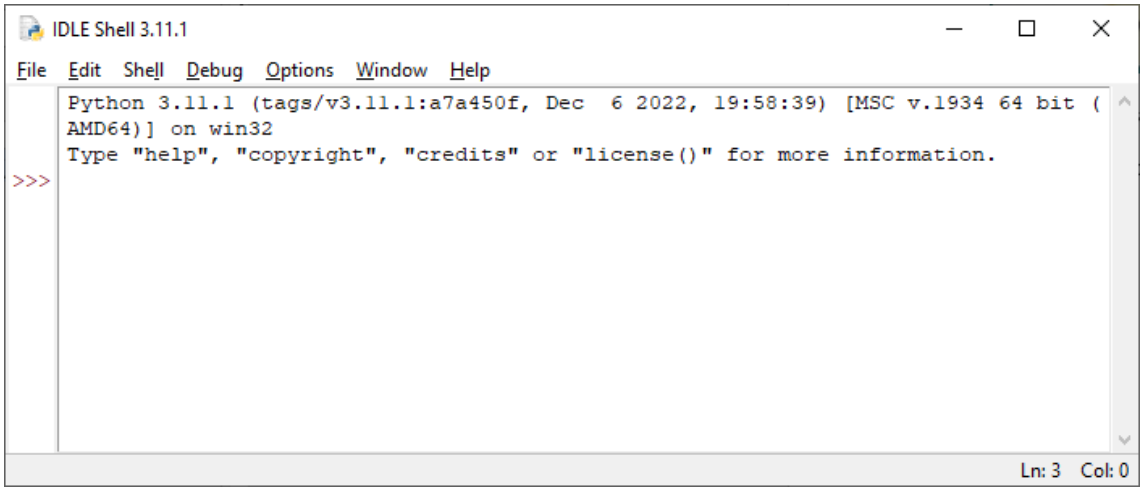
A continuación empezaremos con la instalación.





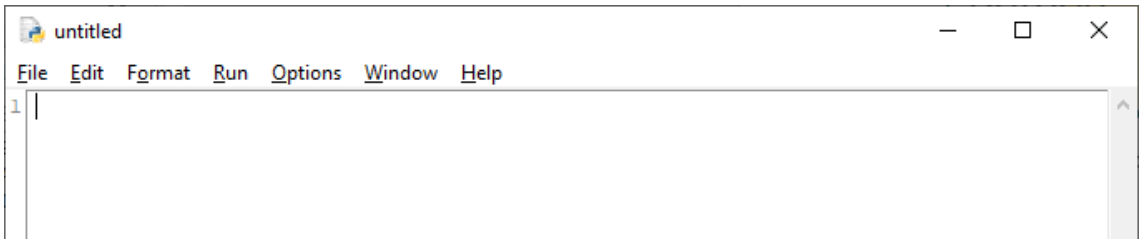
Ya hemos instalada Python seleccionamos el botón Close.

Python trae por defecto un editor que nos servirá para realizar los proyectos del curso.

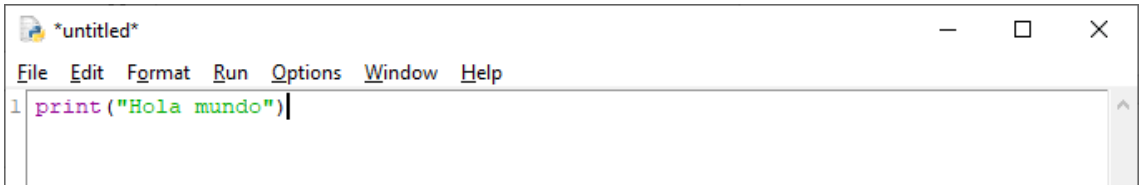


Este editor de texto es muy limitado pero para el aprendizaje es el adecuado.

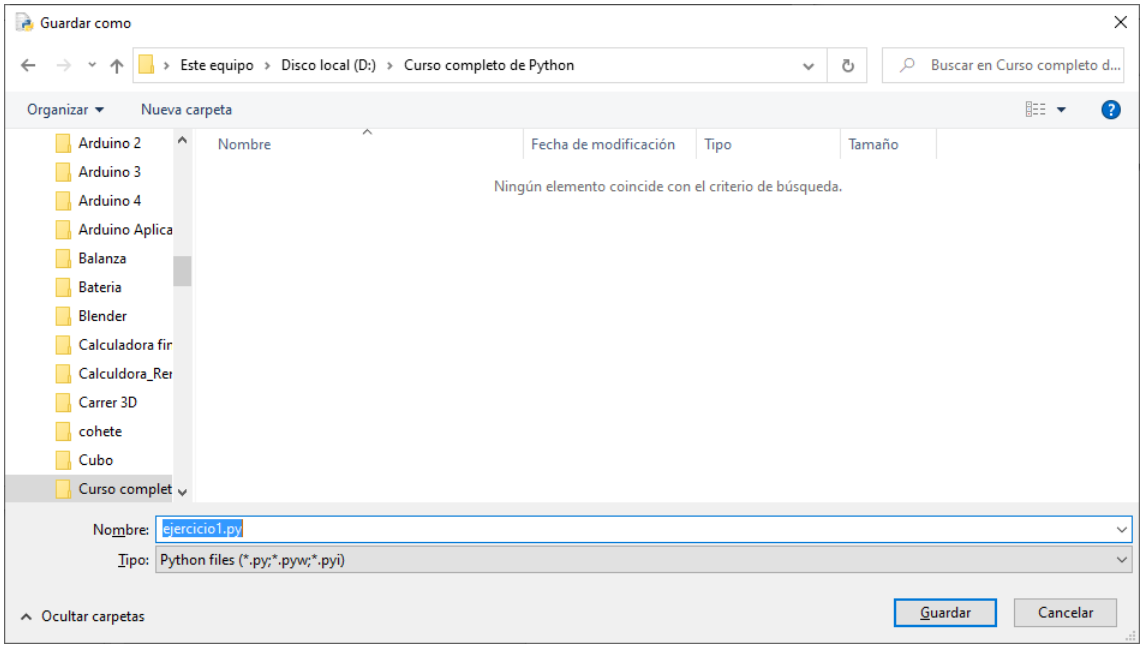
Para crear una aplicación del menú File seleccionaremos New File.



Vamos a escribir el siguiente texto:

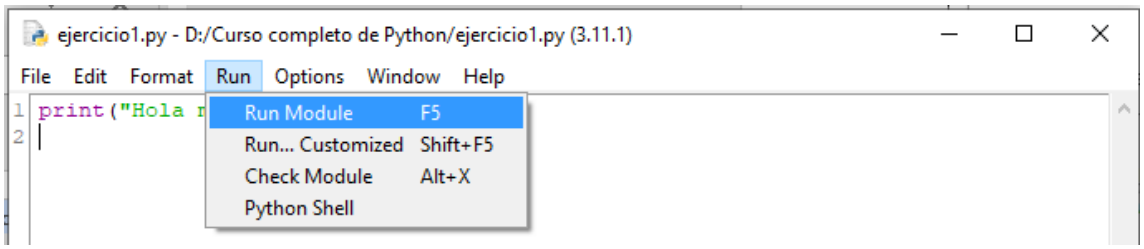


Ahora tenemos que guardar el proyecto, para ello del menú File seleccionaremos Save as…

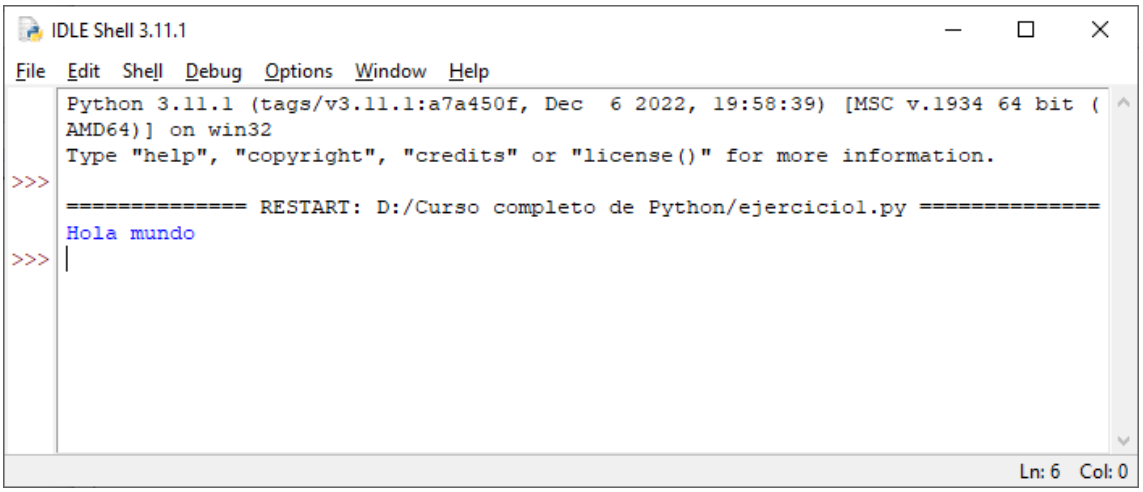


Lo guardaremos en una carpeta que previamente hemos creado para el curos en la que pondremos un nombre seguido de su extensión 'py', en este caso ejemplo1.py. Seguido del botón Guardar.

Para ejecutar del menú seleccionaremos la opción Run y de este Run Module o bien la tecla de función F5.



Este será el resultado:



Más adelante instalaremos otros editores que nos van a facilitar la programación, cuando hagamos programas más complejos.

Capítulo 2.- Codificación del diagrama de flujo en Python

¿Qué es un programa?

Programa: Conjunto de instrucciones que contiene una computadora para realizar una actividad. Todo programa tiene un objetivo bien definido: un procesador de textos es un programa que permite cargar, modificar e imprimir textos, un programa de ajedrez permite jugar contra el ordenador u otro contrincante humano.

La actividad fundamental del programador es resolver problemas empleando el ordenador como herramienta fundamental.

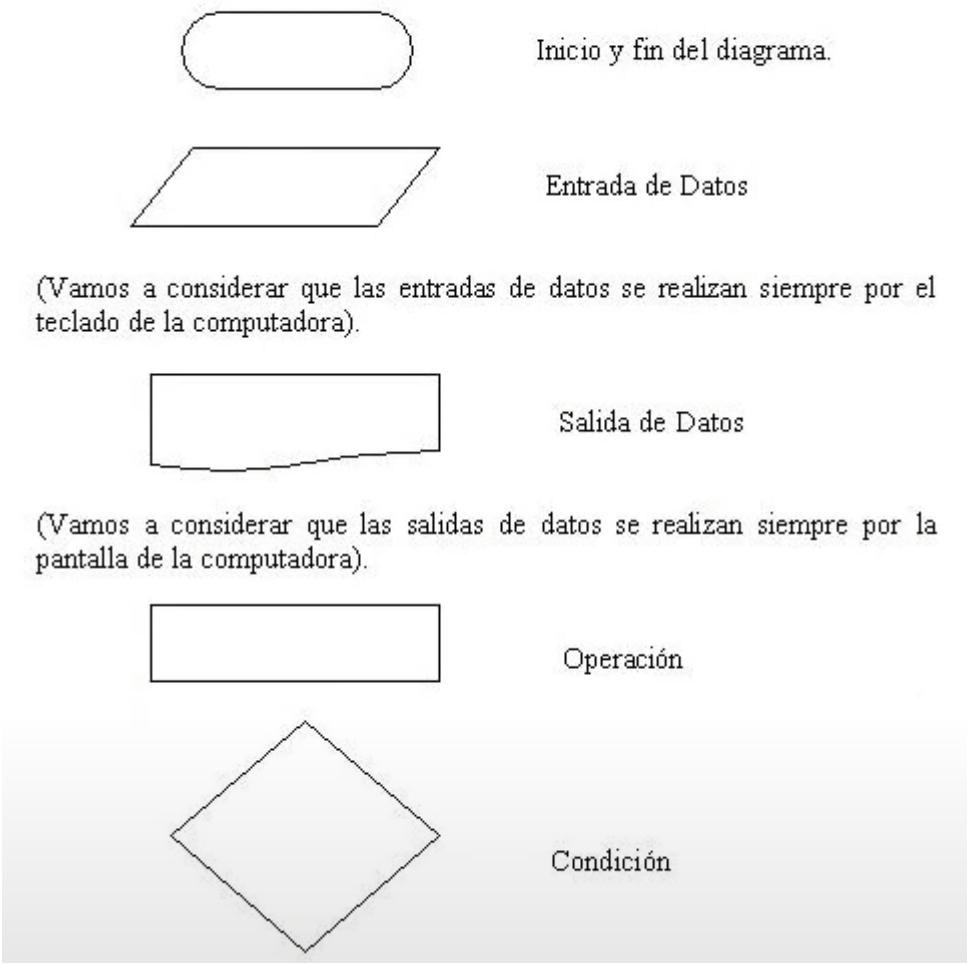
Para la resolución de un problema hay que plantear un algoritmo.

Algoritmo: Son los pasos a seguir par resolver un problema.

Diagrama de flujo

Un diagrama de flujo es la representación gráfica de un ALGORITMO.

Los símbolos gráficos principales a utilizar para el plateo de diagramas de flujo son:

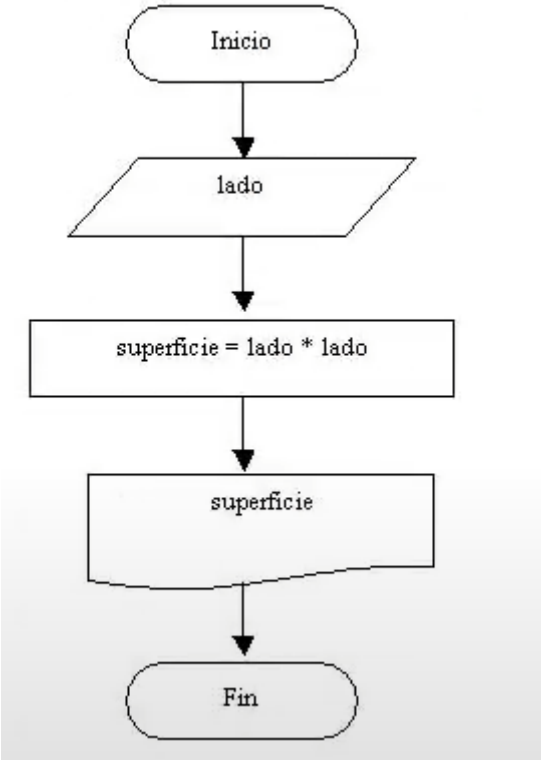


Planteo de un problema utilizando un diagrama de flujo

Para plantear un diagrama de flujo debemos tener muy en claro el problema a resolver.

Problema

Halla la superficie de un cuadrado conociendo el valor de un lado.



Resulta mucho más fácil entender un gráfico que un texto.

El diagrama de flujo nos identifica claramente los datos de entrada, operaciones y datos de salida.

En el ejemplo tenemos como entrada el valor del lado del cuadrado, en las entradas las representamos con un paralelogramo y hacemos un paralelogramo por cada dato de entrada.

La operación se representa con un rectángulo, debemos hacer un rectángulo por cada operación. A la salida la representamos con la hoja rota.

El diagrama de flujo nos da una idea del orden de ejecución de las actividades en el tiempo. Primero cargamos los datos de entrada, luego hacemos las operaciones necesarias y por último mostramos los resultados.

Variable: Es un depósito donde hay un valor. Consta de un nombre y pertenece a un tipo de dato.

Para el ejemplo planteado la variable lado almacena la medida del lado del cuadrado y la variable superficie almacena la superficie del cuadrado.

En el ejemplo tenemos dos variables.

Tipos de variables

Una variable puede almacenar:

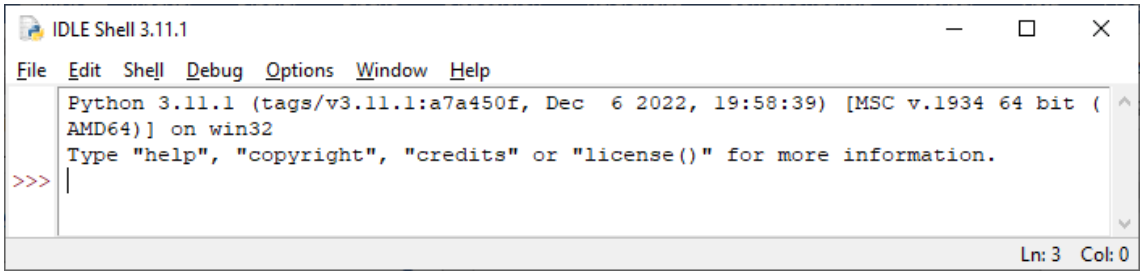
- Valores enteros (100, 260, etc.)

- Valores reales (1.24, 2.90, 5.00, etc.)
- Cadenas de caracteres ("Juan", "Compras", "Listado", etc.)

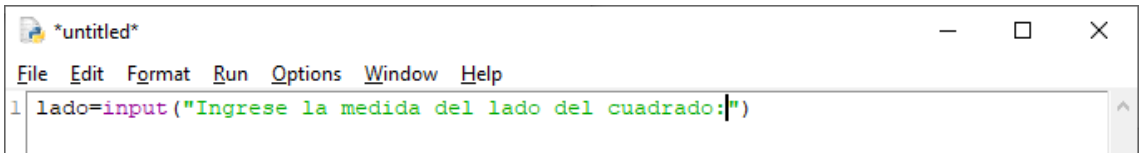
Elección del nombre de una variable:

Debemos elegir nombres de variables representativas. En el ejemplo el nombre lado es lo suficientemente claro para darnos una idea acabada sobre su contenido.

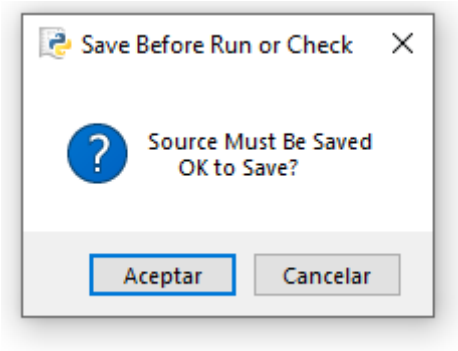
Vamos al entorno de programación Python.



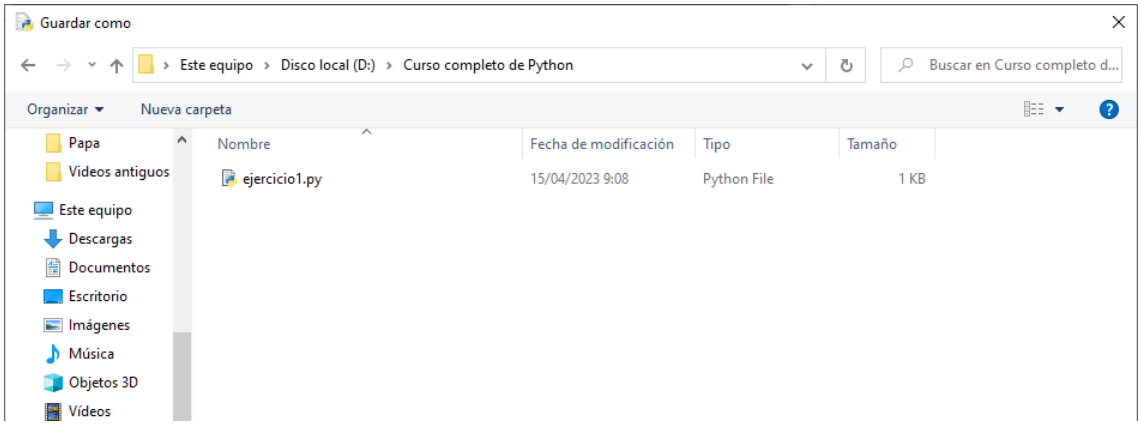
Del menú Files seleccionaremos New Files

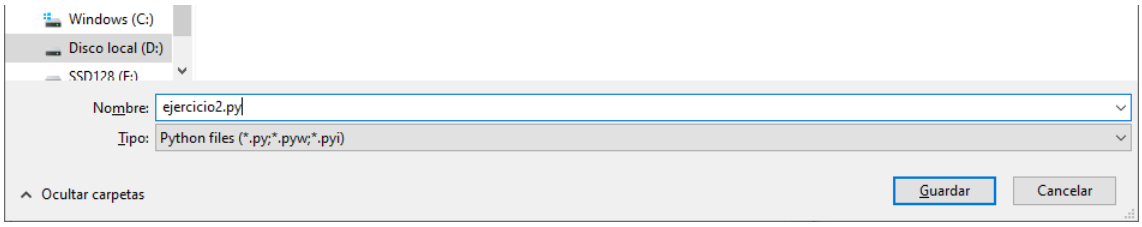


Si del menú Run seleccionamos Run Module.

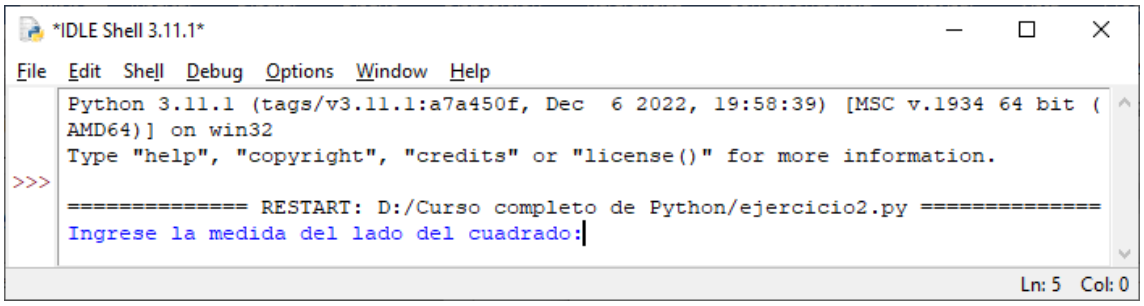


Nos dice que hay que guardarlo, le damos a aceptar.



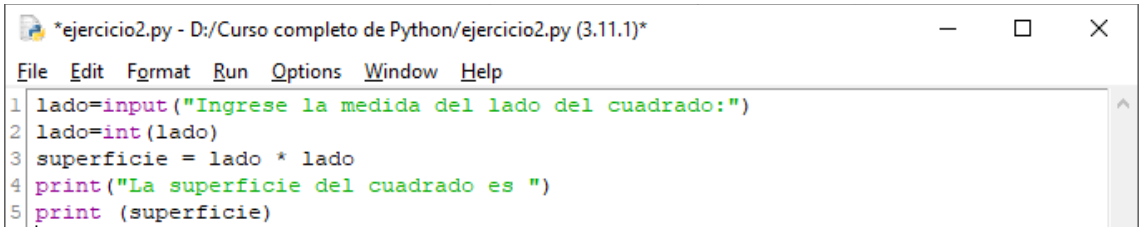


Seleccionamos la carpeta donde vayamos a guardar nuestros programas seguido del nombre 'Ejemplo2.py', a continuación seleccionamos el botón Guardar.



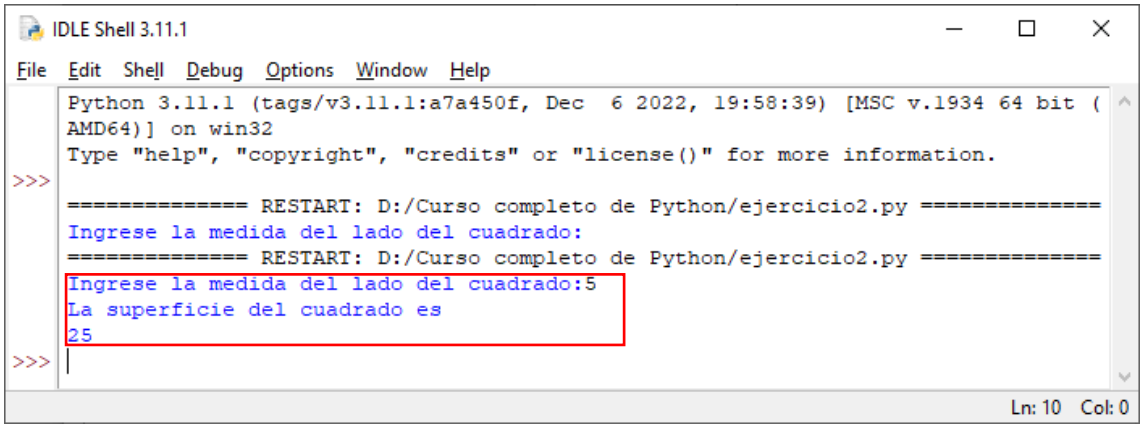
Ya nos muestra el texto para que introduzcamos el valor del lado del cuadrado.

El programa aún no está terminado.

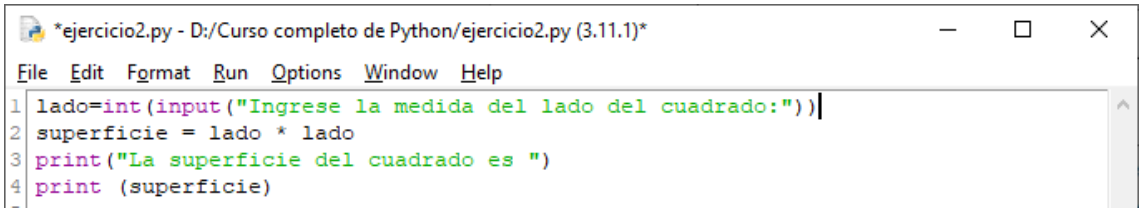


Input retorna un valor de texto para convertirlo a número utilizamos la función int().

Ejecutamos de nuevo.



También se puede realizar de la siguiente forma:



Ten en cuenta que este lenguaje es sensible de mayúsculas a minúsculas .

```
*ejercicio2.py - D:/Curso completo de Python/ejercicio2.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 lado=int(input("Ingrese la medida del lado del cuadrado:"))
2 Superficie = lado * lado
3 print("La superficie del cuadrado es ")
4 print (superficie)
```

Vamos a ejecutar:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio2.py ==============
    Ingrese la medida del lado del cuadrado:10
    La superficie del cuadrado es
    Traceback (most recent call last):
      File "D:/Curso completo de Python/ejercicio2.py", line 4, in <module>
        print (superficie)
    NameError: name 'superficie' is not defined. Did you mean: 'Superficie'?
>>>
                                                                    Ln: 18  Col: 0
```

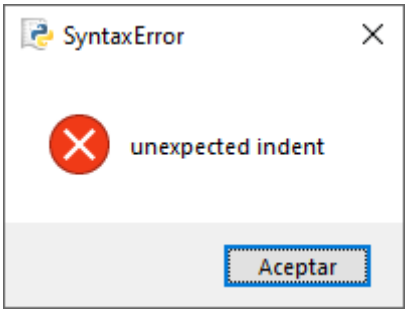
Nos muestra un error en la línea 4 ya que la variable Superficie y superficie no son las mismas.

Las variables no pueden tener espacios en blanco. Nos dará un error sintáctico.

Otro error es no ponerlo en las mismas columnas.

```
*ejercicio2.py - D:/Curso completo de Python/ejercicio2.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 lado=int(input("Ingrese la medida del lado del cuadrado:"))
2 Superficie = lado * lado
3     print("La superficie del cuadrado es ")
4 print (superficie)
```

Si ejecutamos:




```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio2.py ==============
    Ingrese la medida del lado del cuadrado:10
    La superficie del cuadrado es
    Traceback (most recent call last):
      File "D:/Curso completo de Python/ejercicio2.py", line 4, in <module>
        print (superficie)
    NameError: name 'superficie' is not defined. Did you mean: 'Superficie'?
>>>
                                                                    Ln: 18  Col: 0
```

```
lado=int(input("Ingrese la medida del lado del cuadrado:"))
superficie = lado * lado
print("La superficie del cuadrado es ")
print("superficie")
```

La variable superficie no debe ir entre comillas ya que esto imprimirá el texto, vamos a ejecutar.

```
========== RESTART: D:/Curso completo de Python/ejercicio2.py ==========
Ingrese la medida del lado del cuadrado:15
La superficie del cuadrado es
superficie
>>>
```

Capítulo 3.- Errores sintácticos y lógicos

Modificaremos el problema del concepto anterior y agregaremos adrede una serie de errores tipográficos. Este tipo de errores siempre son detectados por el intérprete de Python, antes de ejecutar el programa.

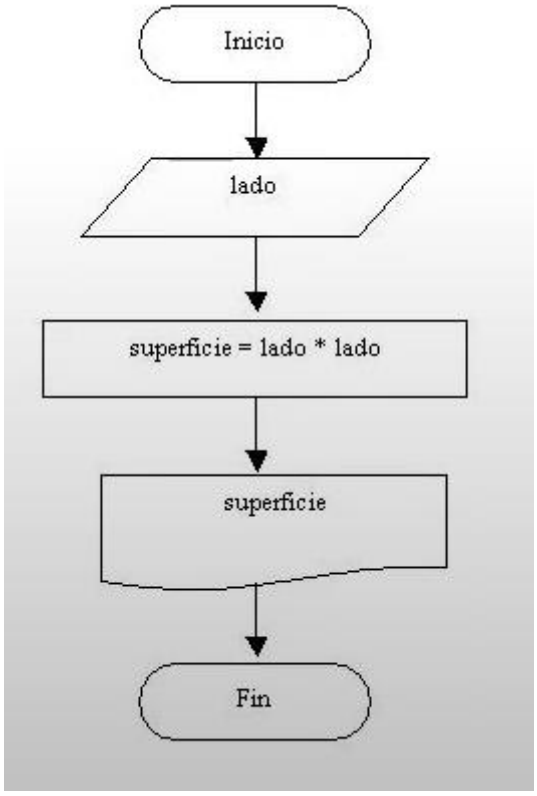
A los errores tipográficos, como por ejemplo indicar el nombre incorrecto de la función, nombres de variables incorrectas, falta de paréntesis, palabras claves mal escritas, etc. los llamamos errores SINTÁCTICOS.

Un programa no se puede ejecutar por completo sin corregir absolutamente todos los errores sintácticos.

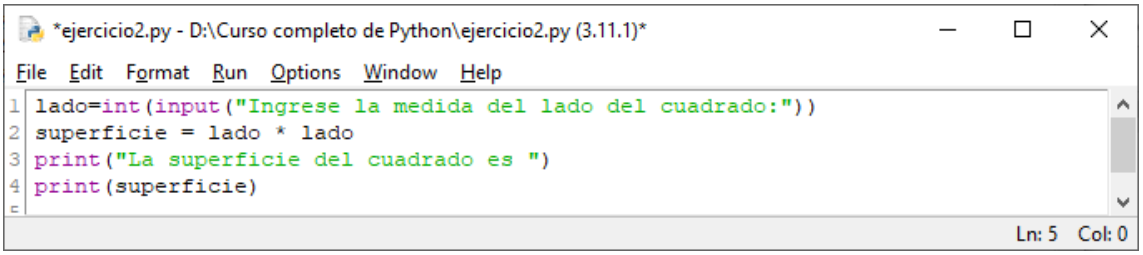
Existen otro tipo de errores llamados ERRORES LÓGICOS. Este tipo de errores en programas grandes (miles de líneas) son más difíciles de localizar. Por ejemplo un programa que permite hacer la facturación pero la salida de datos por impresora es incorrecta.

Problema

Hallar la superficie de un cuadrado conociendo el valor de un lado.



Vamos a recuperar el programa llamado Ejercicio2.py.



Cambiamos la función input por imput y ahora ejecuta el programa.

Cambiamos la función input por Input y ahora ejecuta el programa.

Borramos las dobles comillas finales de la línea 1.

La variable superficie le asignamos un espacio en medio super ficie.

La variable lado en otro momento la ponemos en mayúsculas Lado.

Estos errores son errores sintácticos, estos tendrán que corregirse para poder ejecutar el programa.

Los errores lógicos son mucho más difícil de localizarlos.

```
*ejercicio2.py - D:\Curso completo de Python\ejercicio2.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 lado=int(input("Ingrese la medida del lado del cuadrado:"))
2 superficie = lado * lado * lado
3 print("La superficie del cuadrado es ")
4 print(superficie)
                                                    Ln: 6  Col: 0
```

Lo vamos a ejecutar:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:\Curso completo de Python\ejercicio2.py ==============
Ingrese la medida del lado del cuadrado:5
La superficie del cuadrado es
125
                                                    Ln: 8  Col: 0
```

El programa se ejecuta pero el resultado obtenido es erróneo.

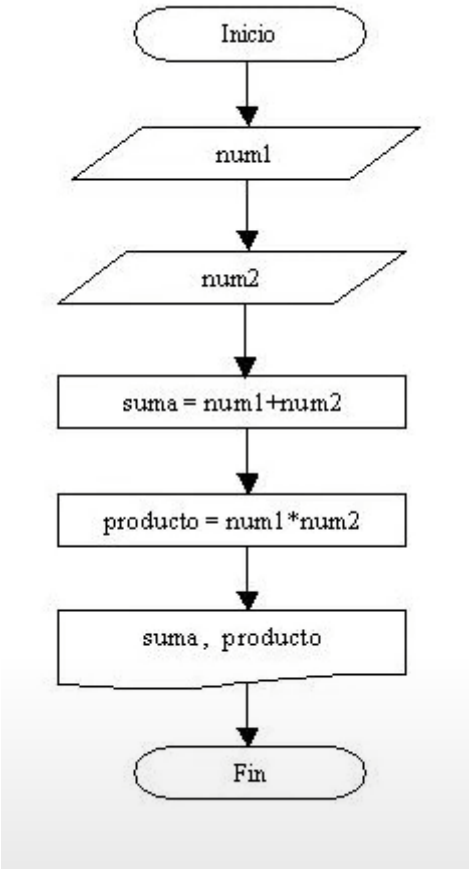
Capítulo 4.- Estructura de programación secuencial – 1

Cuando en un problema sólo participan operaciones, entradas y salidas se la denomina una estructura secuencial.

El problema diagramados y codificado anteriormente solo una estructura secuencial.

Problema

Realizar la carga de do números enteros por teclado e imprimir la suma y su producto.



Tenemos dos entradas num1 y num2, dos operaciones: la suma y el producto de los valores ingresados y dos salidas, que son los resultados de la suma y el producto de los valores ingresados. En el símbolo de impresión podemos indicar una o más salidas, eso queda a criterio del programador, lo mismo para indicar las entradas por teclado.

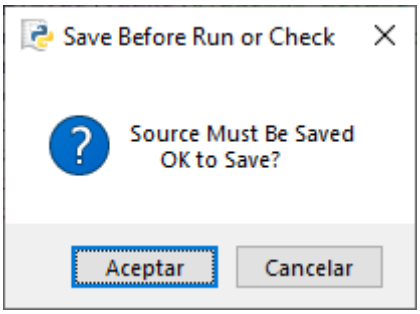
Vamos a realizar el programa llamado Ejercicio3.py.

ejercicio3.py - D:/Curso completo de Python/ejercicio3.py (3.11.1)

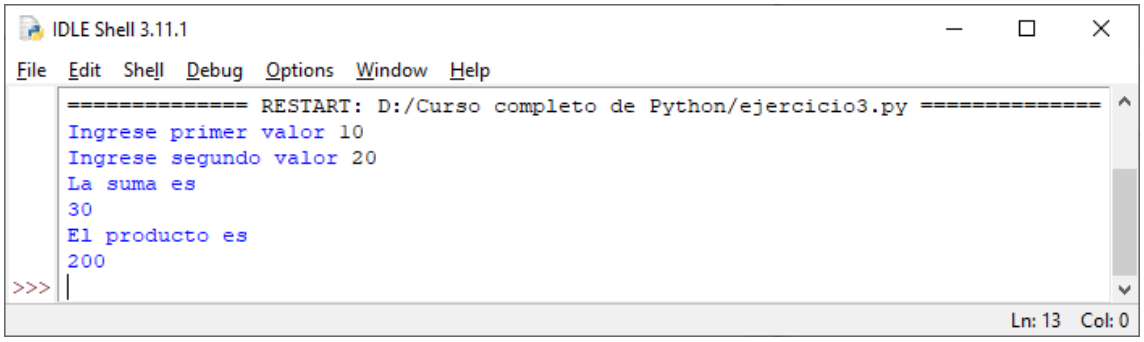
File Edit Format Run Options Window Help

```
1 num1=int(input("Ingrese primer valor "))
2 num2=int(input("Ingrese segundo valor "))
3 suma=num1+num2
4 producto=num1*num2
5 print("La suma es")
6 print(suma)
7 print("El producto es")
8 print(producto)
9
```

Vamos a ejecutar:



Nos pregunta si queremos guardar los cambios a lo que contestaremos Aceptar.

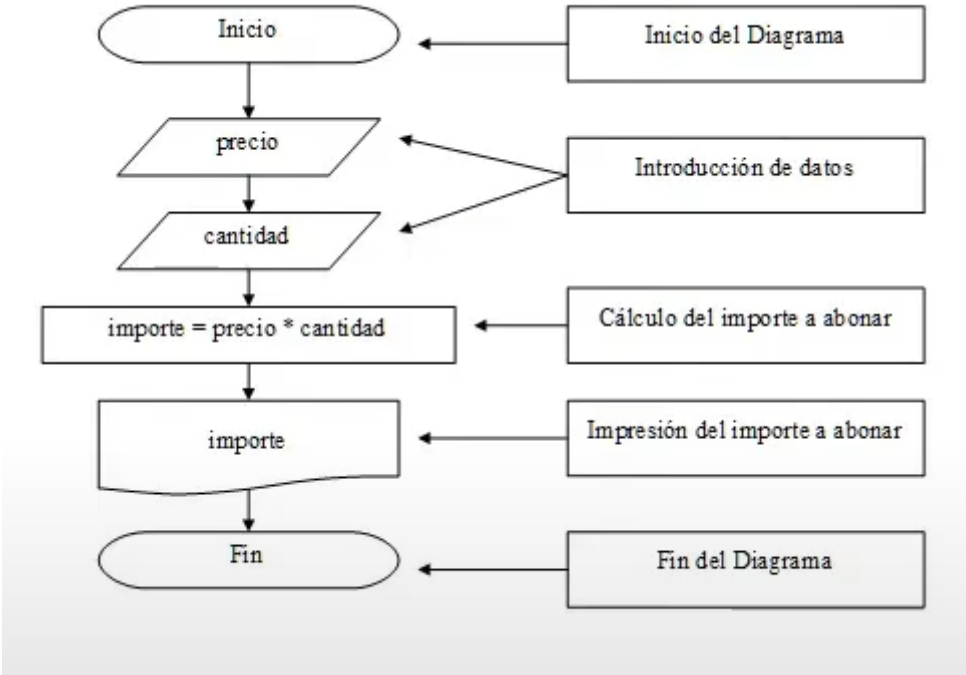


Introducimos los valores 10 y 20 obteniendo la suma 30 y el producto 200.

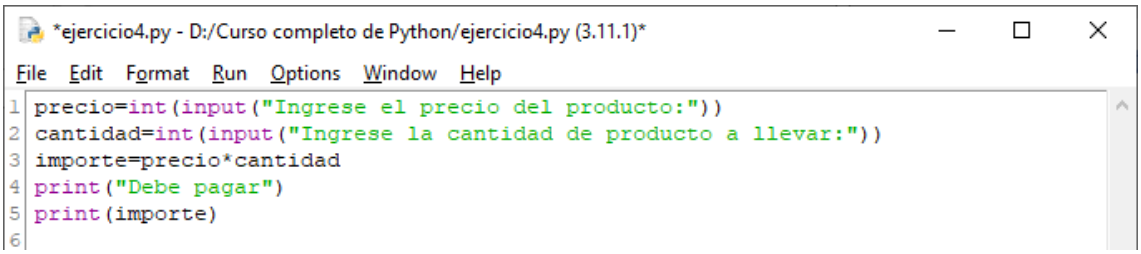
Capítulo 5.- Estructura de programación secuencial – 2

Problema

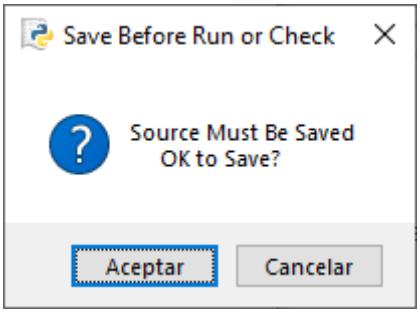
Realizar la carga de precio de un producto y la cantidad a llevar. Mostrar cuanto se debe pagar (se ingresa un valor entero en el precio del producto y la cantidad)



Vamos a realizar el programa con el nombre Ejercicio4.py.



Para ejecutar también podemos pulsar la tecla de función F5.



Guardamos los cambios pulsando el botón Aceptar.

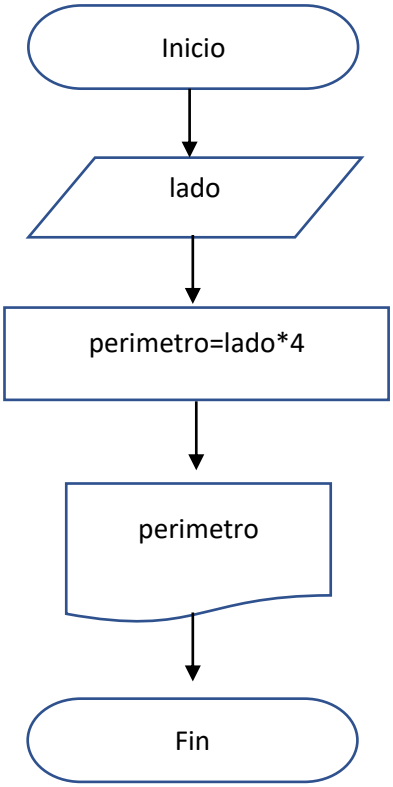
```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio4.py ==============
Ingrese el precio del producto:10
Ingrese la cantidad de producto a llevar:2
Debe pagar
20
>>>
                                                                    Ln: 9  Col: 0
```

Capítulo 6.- Estructura de programación secuencial – 3

Problema

Realiza la carga del lado de un cuadrado, mostrar por pantalla el perímetro del mismo (El perímetro de un cuadrado se calcula multiplicando el valor del lado por cuatro).

Diagrama de flujo:



El programa:

```
*untitled*
File  Edit  Format  Run  Options  Window  Help
1 lado=int(input("Lado del cuadrado:"))
2 perimetro=lado*4
3 print("El perimetro es")
4 print(perimetro)
5
```

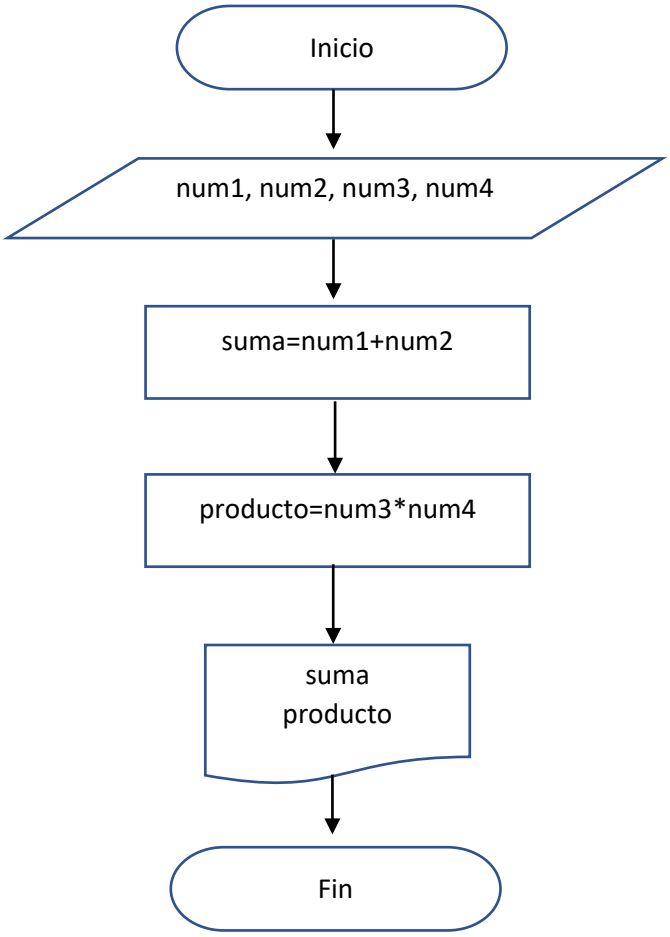
Lo guardamos como Ejercicio5.py.
Lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio5.py ==============
Lado del cuadrado:10
El perimetro es
40
```

Capítulo 7.- Estructura de programación secuencial – 4

Problema propuesto:

Escribir un programa en el cual se ingresen cuatro números, calcular e informar la suma de los dos primeros y el producto del tercero y el cuarto.



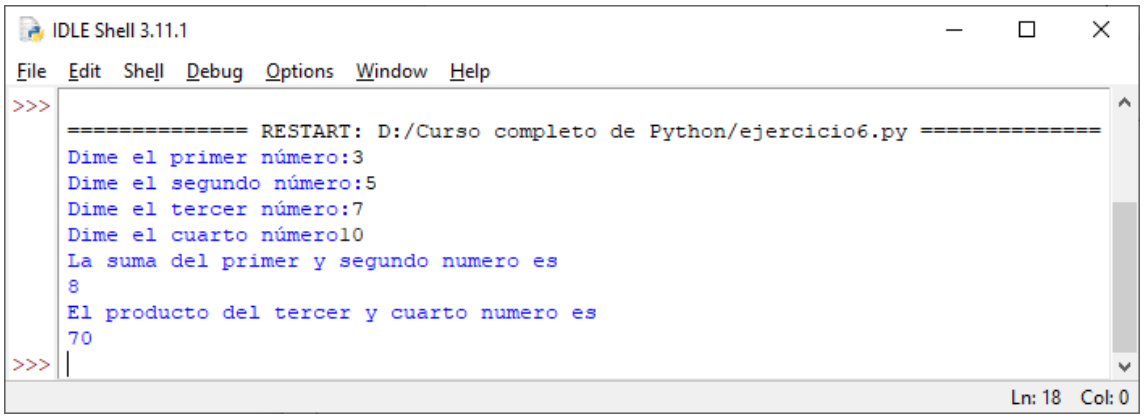
Vamos a realizar el programa con el nombre Ejercicio6.py.

ejercicio6.py - D:/Curso completo de Python/ejercicio6.py (3.11.1)

File Edit Format Run Options Window Help

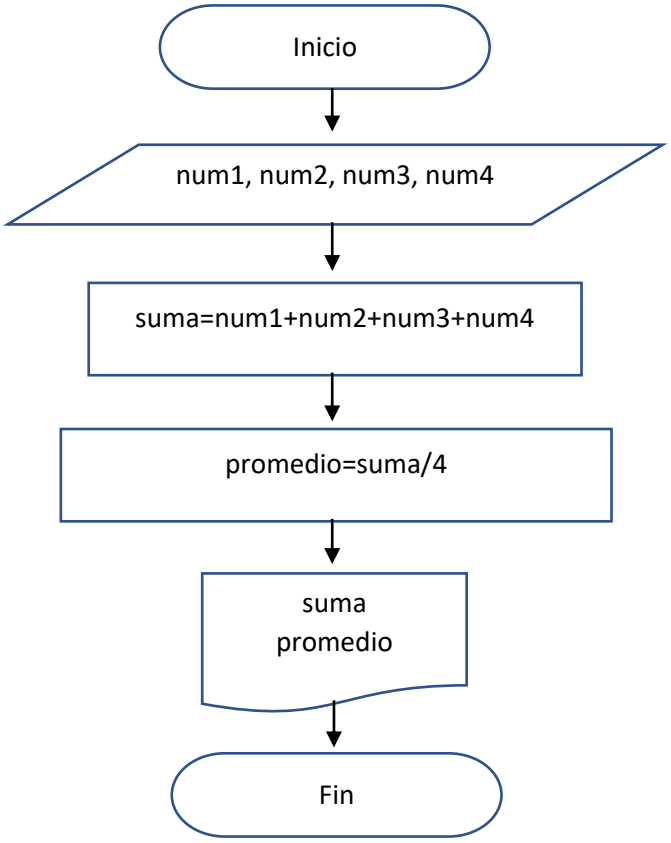
```
1  num1=int(input("Dime el primer número:"))
2  num2=int(input("Dime el segundo número:"))
3  num3=int(input("Dime el tercer número:"))
4  num4=int(input("Dime el cuarto número"))
5  suma=num1+num2
6  producto=num3*num4
7  print("La suma del primer y segundo numero es")
8  print(suma)
9  print("El producto del tercer y cuarto numero es")
10 print(producto)
11
```

Vamos a ejecutar:


IDLE Shell 3.11.1
File Edit Shell Debug Options Window Help
>>>
========== RESTART: D:/Curso completo de Python/ejercicio6.py ==========
Dime el primer número:3
Dime el segundo número:5
Dime el tercer número:7
Dime el cuarto número10
La suma del primer y segundo numero es
8
El producto del tercer y cuarto numero es
70
>>>
Ln: 18 Col: 0

Capítulo 8.- Estructura de programación secuencial – 5

Realizar un programa que lea cuatro valores numéricos e informar su suma y promedio.



Vamos a realizar el programa con el nombre de Ejercicio7.py.

ejercicio7.py - D:/Curso completo de Python/ejercicio7.py (3.11.1)

File Edit Format Run Options Window Help

```
num1=int(input("Dime el primer número:"))
num2=int(input("Dime el segundo número:"))
num3=int(input("Dime el tercer número:"))
num4=int(input("Dime el cuarto número:"))
suma=num1+num2+num3+num4
promedio=suma/4
print("La suma de los cuatro numeros es")
print(suma)
print("El promedio de los cuatro numeros es")
print(promedio)
```

Vamos a ejecutar:

IDLE Shell 3.11.1

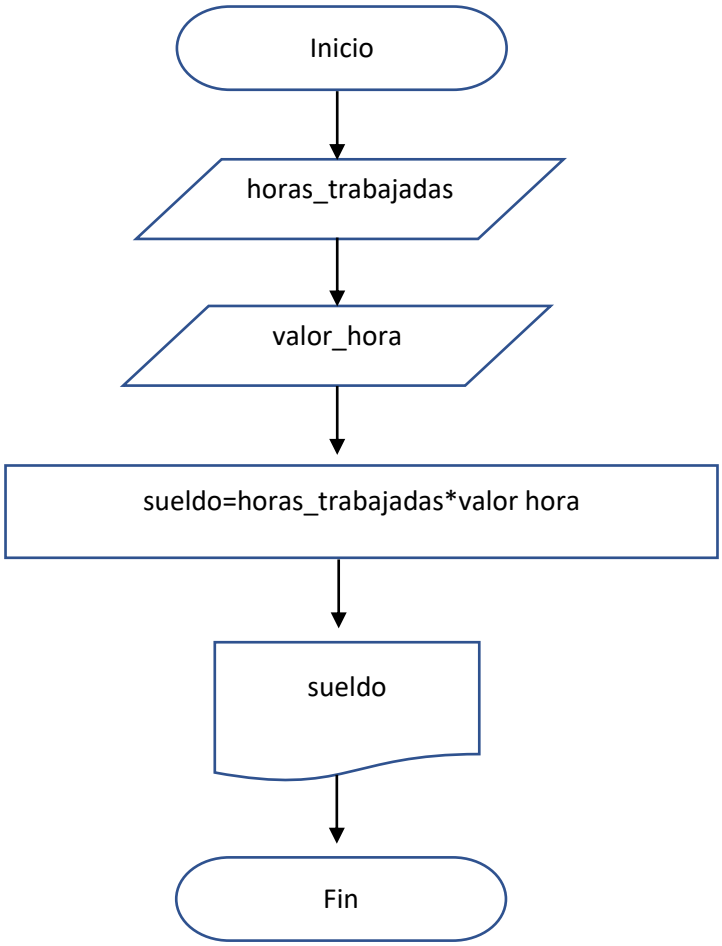
File Edit Shell Debug Options Window Help

```
>>>
========== RESTART: D:/Curso completo de Python/ejercicio7.py ==========
Dime el primer número:2
Dime el segundo número:4
Dime el tercer número:6
Dime el cuarto número:8
La suma de los cuatro numeros es
20
El promedio de los cuatro numeros es
5.0
>>>
```

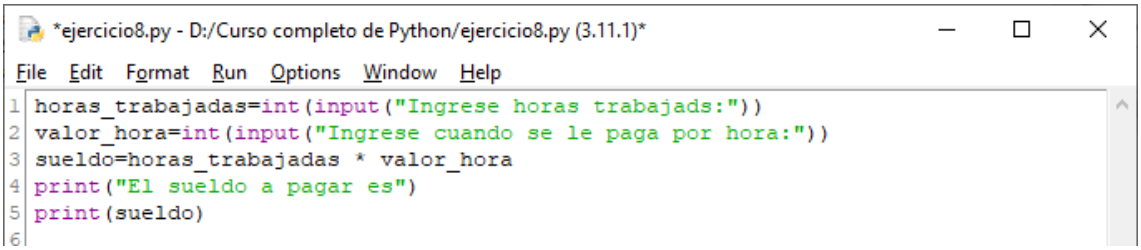
Ln: 38 Col: 0

Capítulo 9.- Estructura de programación secuencial – 6

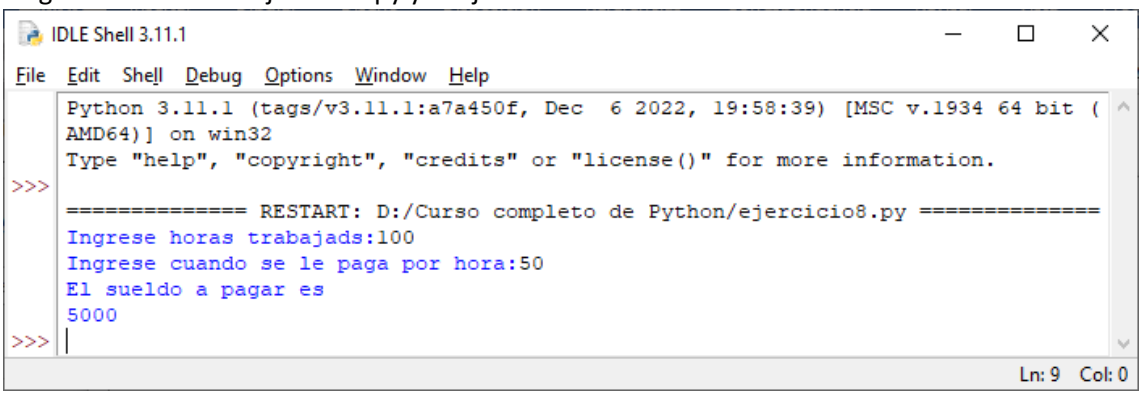
Calcular el sueldo mensual de un operario conociendo la cantidad de horas trabajadas y el valor por hora.



Vamos a la programación:



Lo guardamos como Ejercicio8.py y lo ejecutamos:



Capítulo 10.- Condicionales simples y compuestas – 1

Estructura condicional simple

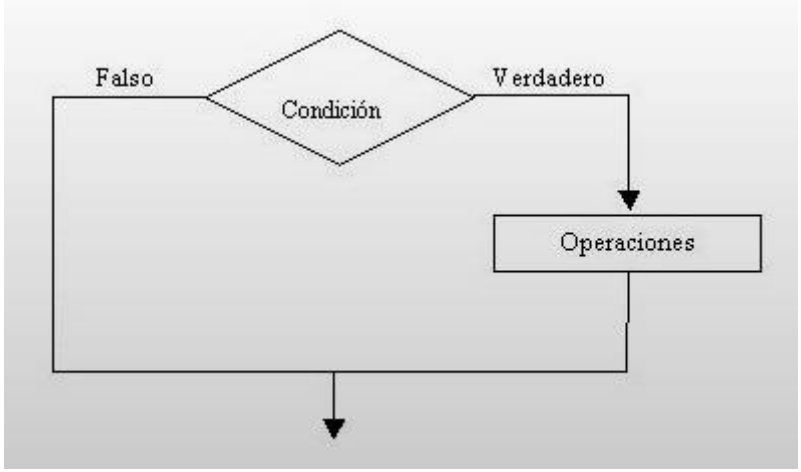
No todos los problemas pueden resolverse empleando estructuras secuenciales. Cuando hay que tomar una decisión aparece las estructuras condicionales.

En nuestra vida diaria se nos presentan situaciones donde debemos decidir.
¿Elijo la carrera A o la carrera B?
¿Me pongo este pantalón?
Para ir al trabajo, ¿Elijo el camino A o el camino B?
Al cursar una carrera, ¿Elijo el turno mañana, tarde o noche?

Es común que en un problema se combinan estructuras secuenciales y condicionales.

Estructura condicional simple.

Cuando se presenta la elección tenemos la opción de realizar una actividad o no realizar ninguna. Representación gráfica de una estructura condicional a utilizar en un diagrama de flujo:



Podemos observar: El rombo representa la condición. Hay dos opciones que se pueden tomar. Si la condición da verdadera se sigue el camino de verdadero, o sea el de la derecha, si la condición da falsa se sigue el camino de la izquierda donde no hay ninguna actividad.

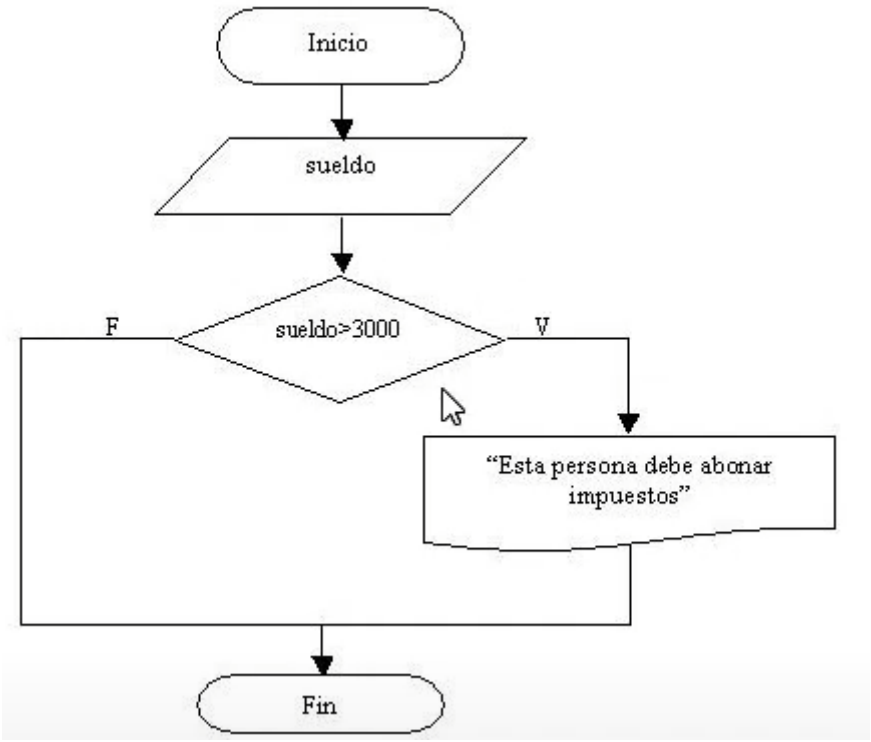
Se trata de una estructura CONDICIONAL SIMPLE porque por el camino de verdadero hay actividades y por el camino de falso no hay actividades.

Por el camino de verdadero pueden existir varias operaciones, entradas y salidas, inclusive ya veremos que puede haber otras estructuras condicionales.

Problema:

Ingresar el sueldo de una persona, si supera los 3000 dólares mostrar un mensaje en pantalla indicando que debe abonar impuestos.

Diagrama de flujo:



Vamos a la programación:

```
*ejercicio9.py - D:/Curso completo de Python/ejercicio9.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 sueldo=int(input("Ingrese el sueldo:"))
2 if sueldo>3000:
3     print("Esta persona debe pagar impuestos")
4     print("Fin")
5
```

Lo guardamos con el nombre de Ejercicio9.py y lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio9.py ==============
Ingrese el sueldo:7000
Esta persona debe pagar impuestos
Fin
>>>
============== RESTART: D:/Curso completo de Python/ejercicio9.py ==============
Ingrese el sueldo:2000
>>>
                                                                  Ln: 11  Col: 0
```

Modificamos el programa:

```
*ejercicio9.py - D:/Curso completo de Python/ejercicio9.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 sueldo=int(input("Ingrese el sueldo:"))
2 if sueldo>3000:
3     print("Esta persona debe pagar impuestos")
4     print("Fin")
5 print("Fin del programa")
6
```

Vamos a ejecutar:

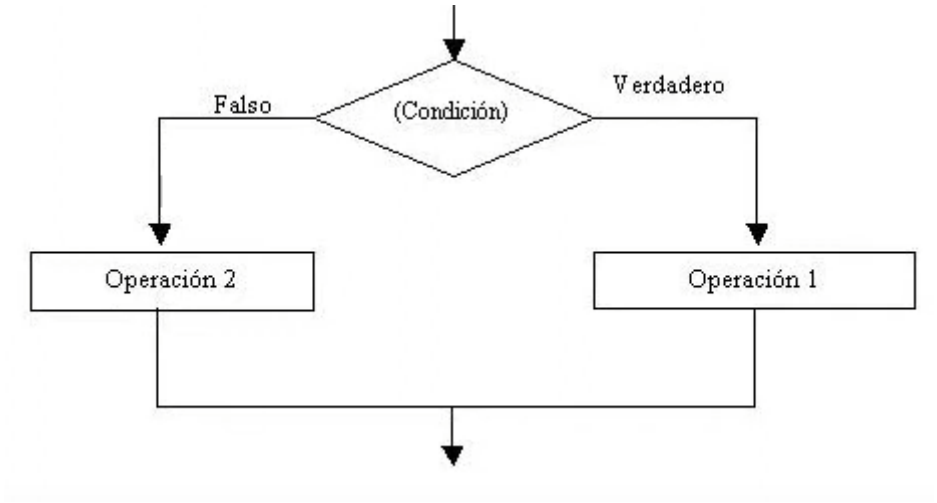
```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio9.py ==============
Ingrese el sueldo:7000
Esta persona debe pagar impuestos
Fin
Fin del programa
>>>
============== RESTART: D:/Curso completo de Python/ejercicio9.py ==============
Ingrese el sueldo:2000
Fin del programa
>>>
                                                                    Ln: 13  Col: 0
```

Las líneas que se encuentras después del if y están tabuladas se ejecutan si la condición se cumple, en teste ejemplo las líneas 3 y 4, en cambio la línea 5 que se encuentra al principio de la línea se ejecuta tanto si se cumple la condición como si no se cumple la condición.

Capítulo 11.- Condicionales simples y compuestas – 2

Cuando se presenta la elección tenemos la opción de realizar una actividad u otra. Es decir tenemos actividades por el verdadero y por el falso de la condición. Lo más importante que hay que tener en cuenta que se realizan las actividades de la rama del verdadero o las del falso, NUNCA se realizan las actividades de las dos ramas.

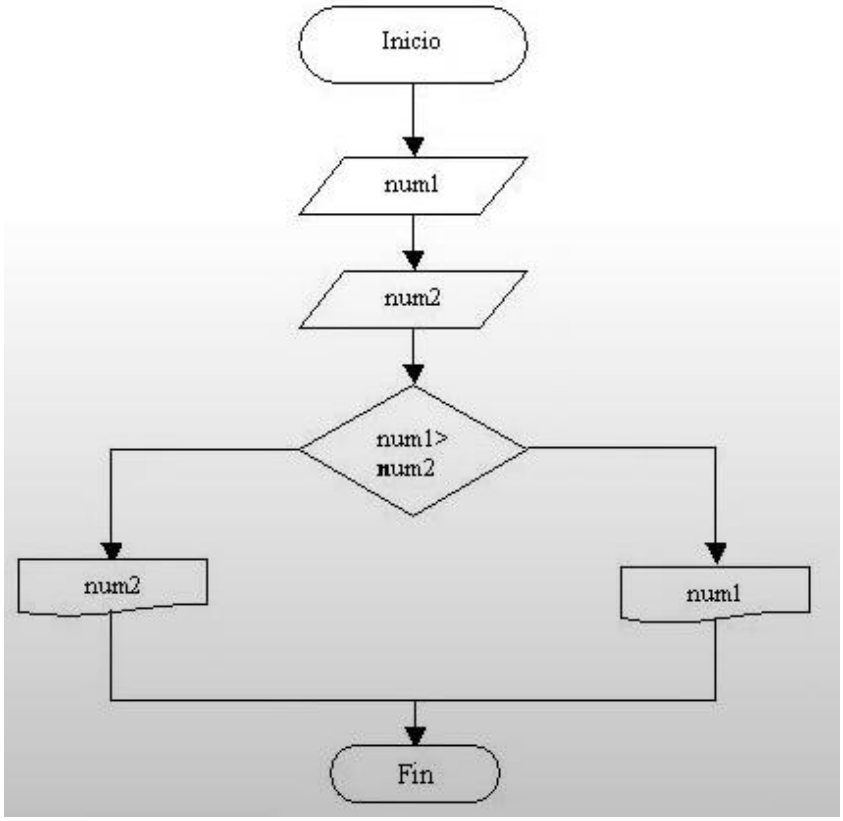
Representación gráfica:



Problema:

Realizar un programa que solicite ingresar dos números distintos y muestre en pantalla el mayor de ellos:

Diagrama de flujo:



Vamos a realizar el programa:

```
*ejercicio10.py - D:/Curso completo de Python/ejercicio10.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
num1=int(input("Ingrese primer valor:"))
num2=int(input("Ingrese segundo valor:"))
if num1>num2:
    print(num1)
else:
    print(num2)
```

Lo guardamos con el nombre Ejercicio10.py y lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio10.py =============
Ingrese primer valor:10
Ingrese segundo valor:5
10
>>>
============== RESTART: D:/Curso completo de Python/ejercicio10.py =============
Ingrese primer valor:2
Ingrese segundo valor:67
67
>>>
```

si queremos agregar un mensaje:

```
*ejercicio10.py - D:/Curso completo de Python/ejercicio10.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 num1=int(input("Ingrese primer valor:"))
2 num2=int(input("Ingrese segundo valor:"))
3 print("El valor mayor es")
4 if num1>num2:
5     print(num1)
6 else:
7     print(num2)
8
```

Correcto

Esto es lo mismo que:

```
*ejercicio10.py - D:/Curso completo de Python/ejercicio10.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 num1=int(input("Ingrese primer valor:"))
2 num2=int(input("Ingrese segundo valor:"))
3 if num1>num2:
4     print("El valor mayor es")
5     print(num1)
6 else:
7     print("El valor mayor es")
8     print(num2)
9
```

No correcto

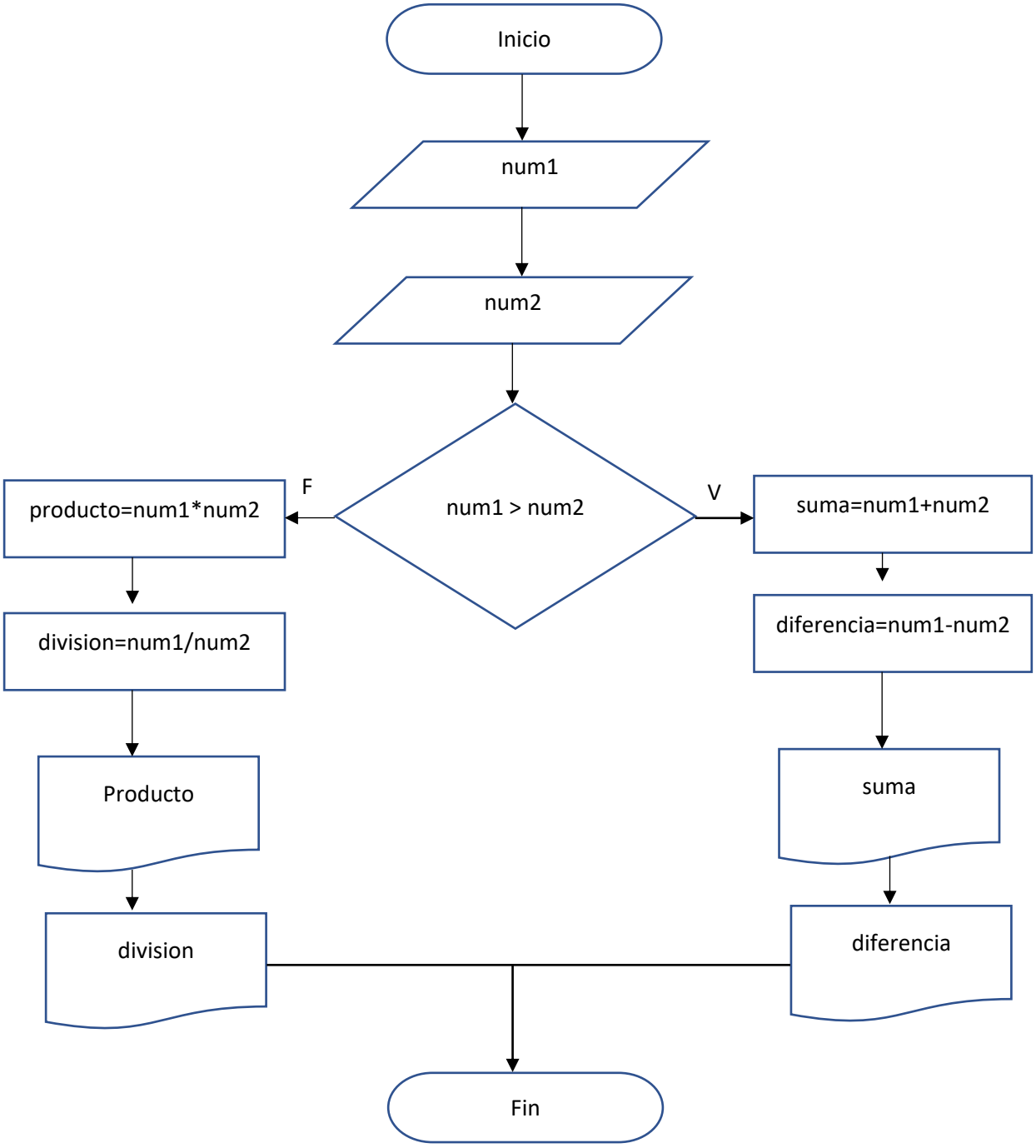
Como las líneas 4 y 7 se repiten en las distintas condiciones y no depende de la condición lo correcto es realizarlo como se encuentra en el ejemplo anterior.

Capítulo 12.- Condicionales simples y compuestas – 3

Problema propuesto:

Realice un programa que solicite la carga por teclado de dos números, si el primero es mayor al segundo informar de su suma y diferencia, en caso contrario informar del producto y la división del primero con respecto al segundo.

Vamos a realizar el diagrama de flujo:



Vamos a la programación:

```
*ejercicio11.py - D:/Curso completo de Python/ejercicio11.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
num1=int(input("Ingrese el primer valor:"))
num2=int(input("Ingrese el segundo valor:"))
if num1>num2:
    suma=num1+num2
    diferencia=num1-num2
    print("La sumna de los valores es")
    print(suma)
    print("La diferencia es")
    print(diferencia)
else:
    producto=num1*num2
    division=num1/num2
    print("El producto es")
    print(producto)
    print("La división es")
    print(division)
```

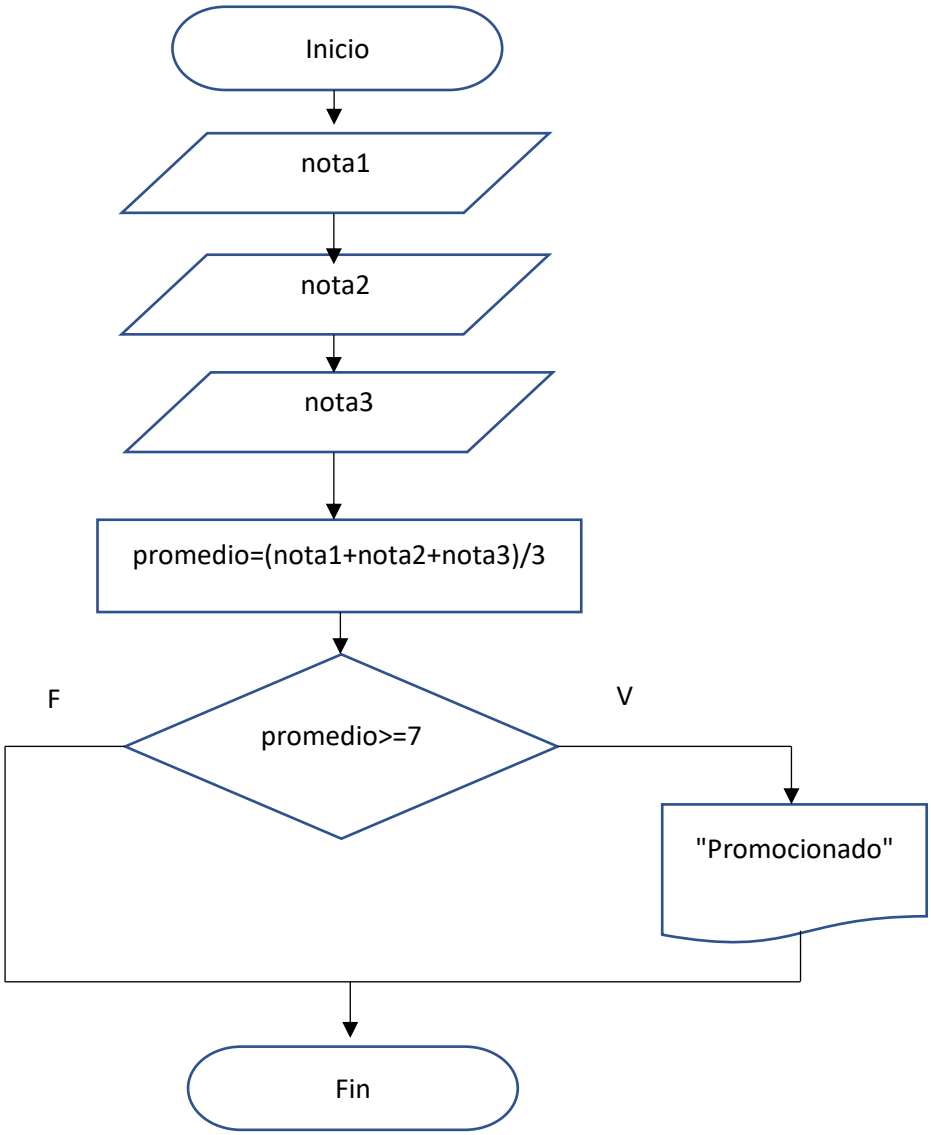
Lo guardamos con el nombre Ejercicio11.py y lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio11.py =============
Ingrese el primer valor:10
Ingrese el segundo valor:5
La sumna de los valores es
15
La diferencia es
5
>>>
============== RESTART: D:/Curso completo de Python/ejercicio11.py =============
Ingrese el primer valor:50
Ingrese el segundo valor:100
El producto es
5000
La división es
0.5
>>>
                                                                Ln: 19  Col: 0
```

Capítulo 13.- Condicionales simples y compuestas – 4

Problema propuesto:

Se ingresan tres notas de un alumno, si el promedio es mayor o igual a siete mostrar un mensaje "Promocionado".



Vamos a realizar el programa:

ejercicio12.py - D:/Curso completo de Python/ejercicio12.py (3.11.1)

File Edit Format Run Options Window Help

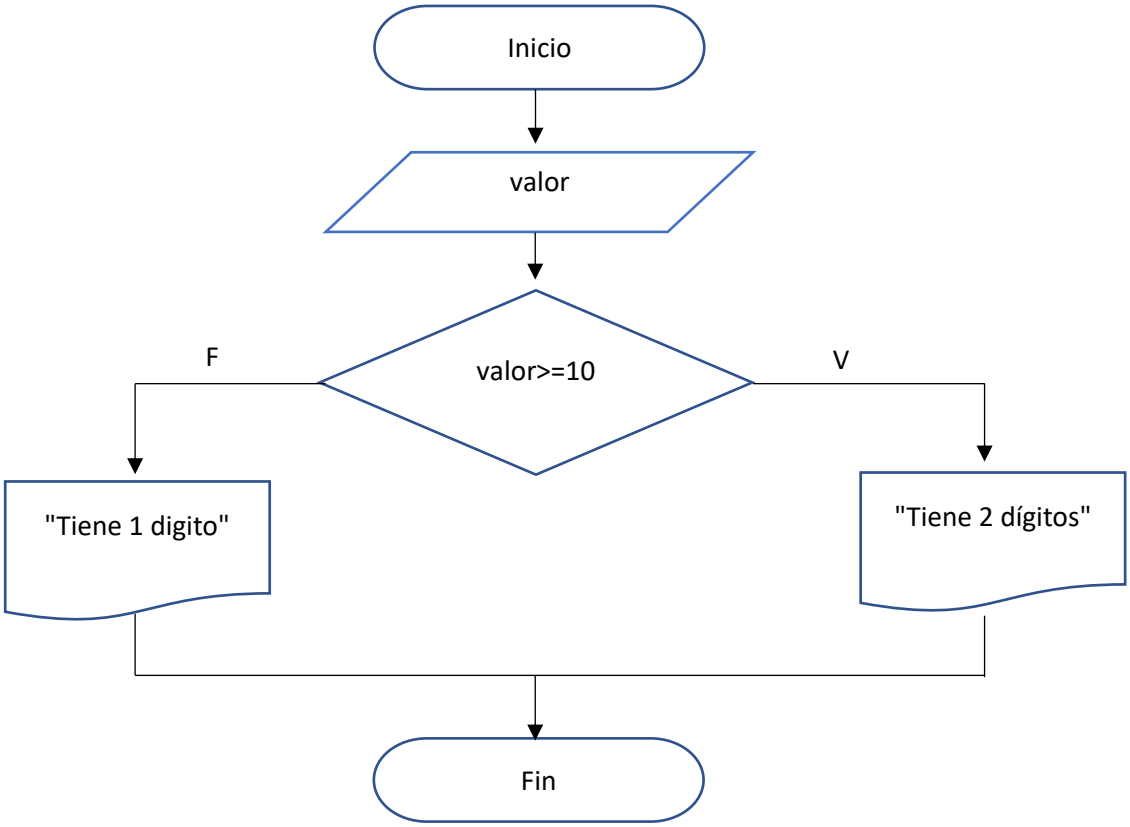
```
nota1=int(input("Ingrese primera nota:"))
nota2=int(input("Ingrese segunda nota:"))
nota3=int(input("Ingrese tercera nota:"))
promedio=(nota1+nota2+nota3)/3
if promedio >=7:
    print("Promocionado")
```

Lo guardamos con el nombre de Ejercicio12.py y lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio12.py =============
Ingrese primera nota:7
Ingrese segunda nota:8
Ingrese tercera nota:6
Promocionado
>>>
============== RESTART: D:/Curso completo de Python/ejercicio12.py =============
Ingrese primera nota:5
Ingrese segunda nota:6
Ingrese tercera nota:4
>>>
```

Capítulo 14.- Condicionales simples y compuestas – 5

Se ingresa por teclado un número positivo de uno o dos dígitos (1..99) mostrar un mensaje indicando si el número tiene uno o dos dígitos.
(Tener en cuenta que condición debe cumplirse para tener dos dígitos un número entero)
Vamos a diagrama de flujo:



Vamos a realizar el programa:

ejercicio13.py - D:/Curso completo de Python/ejercicio13.py (3.11.1)

File Edit Format Run Options Window Help

```
valor=int(input("Ingrese un valor comprendido entre 1 y 99:"))
if valor>=10:
    print("Tiene 2 dígitos")
else:
    print("Tiene 1 digito")
```

Lo guardamos con el nombre de Ejercicio13.py y lo ejecutamos:

IDLE Shell 3.11.1

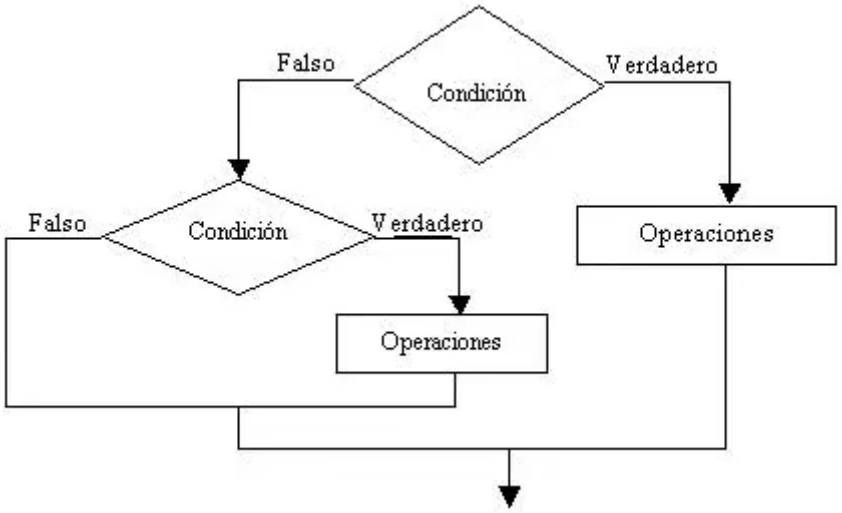
File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio13.py =============
Ingrese un valor comprendido entre 1 y 99:7
Tiene 1 digito
>>>
============== RESTART: D:/Curso completo de Python/ejercicio13.py =============
Ingrese un valor comprendido entre 1 y 99:22
Tiene 2 dígitos
>>>
```

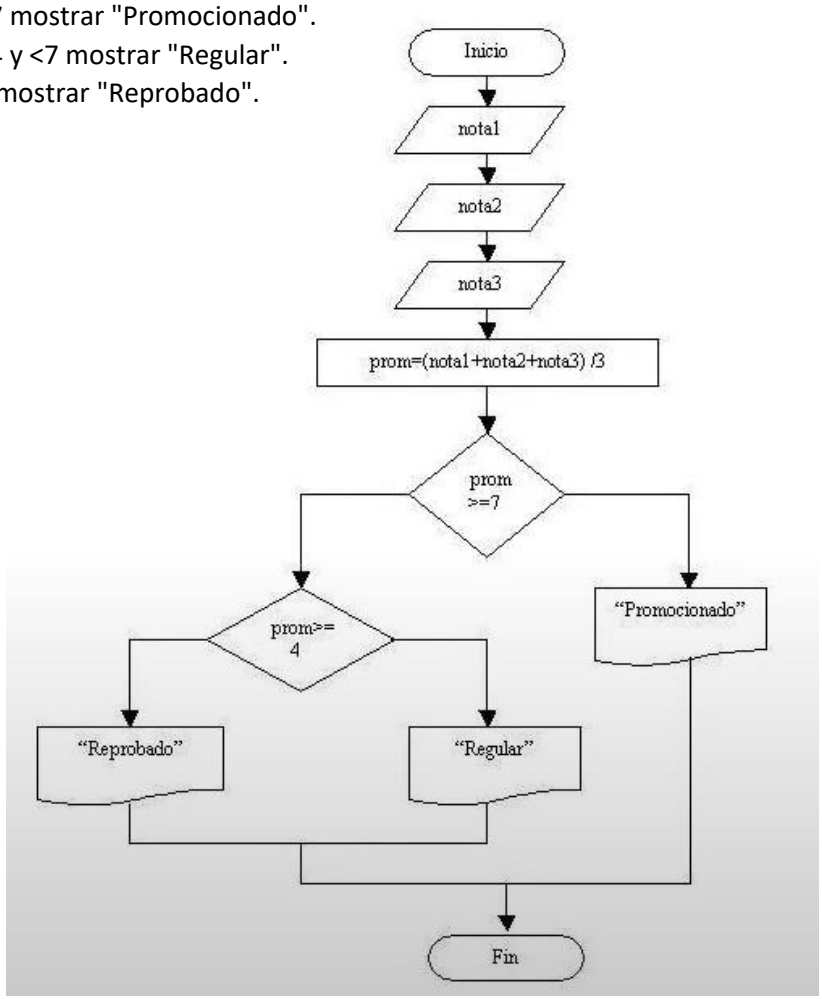
Ln: 11 Col: 0

Capítulo 15.- Estructuras condicionales anidadas – 1

Estamos en presencia de una estructura condicional anidada cuando por la rama de verdadero y/o el falso de una estructura condicional hay otra estructura condicional.
Representación gráfica:



Problema:
Confeccionar un programa que pida por teclado tres notas de un alumno, calcule el promedio e imprima alguno de estos mensajes:
Si el promedio es >=7 mostrar "Promocionado".
Si el promedio es >=4 y <7 mostrar "Regular".
Si el promedio es <4 mostrar "Reprobado".



Vamos al programa:

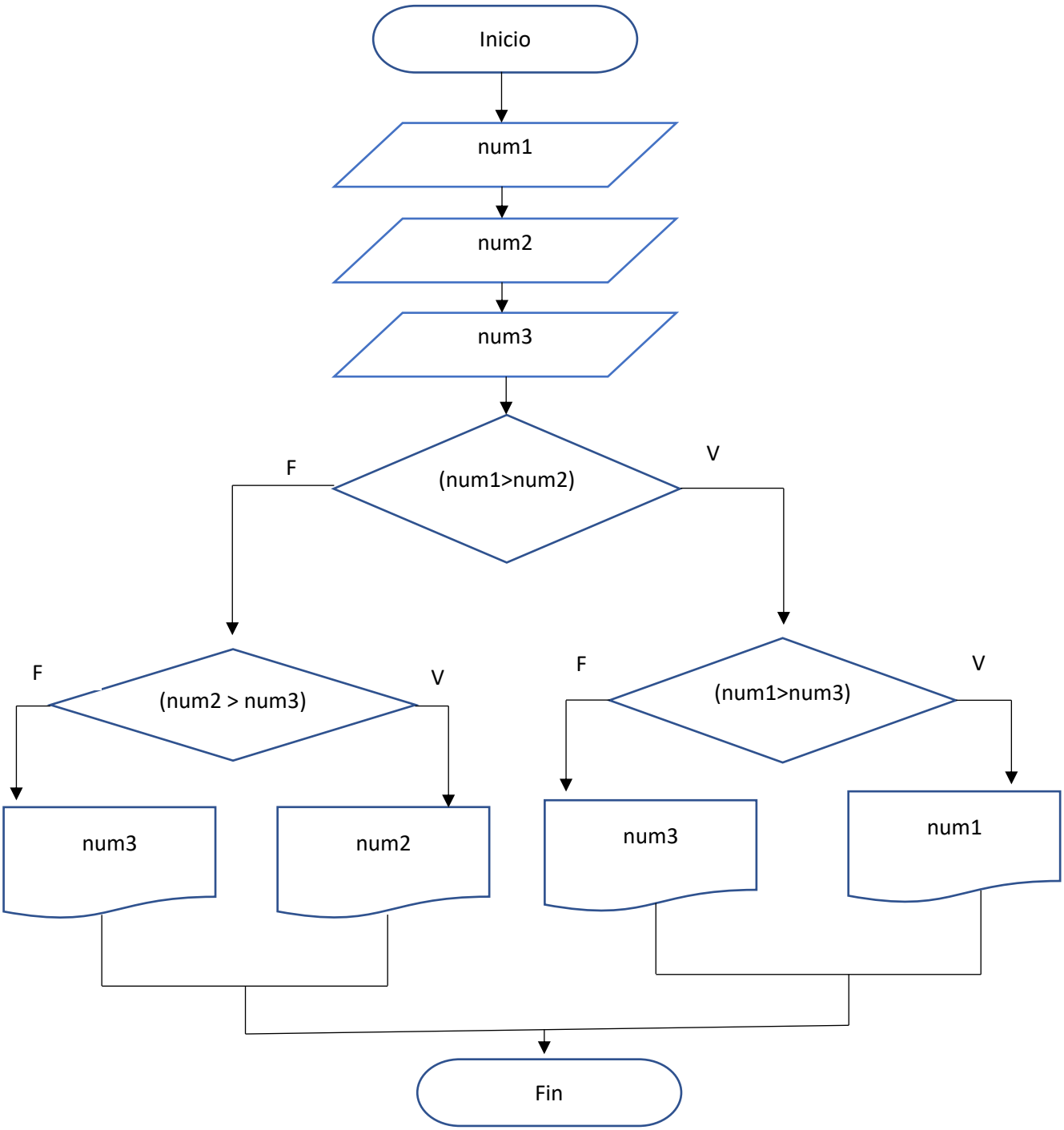
```
*ejercicio14.py - D:/Curso completo de Python/ejercicio14.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1  notal=int(input("Ingrese el primer valor:"))
2  nota2=int(input("Ingrese el segundo valor valor:"))
3  nota3=int(input("Ingrese el tercer valor:"))
4  prom=(notal+nota2+nota3)/3
5  if prom>=7:
6      print("Promocionado")
7  else:
8      if prom>=4:
9          print("Regular")
10     else:
11         print("Reprobado")
12
```

Guarde el programa con el nombre de Ejercicio14.py y ejecútelo.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
   Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
   AMD64)] on win32
   Type "help", "copyright", "credits" or "license()" for more information.
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio14.py =============
   Ingrese el primer valor:7
   Ingrese el segundo valor valor:8
   Ingrese el tercer valor:9
   Promocionado
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio14.py =============
   Ingrese el primer valor:5
   Ingrese el segundo valor valor:6
   Ingrese el tercer valor:7
   Regular
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio14.py =============
   Ingrese el primer valor:2
   Ingrese el segundo valor valor:3
   Ingrese el tercer valor:4
   Reprobado
>>>
```

Capítulo 16.- Estructuras condicionales anidadas – 2

Se carga por teclado tres números distintos. Mostrar por pantalla el mayor de ellos.



Vamos a programar:

```
*ejercicio15.py - D:/Curso completo de Python/ejercicio15.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
num1=int(input("Ingrese el primer valor:"))
num2=int(input("Ingrese el segundo valor:"))
num3=int(input("Ingrese el tercer valor:"))
if num1>num2:
    if num1>num3:
        print(num1)
    else:
        print(num3)
else:
    if num2>num3:
        print(num2)
    else:
        print(num3)
```

Lo guardamos con el nombre Ejercicio15.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio15.py =============
Ingrese el primer valor:7
Ingrese el segundo valor:3
Ingrese el tercer valor:9
9
>>>
============== RESTART: D:/Curso completo de Python/ejercicio15.py =============
Ingrese el primer valor:9
Ingrese el segundo valor:7
Ingrese el tercer valor:3
9
>>>
============== RESTART: D:/Curso completo de Python/ejercicio15.py =============
Ingrese el primer valor:3
Ingrese el segundo valor:9
Ingrese el tercer valor:7
9
>>>
```

Vamos a modificar el código:

```
*ejercicio15.py - D:/Curso completo de Python/ejercicio15.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
num1=int(input("Ingrese el primer valor:"))
num2=int(input("Ingrese el segundo valor:"))
num3=int(input("Ingrese el tercer valor:"))
print("El valor mayor de los tres valores ingresados es")
if num1>num2:
    if num1>num3:
        print(num1)
    else:
        print(num3)
else:
    if num2>num3:
        print(num2)
    else:
        print(num3)
```

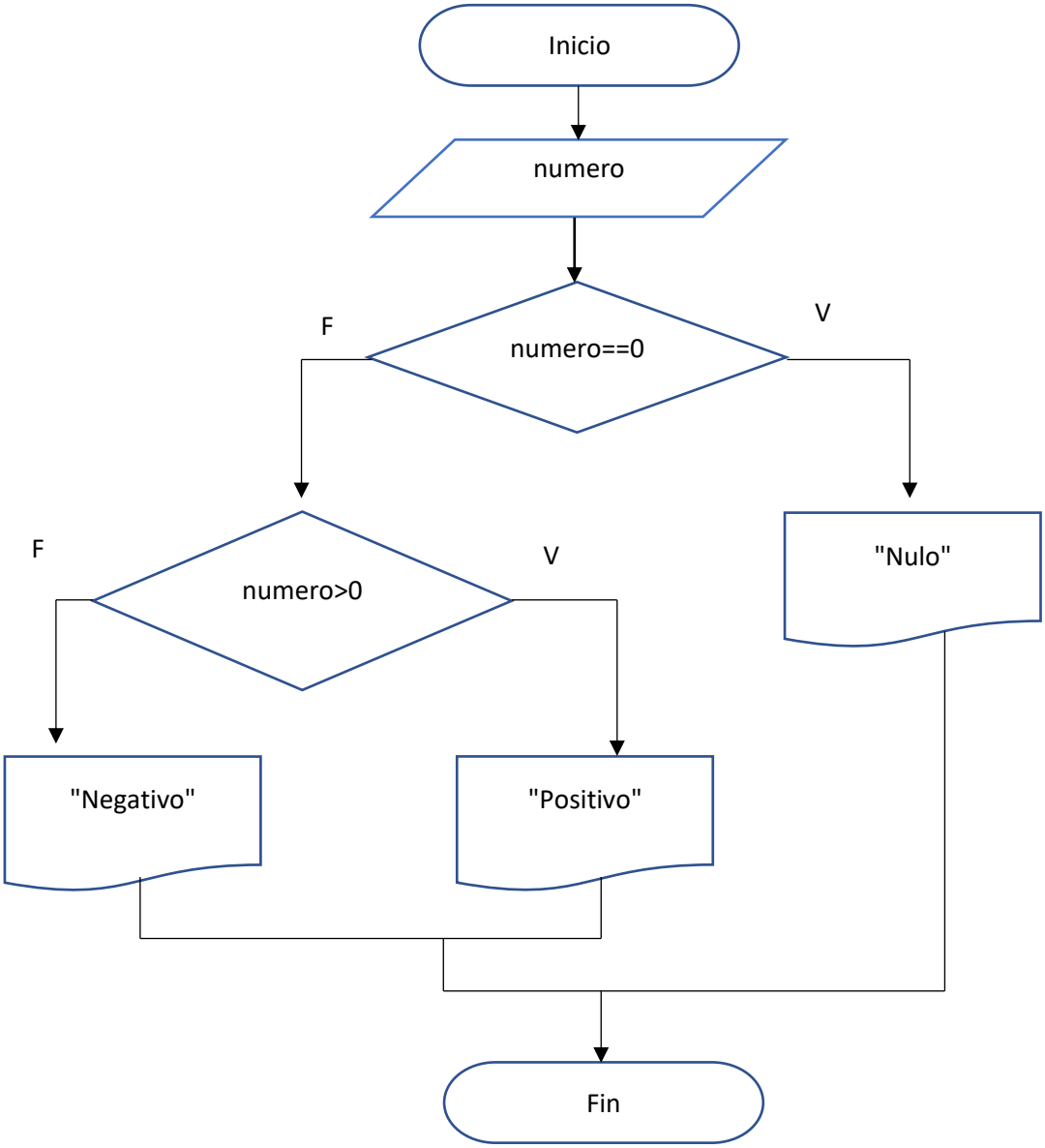
Vamos a ejecutar de nuevo:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio15.py =============
Ingrese el primer valor:10
Ingrese el segundo valor:40
Ingrese el tercer valor:5
El valor mayor de los tres valores ingresados es
40
>>>
```

Capítulo 17.- Estructuras condicionales anidadas – 3

Problema propuesto:

Se ingresa por teclado un valor entero, mostrar una leyenda que indique si el número es positivo, negativo o nulo (es decir cero).



Vamos a programar:

ejercicio16.py - D:/Curso completo de Python/ejercicio16.py (3.11.1)

File Edit Format Run Options Window Help

```
num=int(input("Ingrese un valor:"))
if num==0:
    print("Ingresó un valor nulo")
else:
    if num>0:
        print("El número ingresado es positivo")
    else:
        print("El número ingresado es negativo")
```

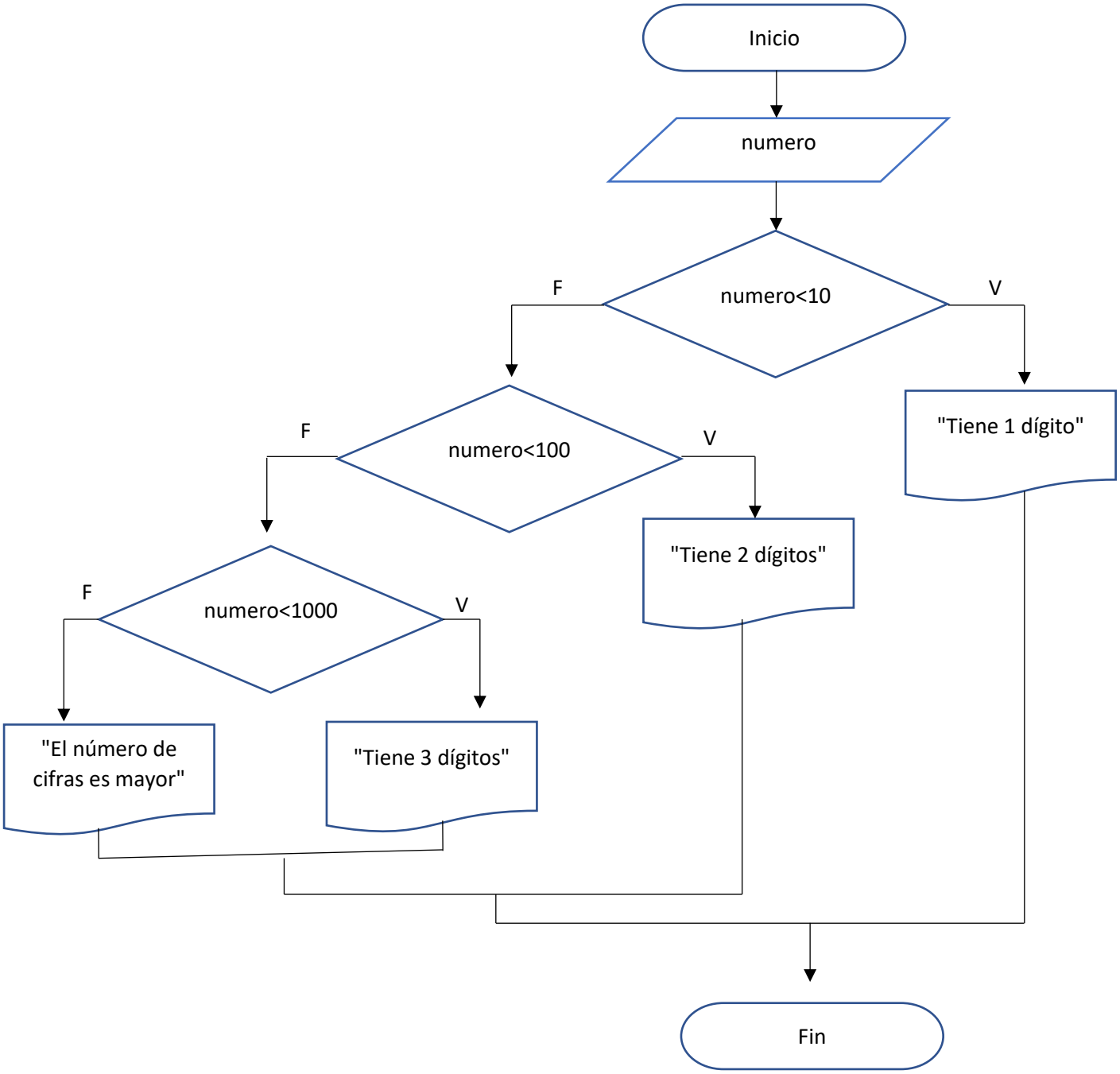
Lo guardamos con el nombre de Ejercicio16.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio16.py =============
Ingrese un valor:10
El número ingresado es positivo
>>>
============== RESTART: D:/Curso completo de Python/ejercicio16.py =============
Ingrese un valor:-5
El número ingresado es negativo
>>>
============== RESTART: D:/Curso completo de Python/ejercicio16.py =============
Ingrese un valor:0
Ingresó un valor nulo
>>>
```

Capítulo 18.- Estructuras condicionales anidadas – 4

Problema propuesto:

Confeccionar un programa que permita cargar un número entero positivo de hasta 3 cifras y muestre un mensaje indicando si tiene 1, 2 o 3 cifras. Mostrar un mensaje de error si el número de cifras es mayor.



Vamos a programar:

```
*ejercicio17.py - D:/Curso completo de Python/ejercicio17.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
numero=int(input("Ingrese un número entre 1 y 999:"))
if numero<10:
    print("Tiene 1 dígito")
else:
    if numero<100:
        print("tiene 2 dígitos")
    else:
        if numero<1000:
            print("Tiene 3 digitos")
        else:
            print("Error en la entrada de datos")
```

Guarda el programa con el nombre de Ejercicio17.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio17.py =============
Ingrese un número entre 1 y 999:4
Tiene 1 dígito
>>>
============== RESTART: D:/Curso completo de Python/ejercicio17.py =============
Ingrese un número entre 1 y 999:89
tiene 2 dígitos
>>>
============== RESTART: D:/Curso completo de Python/ejercicio17.py =============
Ingrese un número entre 1 y 999:888
Tiene 3 digitos
>>>
============== RESTART: D:/Curso completo de Python/ejercicio17.py =============
Ingrese un número entre 1 y 999:45455445
Error en la entrada de datos
>>>
```

Si queremos que nos diga cuando llega al final del programa:

```
*ejercicio17.py - D:/Curso completo de Python/ejercicio17.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
numero=int(input("Ingrese un número entre 1 y 999:"))
if numero<10:
    print("Tiene 1 dígito")
else:
    if numero<100:
        print("tiene 2 dígitos")
    else:
        if numero<1000:
            print("Tiene 3 digitos")
        else:
            print("Error en la entrada de datos")
print("Fin del programa")
```

No importa el numero que le asignemos siempre imprimirá el mensaje "Fin del programa".

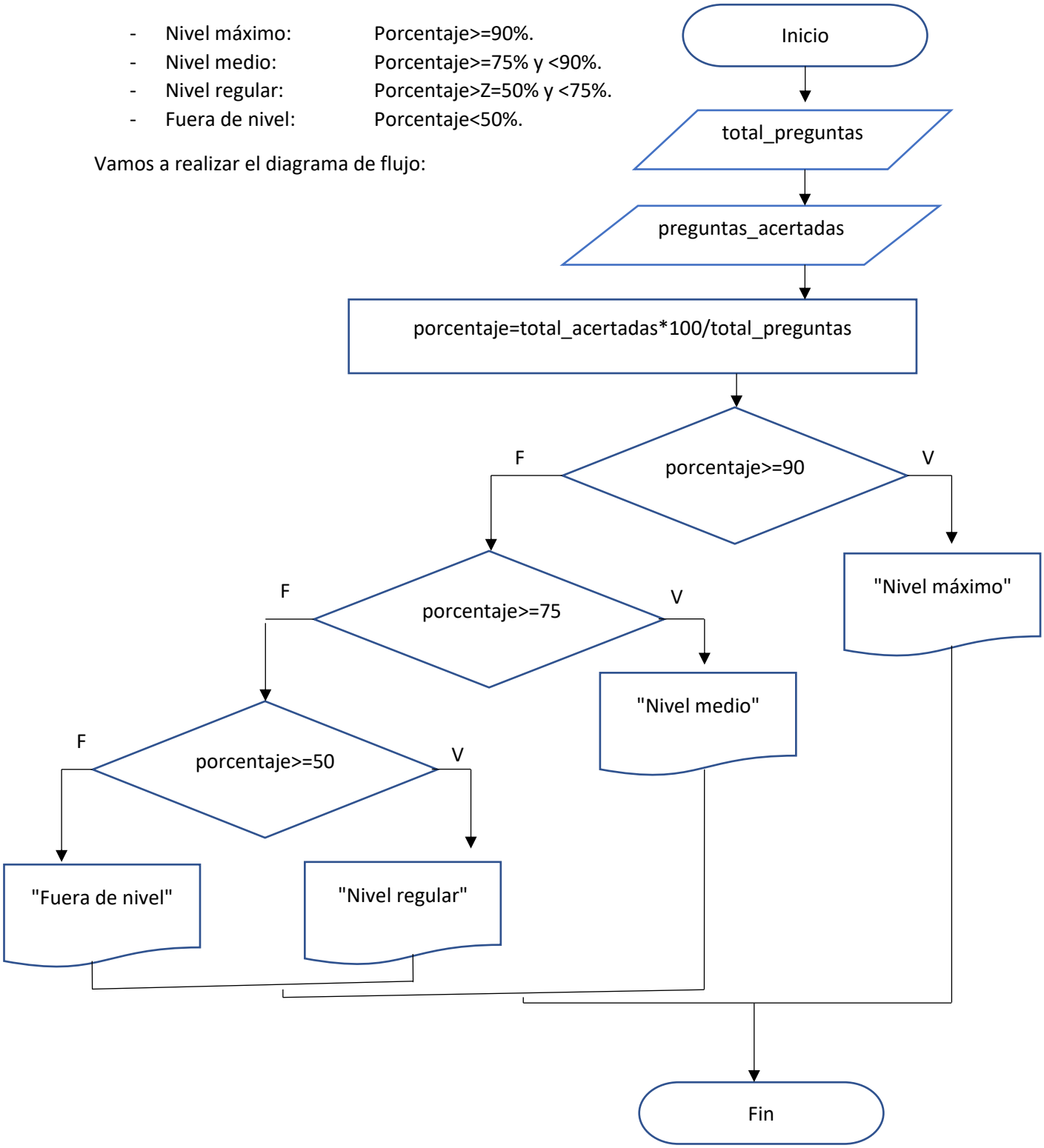
Capítulo 19.- Estructuras condicionales anidadas – 5

Problema propuesto:

Un postulante a un empleo, realiza un test de capacitación, se obtuvo la siguiente información: cantidad total de preguntas que se le realizaron y la cantidad de preguntas que contestó correctamente. Se pide confeccionar un programa que ingrese los dos datos por teclado e informe el nivel del mismo según el porcentaje de respuestas correctas que ha obtenido, y sabiendo que:

- Nivel máximo: Porcentaje>=90%.
- Nivel medio: Porcentaje>=75% y <90%.
- Nivel regular: Porcentaje>Z=50% y <75%.
- Fuera de nivel: Porcentaje<50%.

Vamos a realizar el diagrama de flujo:



Vamos a programar:

ejercicio18.py - D:/Curso completo de Python/ejercicio18.py (3.11.1)

File Edit Format Run Options Window Help

```
total_preguntas=int(input("Número de preguntas que tiene en text:"))
preguntas_acertadas=int(input("Número de respuestas acertadas:"))
porcentaje=preguntas_acertadas*100/total_preguntas
if porcentaje>=90:
    print("Nivel máximo")
else:
    if porcentaje>=75:
        print("Nivel medio")
    else:
        if porcentaje>=50:
            print("Nivel regular")
        else:
            print("Fuera de nivel")
```

Vamos a guardar el programa con el nombre Ejercicio18.py y a continuación ejecutarlo:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio18.py =============
Número de preguntas que tiene en text:100
Número de respuestas acertadas:95
Nivel máximo
>>>
============== RESTART: D:/Curso completo de Python/ejercicio18.py =============
Número de preguntas que tiene en text:100
Número de respuestas acertadas:80
Nivel medio
>>>
============== RESTART: D:/Curso completo de Python/ejercicio18.py =============
Número de preguntas que tiene en text:100
Número de respuestas acertadas:65
Nivel regular
>>>
============== RESTART: D:/Curso completo de Python/ejercicio18.py =============
Número de preguntas que tiene en text:100
Número de respuestas acertadas:35
Fuera de nivel
>>>
```

Capítulo 20.- Condiciones compuestas con operadores lógicos – 1

Hasta ahora hemos visto algunos operadores:

relacionales: (>, <, >=, <=, ==, !=)

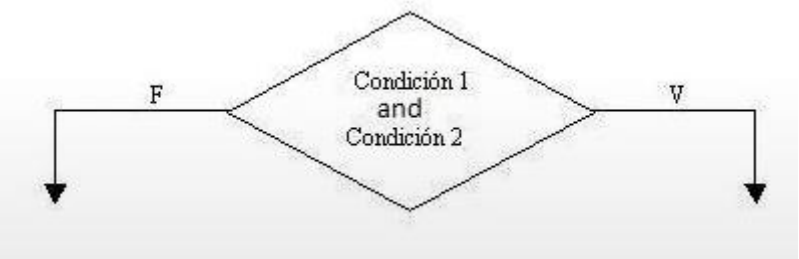
matemáticos: (+, -, *, /, //, **, %)

Pero nos están faltando otros operadores imprescindibles:

lógicos (and y or)

Estos dos operadores se emplean fundamentalmente e las estructuras condicionales par agrupar varias condiciones simples.

Operador and



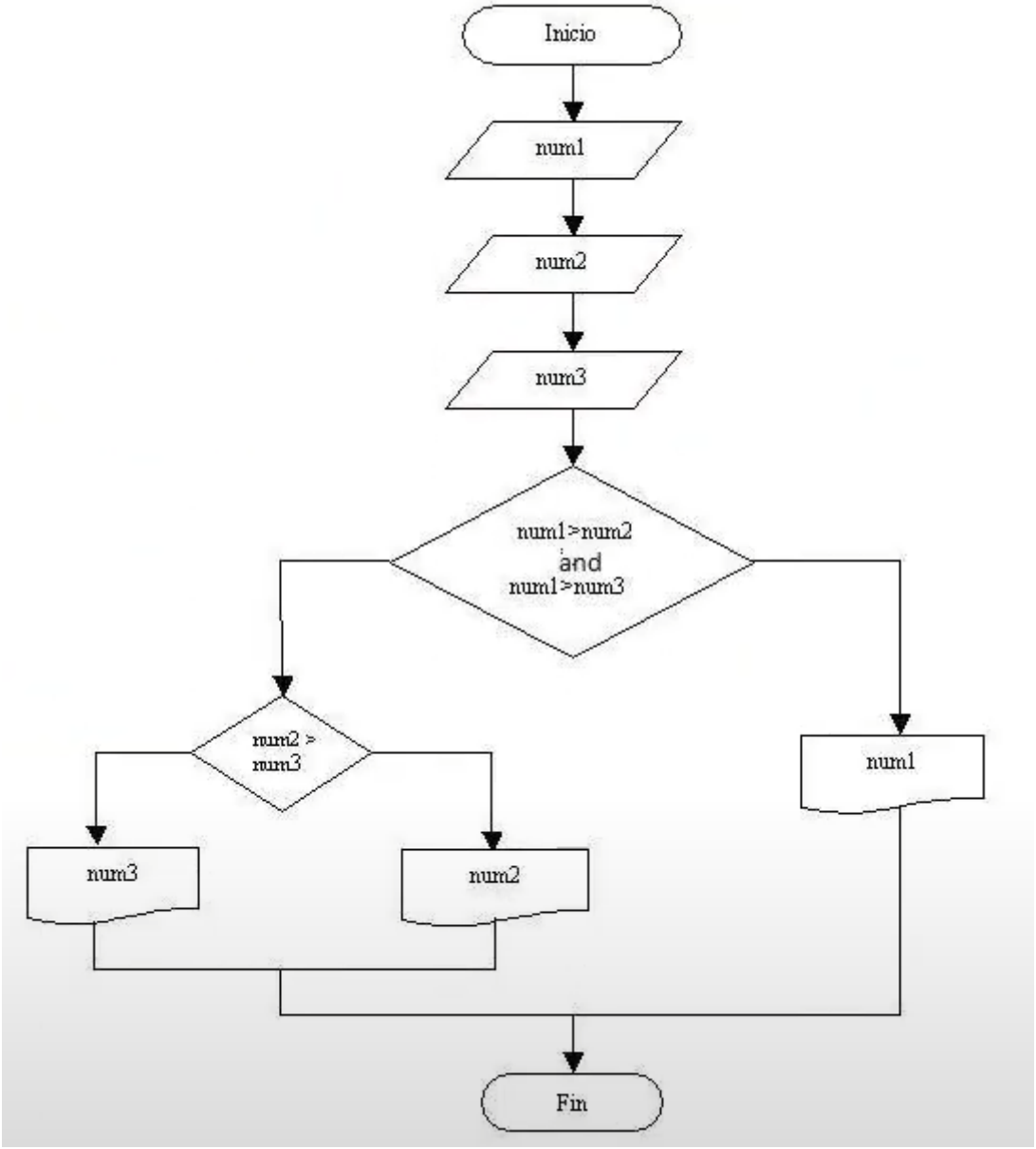
Traduciendo se lo lee como "Y". Si la Condición 1 es verdadera Y la condición 2 es verdadera luego ejecuta la rama del verdadero.

Cuando vinculamos dos condiciones con el operador "and", las dos condiciones deber de ser verdaderas para que el resultado de la condición compuesta de verdadero y continúe por la rama del verdadero de la estructura condicional.

La utilización de operadores lógicos permiten en muchos casos plantear algoritmos más cortos y comprensibles.

Problema:

Confeccionar un programa que lea por teclado tres números enteros distintos y nos muestre el mayor.



Vamos a realizar el programa:

ejercicio19.py - D:/Curso completo de Python/ejercicio19.py (3.11.1)

File Edit Format Run Options Window Help

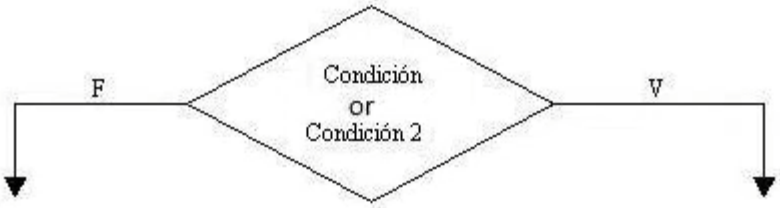
```
num1=int(input("Ingrese primer valor:"))
num2=int(input("Ingrese segundo valor:"))
num3=int(input("Ingrese tercer valor:"))
print("El valor mayor es")
if num1>num2 and num1>num3:
    print(num1)
else:
    if num2>num3:
        print(num2)
    else:
        print(num3)
```

Guardamos el programa con el nombre de Ejercicio19.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio19.py =============
Ingrese primer valor:10
Ingrese segundo valor:40
Ingrese tercer valor:2
El valor mayor es
40
>>>
============== RESTART: D:/Curso completo de Python/ejercicio19.py =============
Ingrese primer valor:100
Ingrese segundo valor:12
Ingrese tercer valor:34
El valor mayor es
100
>>>
============== RESTART: D:/Curso completo de Python/ejercicio19.py =============
Ingrese primer valor:10
Ingrese segundo valor:30
Ingrese tercer valor:120
El valor mayor es
120
>>>
```

Capítulo 21.- Condiciones compuestas con operadores lógicos – 2

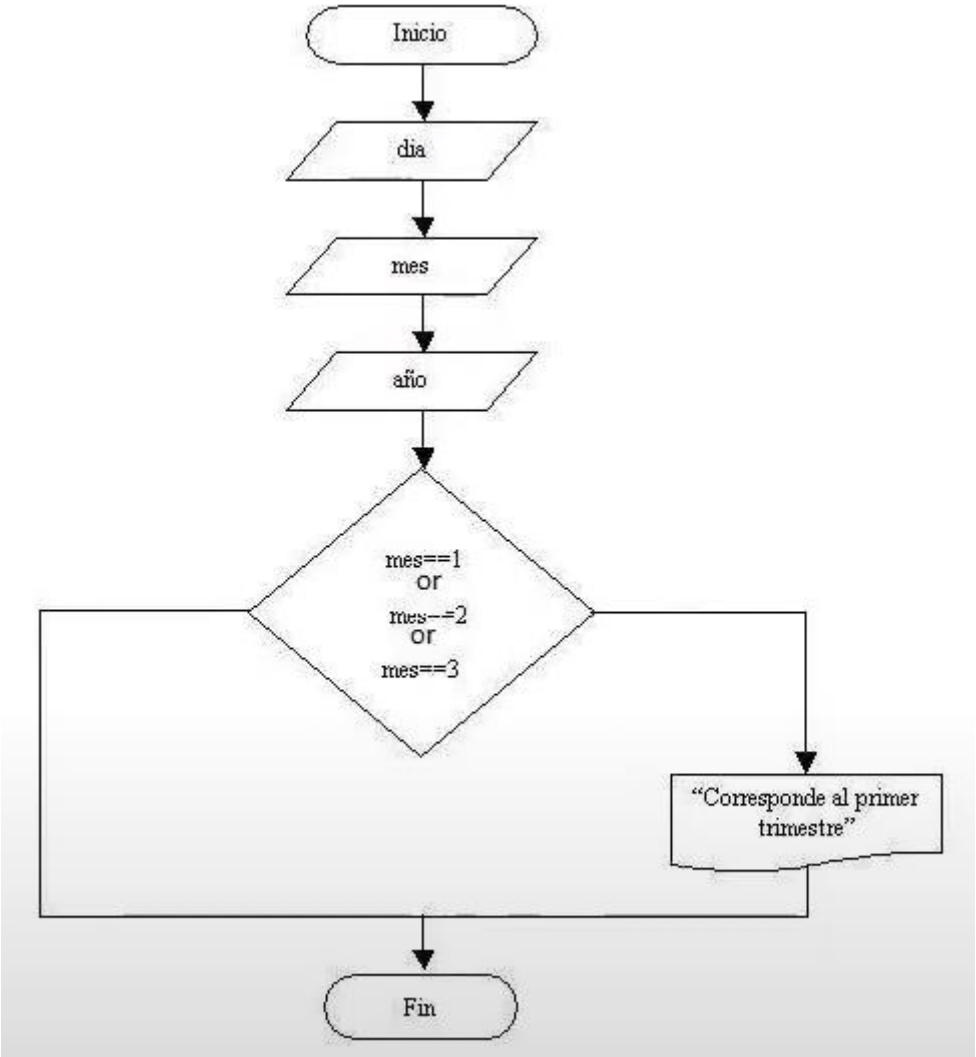
Operador or



Traducido se lo lee como "O". Si la condición 1 o la condición 2 es Verdadera, luego ejecuta la rama de Verdadero.

Cuando vinculamos dos o más condiciones con el operador "or", con que una de las dos condiciones sea Verdadera alcanza para que el resultado de la condición compuesta sea Verdadero.

Problema:
Se carga una fecha (día, mes y año) por teclado. Mostrar un mensaje de error si corresponde al primer trimestre del año (enero, febrero o marzo) Cargar por teclado el valor numérico del día, mes y año. (Ejemplo: dia:10 mes:2 año: 2020)



Vamos a programar:

```
*ejercicio20.py - D:/Curso completo de Python/ejercicio20.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 dia=int(input("Ingrese día:"))
2 mes=int(input("Ingrese mes:"))
3 año=int(input("Ingrese año:"))
4 if mes==1 or mes==2 or mes==3:
5     print("La fecha ingresada corresponde al primer trimestre del año")
```

Guardamos el programa con el nombre Ejercicio20.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio20.py =============
    Ingrese día:10
    Ingrese mes:2
    Ingrese año:2020
    La fecha ingresada corresponde al primer trimestre del año
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio20.py =============
    Ingrese día:10
    Ingrese mes:12
    Ingrese año:2020
>>> |
```

Para entender el operador or hemos realizado este ejemplo pero también se podría realizar de la siguiente forma:

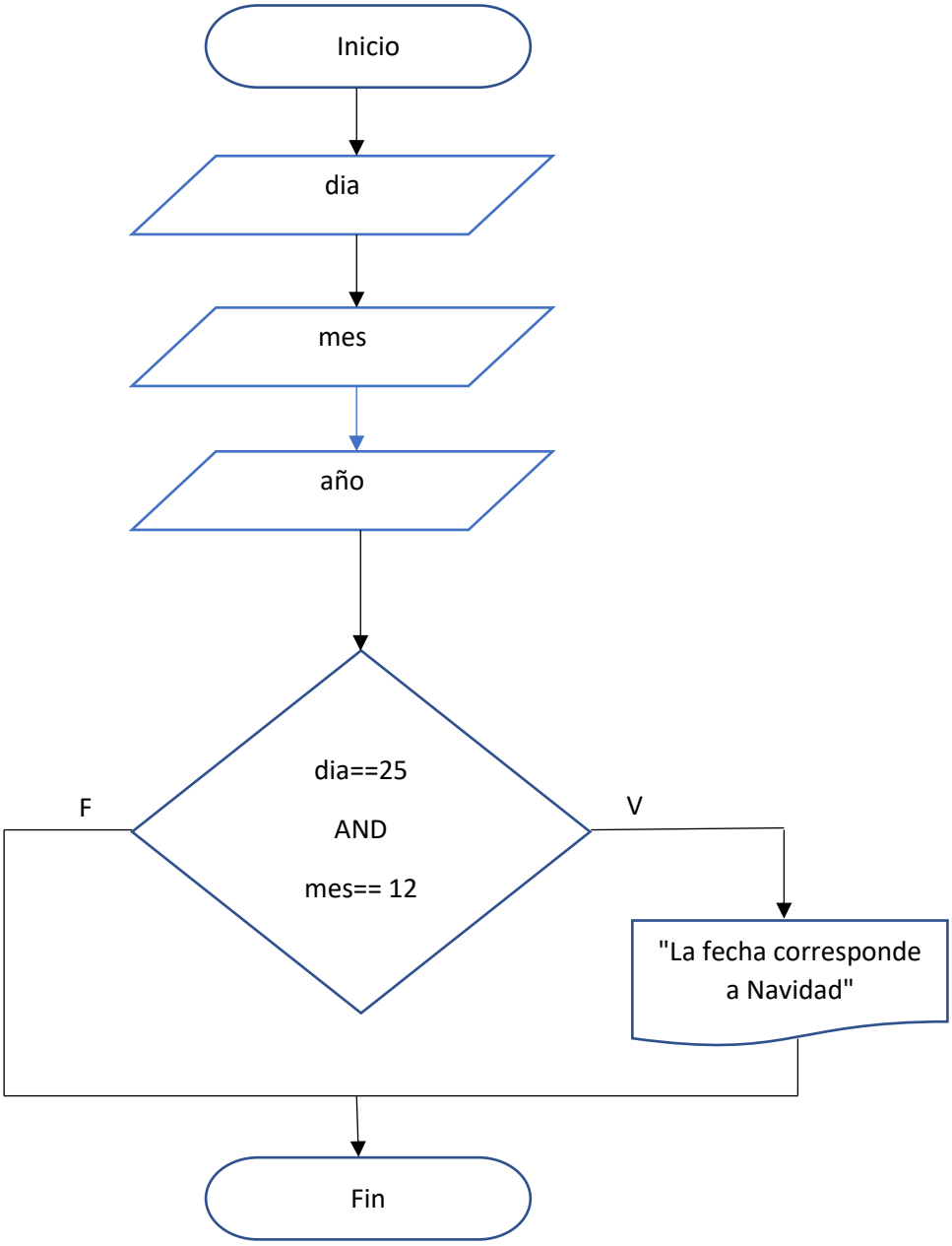
```
*ejercicio20.py - D:/Curso completo de Python/ejercicio20.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 dia=int(input("Ingrese día:"))
2 mes=int(input("Ingrese mes:"))
3 año=int(input("Ingrese año:"))
4 if mes<=3:
5     print("La fecha ingresada corresponde al primer trimestre del año")
6
```

El resultado será el mismo.

Capítulo 22.- Condiciones compuestas con operadores lógicos – 3

Problema propuesto:

Realizar un programa que pida cargar una fecha cualquiera, luego verificar si la fecha corresponde a Navidad.



Vamos a programar:

```
*ejercicio21.py - D:/Curso completo de Python/ejercicio21.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 dia=int(input("Ingrese el día:"))
2 mes=int(input("Ingeese el mes:"))
3 año=int(input("Ungrese el año:"))
4 if dia==25 and mes==12:
5     print("La fecha corresponde a navidad")
6
```

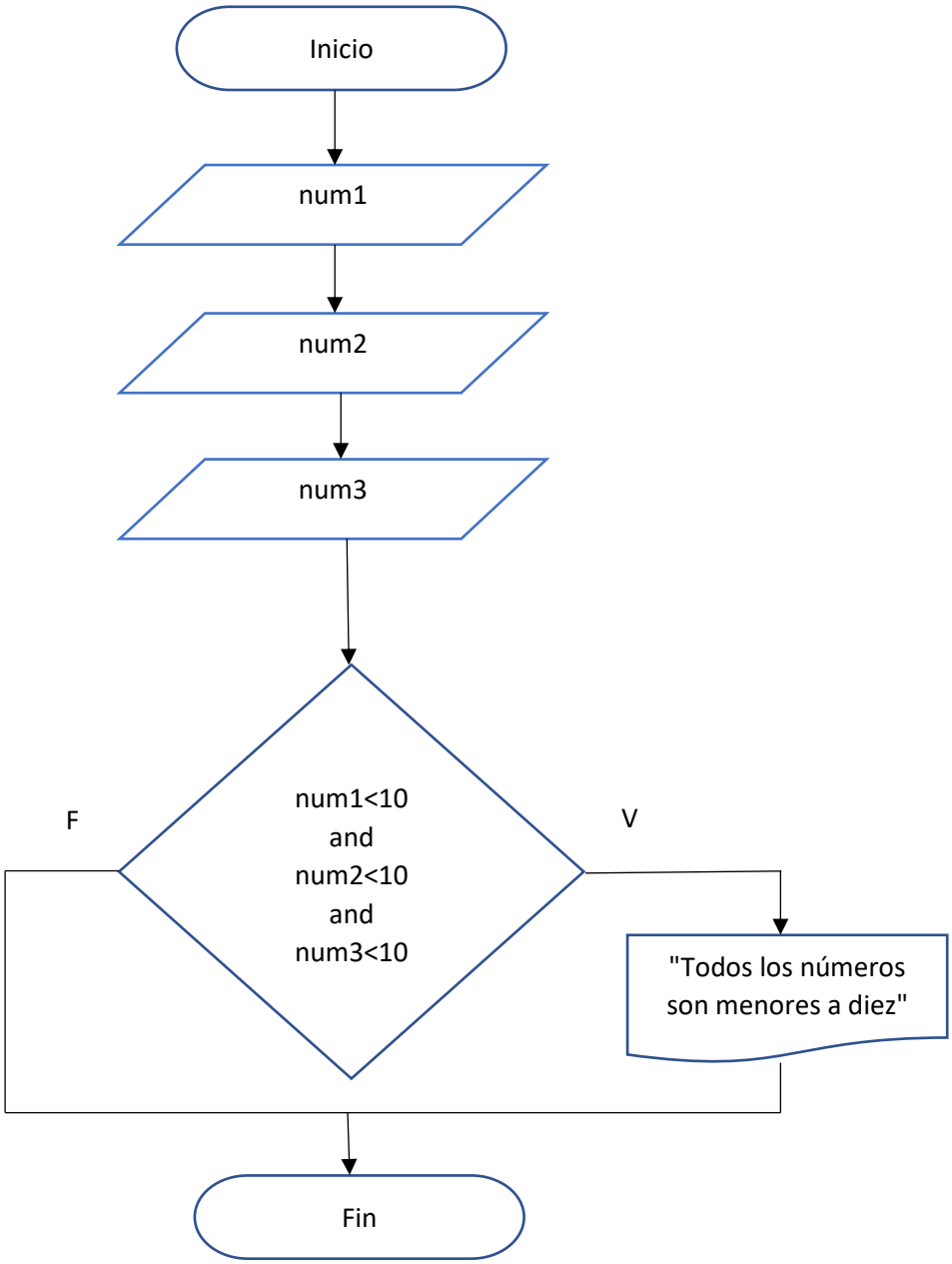
Guardarlo con el nombre de Ejercicio21.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio21.py =============
Ingrese el día:23
Ingeese el mes:1
Ungrese el año:2020
>>>
============== RESTART: D:/Curso completo de Python/ejercicio21.py =============
Ingrese el día:25
Ingeese el mes:12
Ungrese el año:2020
La fecha corresponde a navidad
>>>
```

Capítulo 23.- Condiciones compuestas con operadores lógicos – 4

Problema propuesto:

Se ingresan por teclado tres números, si todos los valores ingresados son menores a 10, imprimir en pantalla la leyenda "Todos los números son menores a diez".



Vamos a programar:

ejercicio22.py - D:/Curso completo de Python/ejercicio22.py (3.11.1)

File Edit Format Run Options Window Help

```
num1=int(input("Ingrese el primer valor:"))
num2=int(input("Ingrese el segundo valor:"))
num3=int(input("Ingrese el tercer valor:"))
if num1<10 and num2<10 and num3<10:
    print("Todos los valores son menores a 10")
```

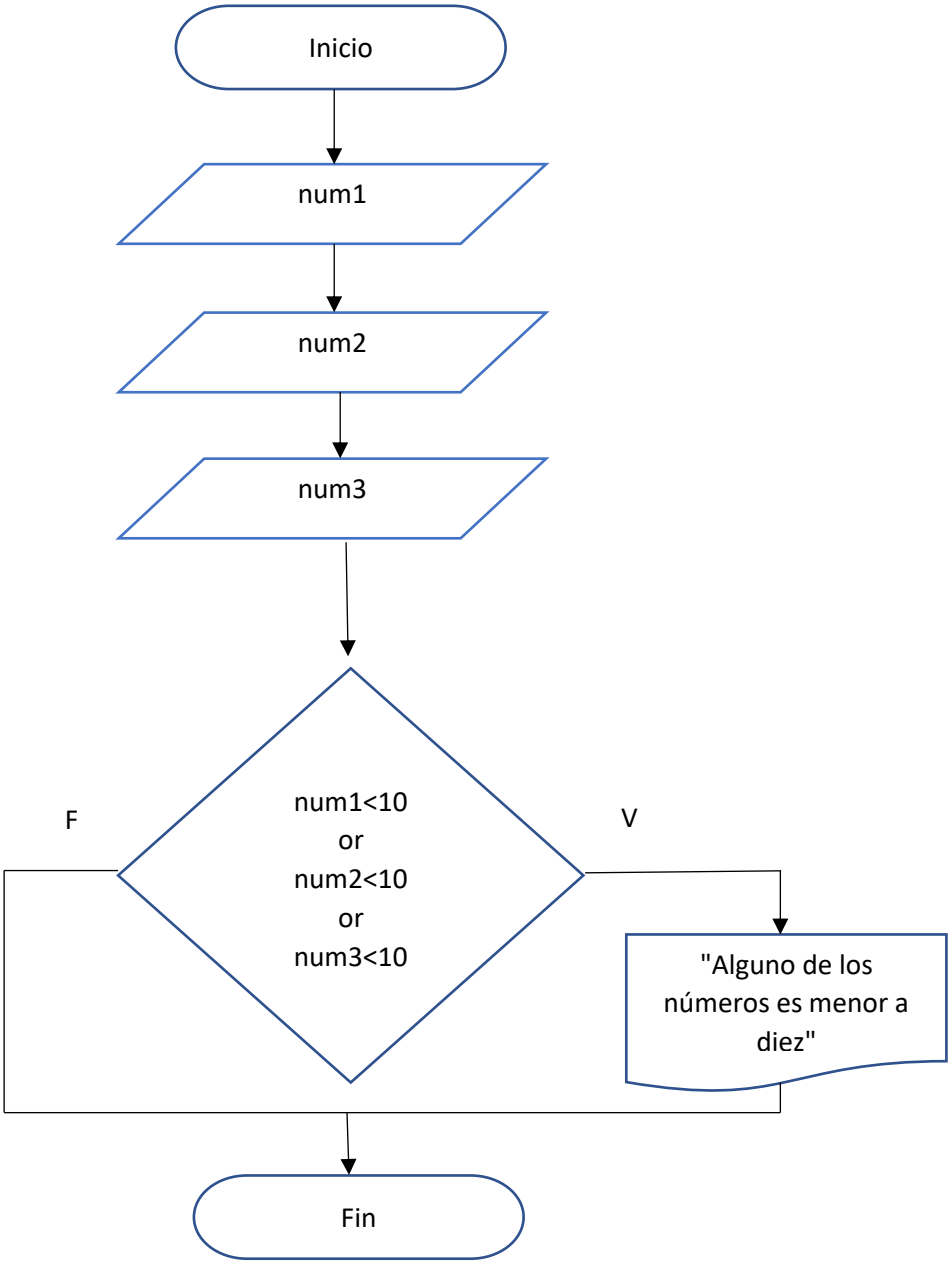
Vamos a guardar el programa con el nombre Ejerccicio22.py y a continuación vamos a ejecutarlo.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio22.py =============
Ingrese el primer valor:1
Ingrese el segundo valor:2
Ingrese el tercer valor:45
>>>
============== RESTART: D:/Curso completo de Python/ejercicio22.py =============
Ingrese el primer valor:1
Ingrese el segundo valor:2
Ingrese el tercer valor:3
Todos los valores son menores a 10
>>>
```

Capítulo 24.- Condiciones compuestas con operadores lógicos – 5

Problema propuesto:

Se ingresan por teclado tres números, Si al menos uno de los valores ingresados en menor a 10, imprimir la leyenda "Alguno de los números es menor a diez".



Vamos a realizar el programa:

```
num1=int(input("Ingrese el primer valor:"))
num2=int(input("Ingrese el segundo valor:"))
num3=int(input("Ingrese el tercer valor:"))
if num1<10 or num2<10 or num3<10:
    print("Al menos uno de los valores es menor a 10")
```

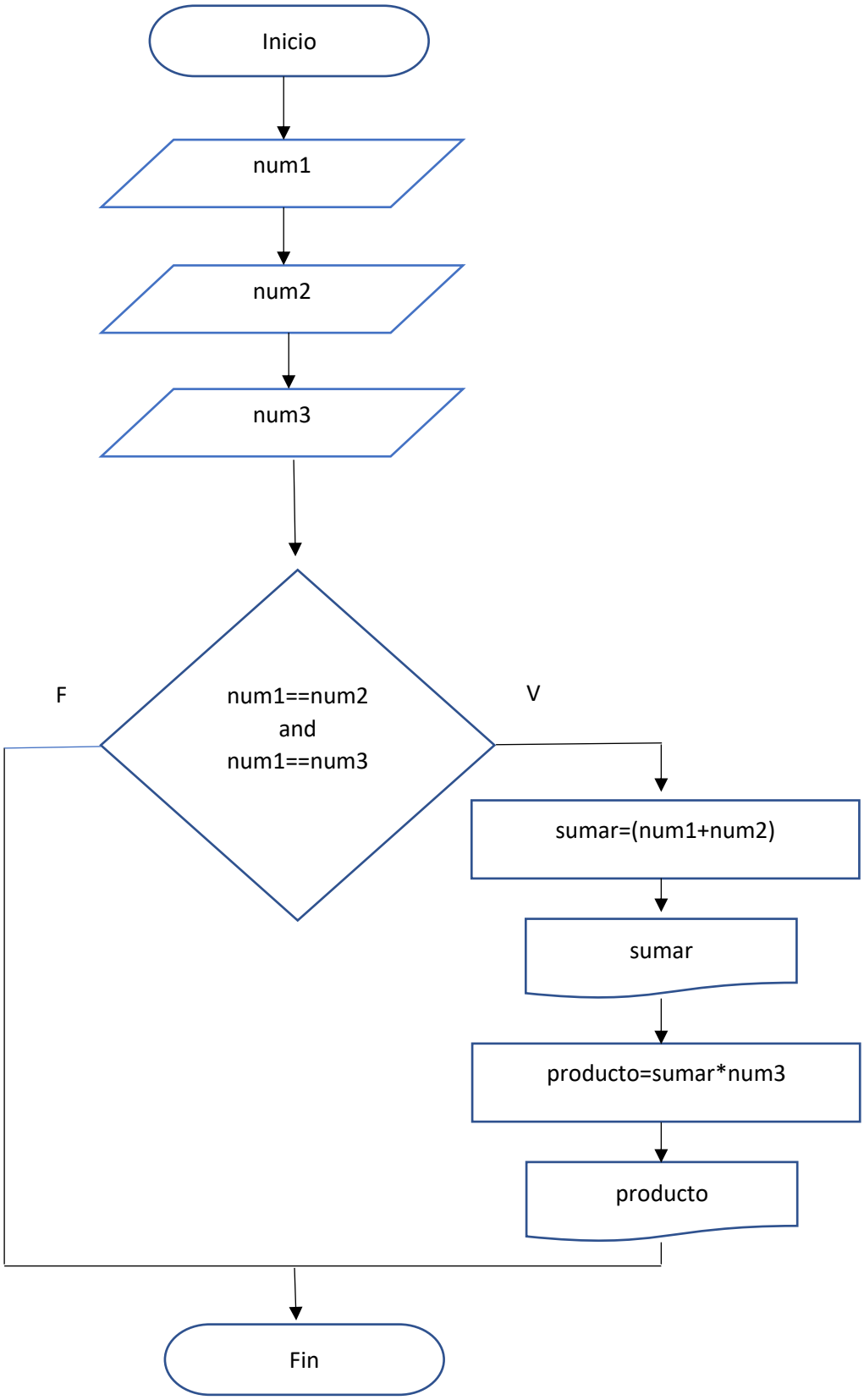
Vamos a guardarlo con el nombre Ejercicio23.py y a continuación lo vamos a ejecutar:

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio23.py =============
Ingrese el primer valor:12
Ingrese el segundo valor:14
Ingrese el tercer valor:23
>>>
============== RESTART: D:/Curso completo de Python/ejercicio23.py =============
Ingrese el primer valor:1
Ingrese el segundo valor:10
Ingrese el tercer valor:10
Al menos uno de los valores es menor a 10
>>>
```

Capítulo 25.- Condiciones compuestas con operadores lógicos – 6

Problema propuesto:

Se ingresan tres valores por teclado, si todos son iguales se imprime la suma del primero con el segundo y a este resultado se lo multiplica por el tercero.



Vamos a programar:

ejercicio24.py - D:/Curso completo de Python/ejercicio24.py (3.11.1)

File Edit Format Run Options Window Help

```
num1=int(input("Ingrese el primer valor:"))
num2=int(input("Ingrese el segundo valor:"))
num3=int(input("Ingrese el tercer valor:"))
if num1==num2 and num1==num3:
    suma=num1+num2
    print(suma)
    producto=suma*num3
    print("La suma del primero y segundo valor multiplicado por el tercero es")
    print(producto)
```

Vamos a guardarlo con el nombre de Ejercicio24.py y a continuación vamos a ejecutarlo:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

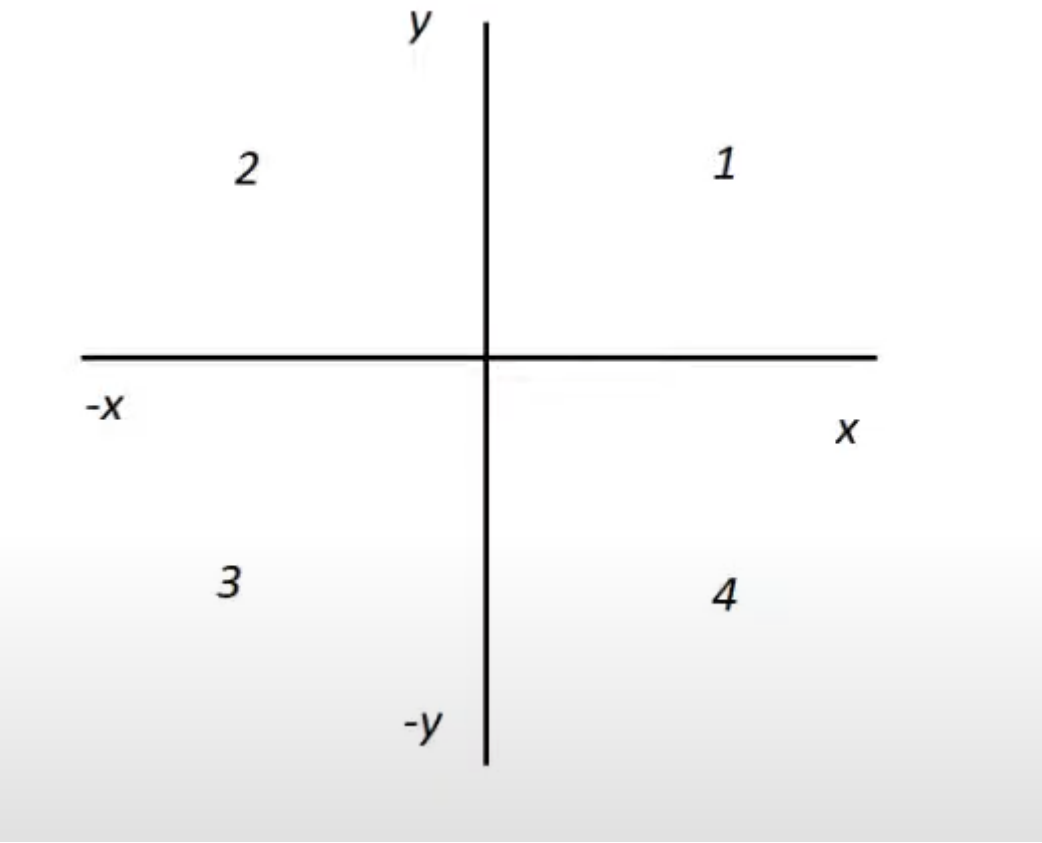
```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio24.py =============
Ingrese el primer valor:1
Ingrese el segundo valor:2
Ingrese el tercer valor:3
>>>
============== RESTART: D:/Curso completo de Python/ejercicio24.py =============
Ingrese el primer valor:10
Ingrese el segundo valor:10
Ingrese el tercer valor:10
20
La suma del primero y segundo valor multiplicado por el tercero es
200
>>>
```

Capítulo 26.- Condiciones compuestas con operadores lógicos – 7

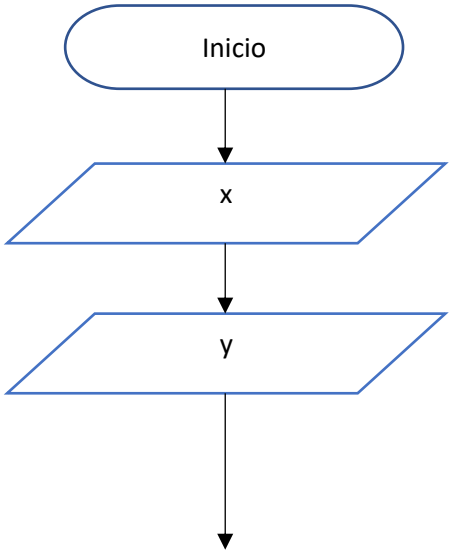
Problema propuesto:

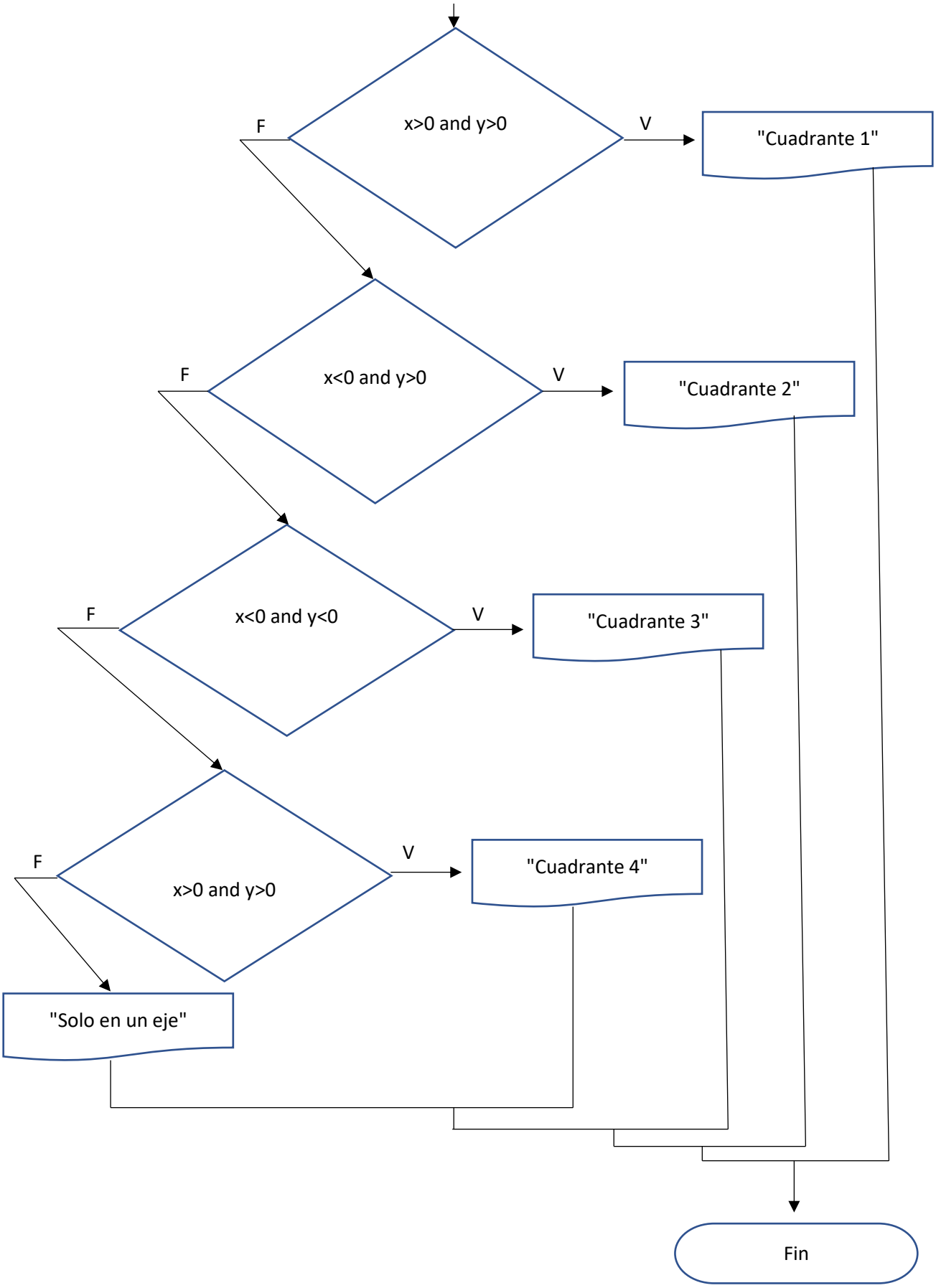
Escribir un programa que pida ingresar la coordenada de un punto plano, es decir dos valores enteros x e y (distintos de cero).

Posteriormente imprimir en pantalla en que cuadrante se ubica dicho punto. (1º Cuadrante si x>0 Y y>0, 2º Cuadrante: x<0 Y Y y>0, etc.)



Vamos al diagrama de flujo:




x>0 and y>0
F
V
"Cuadrante 1"
x<0 and y>0
F
V
"Cuadrante 2"
x<0 and y<0
F
V
"Cuadrante 3"
x>0 and y>0
F
V
"Cuadrante 4"
"Solo en un eje"
Fin

Vamos a programar:

ejercicio25.py - D:/Curso completo de Python/ejercicio25.py (3.11.1)

File Edit Format Run Options Window Help

```
x=int(input("Ingrese coordenada x:"))
y=int(input("Ingrese coordenada y:"))
if x>0 and y>0:
    print("Se encuentra en el primer cuadrante")
else:
    if x<0 and y>0:
        print("Se encuentra en el segundo cuadrante")
    else:
        if x<0 and y<0:
            print("Se encuentra en el tercer cuadrante")
        else:
            if x>0 and y<0:
                print("Se encuentra en el cuarto cuadrante")
            else:
                print("Se encuentra sobre un eje")
```

Lo guardamos con el nombre de Ejercicio25.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

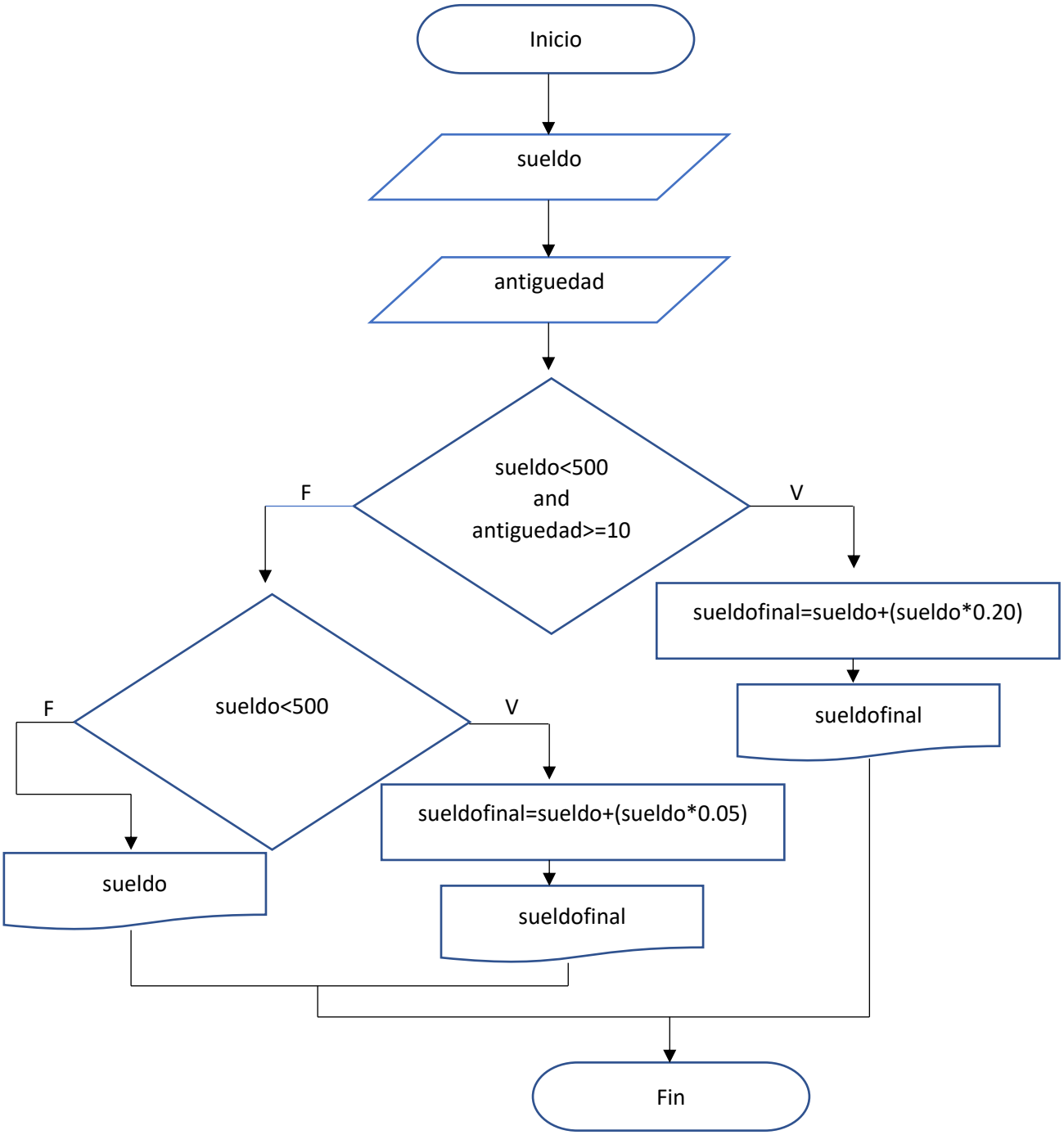
```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio25.py =============
Ingrese coordenada x:5
Ingrese coordenada y:3
Se encuentra en el primer cuadrante
>>>
============== RESTART: D:/Curso completo de Python/ejercicio25.py =============
Ingrese coordenada x:-3
Ingrese coordenada y:5
Se encuentra en el segundo cuadrante
>>>
============== RESTART: D:/Curso completo de Python/ejercicio25.py =============
Ingrese coordenada x:-3
Ingrese coordenada y:-3
Se encuentra en el tercer cuadrante
>>>
============== RESTART: D:/Curso completo de Python/ejercicio25.py =============
Ingrese coordenada x:4
Ingrese coordenada y:-4
Se encuentra en el cuarto cuadrante
>>>
============== RESTART: D:/Curso completo de Python/ejercicio25.py =============
Ingrese coordenada x:0
Ingrese coordenada y:3
Se encuentra sobre un eje
>>>
```

Capítulo 27.- Condiciones compuestas con operadores lógicos – 8

Problema propuesto:

De un operario se conoce su sueldo y los años de antigüedad. Se pide confeccionar un programa que lea los datos de entra e informe:

a) Si el sueldo es inferior a 500 y su antigüedad es igual o superior a 10 años, otorgarle un aumento del 20%, mostrar el sueldo a pagar.
b) Si el sueldo e inferior a 500 pero su antigüedad es menor a 10 años, otorgarle un aumento de 5%.
c) Si el sueldo es mayor o igual a 500 mostrar el sueldo en pantalla sin cambios.



vamos a programar:

ejercicio26.py - D:/Curso completo de Python/ejercicio26.py (3.11.1)

File Edit Format Run Options Window Help

```
sueldo=int(input("Ingrese el sueldo del empleado:"))
antiguedad=int(input("Ingrese la antiguedad en años:"))
if sueldo<500 and antiguedad>=10:
    sueldofinal=sueldo+(sueldo*0.20)
    print(sueldofinal)
else:
    if sueldo<500:
        sueldofinal=sueldo+(sueldo*0.05)
        print(sueldofinal)
    else:
        print(sueldo)
```

Lo guardamos con el nombre de Ejercicio26.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

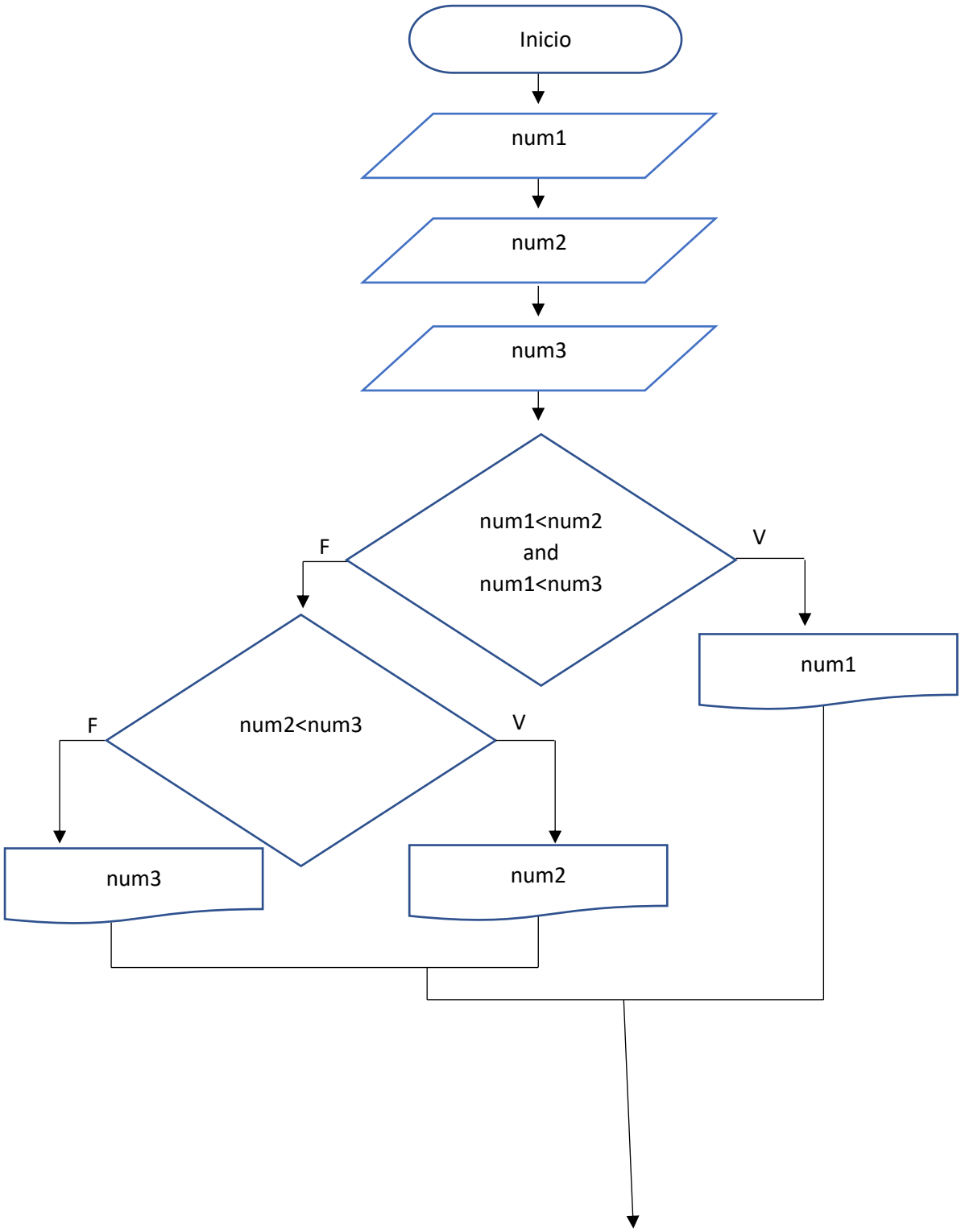
File Edit Shell Debug Options Window Help

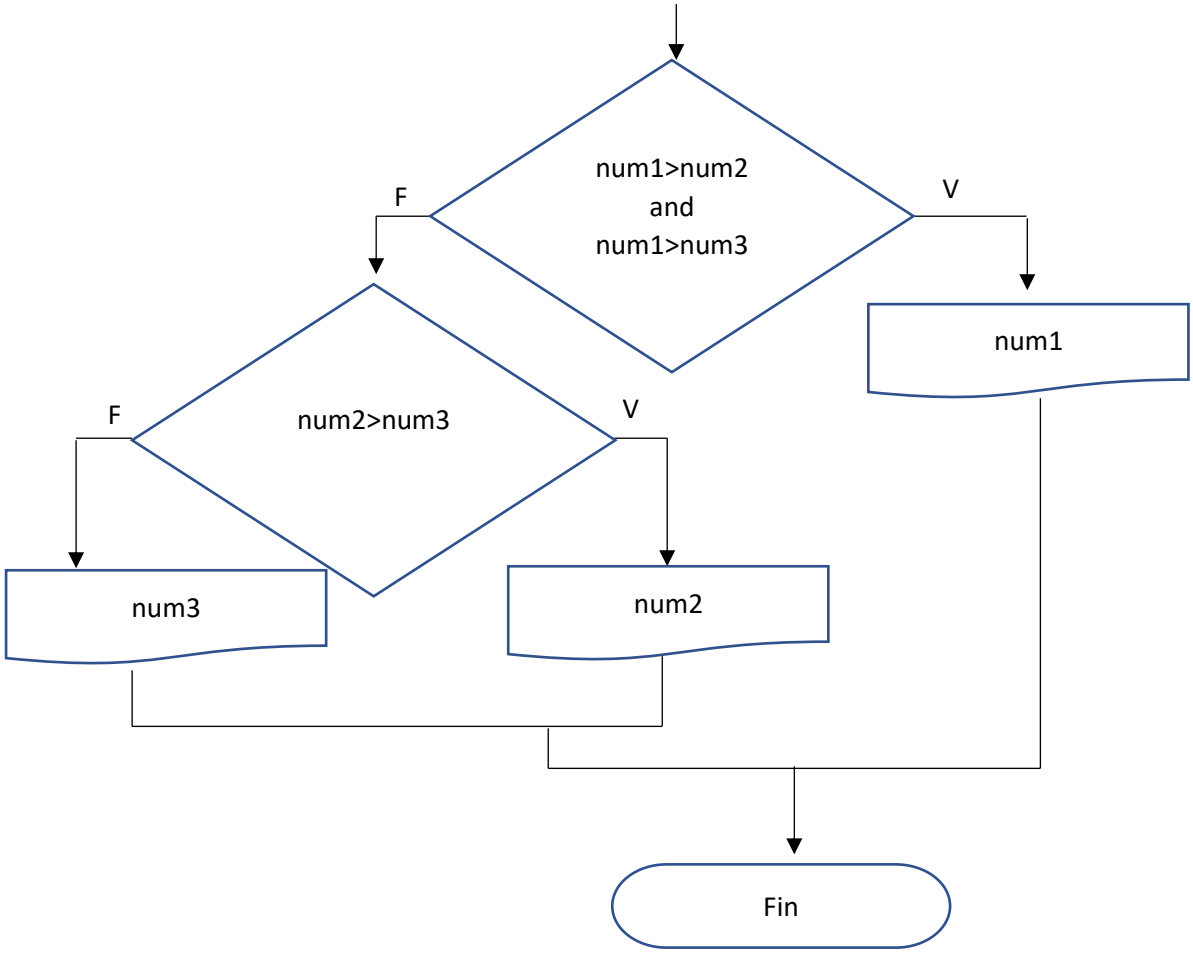
```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio26.py =============
Ingrese el sueldo del empleado:400
Ingrese la antiguedad en años:12
480.0
>>>
============== RESTART: D:/Curso completo de Python/ejercicio26.py =============
Ingrese el sueldo del empleado:400
Ingrese la antiguedad en años:6
420.0
>>>
============== RESTART: D:/Curso completo de Python/ejercicio26.py =============
Ingrese el sueldo del empleado:600
Ingrese la antiguedad en años:12
600
>>>
```

Capítulo 28.- Condiciones compuestas con operadores lógicos – 9

Problema propuesto:

Escribir un programa en e cual: dada una lista de tres valores numéricos distintos se calcule e informe su rango de variación (debe mostrar el mayor y el menor de ellos).





Vamos a programar:

ejercicio27.py - D:/Curso completo de Python/ejercicio27.py (3.11.1)

File Edit Format Run Options Window Help

```
num1=int(input("Ingrese primer valor:"))
num2=int(input("Ingrese segundo valor:"))
num3=int(input("Ingrese tercer valor:"))
if num1<num2 and num1<num3:
    print(num1)
else:
    if num2<num3:
        print(num2)
    else:
        print(num3)
if num1>num2 and num1>num3:
    print(num1)
else:
    if num2>num3:
        print(num2)
    else:
        print(num3)
```

Guardamos el programa con el nombre Ejercicio27.py y a continuación lo ejecutamos:

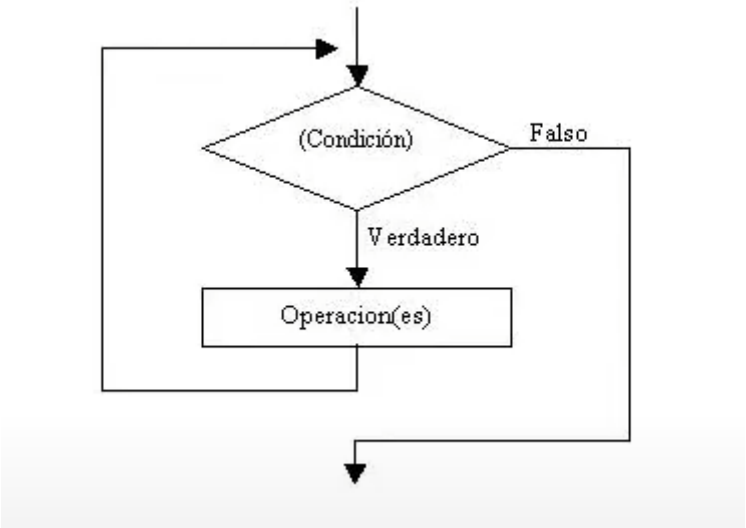
```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio27.py =============
Ingrese primer valor:1
Ingrese segundo valor:2
Ingrese tercer valor:3
1
3
>>>
============== RESTART: D:/Curso completo de Python/ejercicio27.py =============
Ingrese primer valor:67
Ingrese segundo valor:123
Ingrese tercer valor:4
4
123
>>>
============== RESTART: D:/Curso completo de Python/ejercicio27.py =============
Ingrese primer valor:100
Ingrese segundo valor:33
Ingrese tercer valor:1
1
100
>>>
```

Capítulo 29.- Estructura repetitiva while – 1

Hasta ahora hemos empleado estructuras SECUENCIALES y CONDICIONALES. Existen otro tipo de estructura tan importante como las anteriores que son las estructuras REPETITIVAS.

Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

Estructura repetitiva while



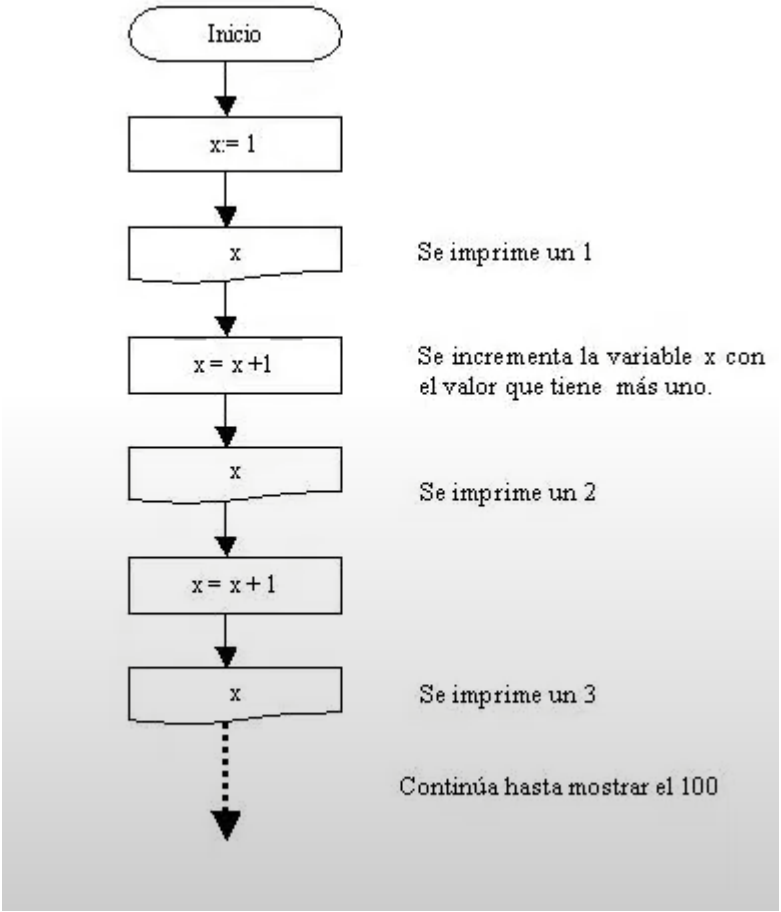
Una estructura repetitiva se caracteriza por:

- La sentencia o las sentencias que se repiten.
- El test o prueba de condición antes de cada repetición, que motivará que se repitan o no las instrucciones.

Problema

Realiza un programa que imprima en pantalla los números del 1 al 100.

Sin conocer las estructuras repetitivas podemos resolver el problema empleando una estructura secuencial. Iniciamos una variable con el valor 1, luego imprimimos la variable, incrementamos nuevamente la variable y así sucesivamente.



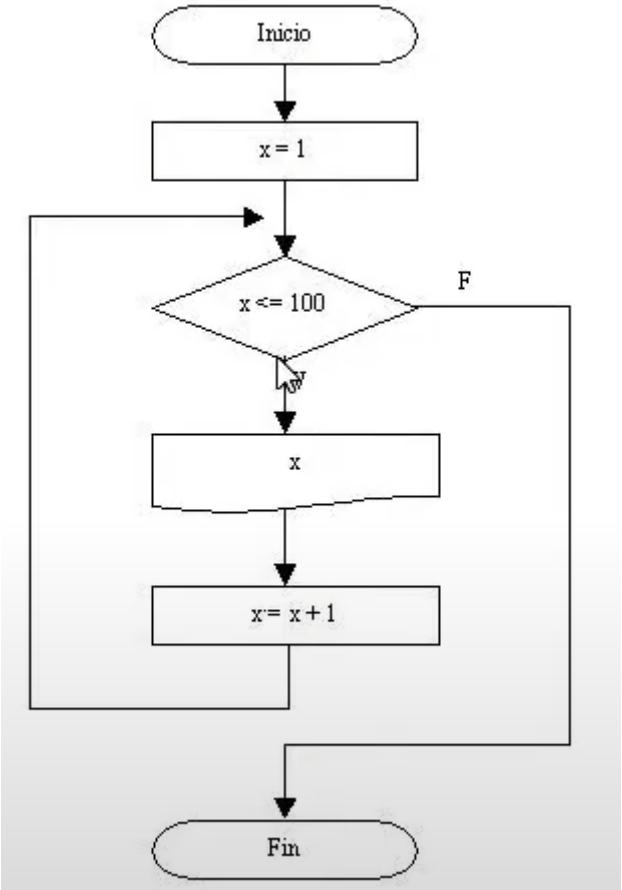
Veamos este ejemplo:

```
*ejercicio28.py - D:/Curso completo de Python/ejercicio28.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 x=1
2 print(x)
3 x=x+1
4 print(x)
5 x=x+1
6 print(x)
7 x=x+1
8 print(x)
9
```

Vamos a guardar el programa con el nombre Ejercicio28.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio28.py =============
    1
    2
    3
    4
>>>
```

Para llegar hasta el número 10 el programa va a ser muy largo y tedioso.
Con estructuras secuenciales este algoritmo no va a ser muy eficaz.



Vamos a programa el diagrama de flujo:

```
*ejercicio29.py - D:/Curso completo de Python/ejercicio29.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 x=1
2 while x<=100:
3     print(x)
4     x=x+1
```

Lo guardamos con el nombre Ejercicio29.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio29.py =============
1
2
3
4
5
6
7
8
9
10
11
12
```

. . .

```
87
88
89
90
91
92
93
94
95
96
97
98
99
100
>>>
Ln: 105  Col: 0
```

Queremos mostrar lo números impares:

```
*ejercicio29.py - D:/Curso completo de Python/ejercicio29.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 x=1
2 while x<=100:
3     print(x)
4     x=x+2
```

Guardamos los cambios y ejecutamos de nuevo.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio29.py =============
1
3
5
7
9
11
13
15
17
19
```

. . .

```
77
79
81
83
85
87
89
91
93
95
97
99
>>>
Ln: 55  Col: 0
```

Si queremos los números pares.

```
*ejercicio29.py - D:/Curso completo de Python/ejercicio29.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 x=2
2 while x<=100:
3     print(x)
4     x=x+2
5
```

Guardamos los cambios y ejecutamos de nuevo.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio29.py =============
2
4
6
8
```

. . .

```
96
98
100
>>>
Ln: 55  Col: 0
```

Queremos mostrar los números múltiplos de 5.

ejercicio29.py - D:/Curso completo de Python/ejercicio29.py (3.11.1)

File Edit Format Run Options Window Help

```
x=0
while x<=100:
    print(x)
    x=x+5
```

Guardamos los cambios y volvemos a ejecutar:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio29.py =============
0
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
>>>
```

Ln: 26 Col: 0

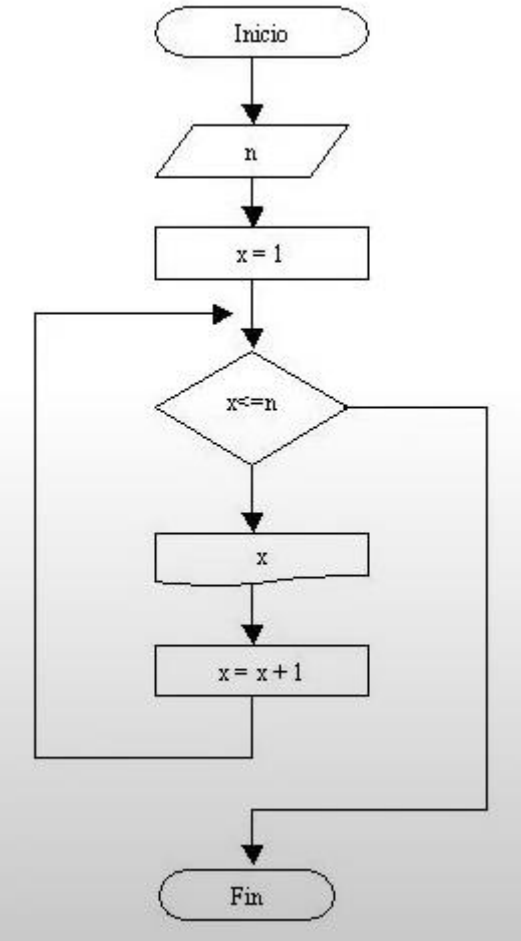
Capítulo 30.- Estructura repetitiva while – 2

Problema:

Codificar un programa que solicite la carga de un valor positivo y nos muestre desde 1 hasta el valor ingresado de uno en uno.

Ejemplo: Si ingresamos 30 se debe mostrar en pantalla los números del 1 al 30.

Es de FUNDAMENTAL importancia analizar los diagramas de flujo y la posterior codificación en Python de los siguientes problemas, en varios problemas se presentan otras situaciones no vistas en el ejercicio anterior.



Vamos a programar:

ejercicio30.py - D:/Curso completo de Python/ejercicio30.py (3.11.1)

File Edit Format Run Options Window Help

```
n=int(input("Ingrese el valor final:"))
x=1
while x<=n:
    print(x)
    x=x+1
```

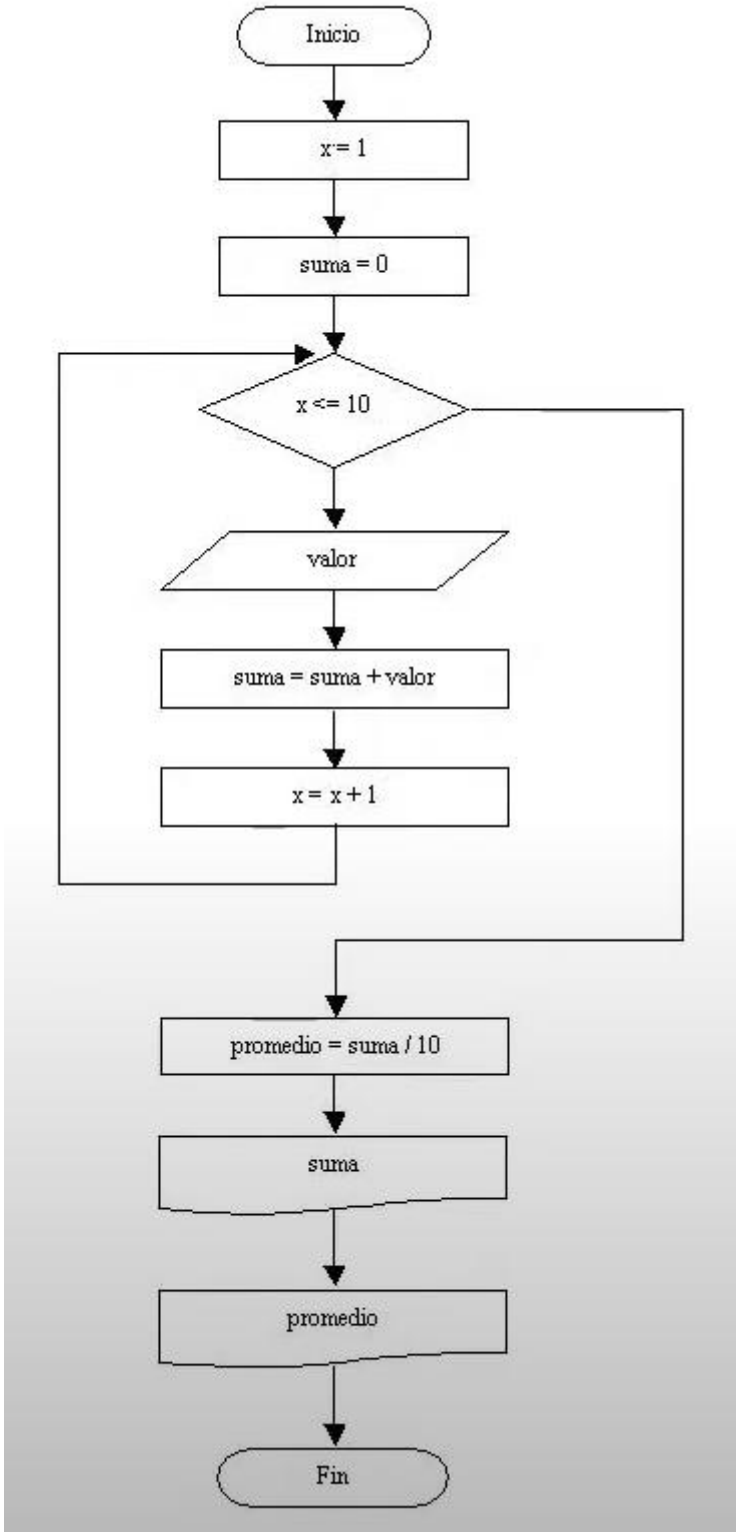
Lo guardamos con el nombre Ejercicio30.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio30.py =============
Ingrese el valor final:30
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
>>>
Ln: 36  Col: 0
```

Capítulo 31.- Estructura repetitiva while – 3

Problema:

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio.



Vamos a programar:

```
*ejercicio31.py - D:/Curso completo de Python/ejercicio31.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1  x=1
2  suma=0
3  while x<=10:
4      valor=int(input("Ingrese un valor:"))
5      suma=suma+valor
6      x=x+1
7  promedio=suma/10
8  print("La suma de los 10 valores es")
9  print(suma)
10 print("El promedio es")
11 print(promedio)
12
```

Guardamos el programa con el nombre de Ejercicio31.py y a continuación ejecutamos:

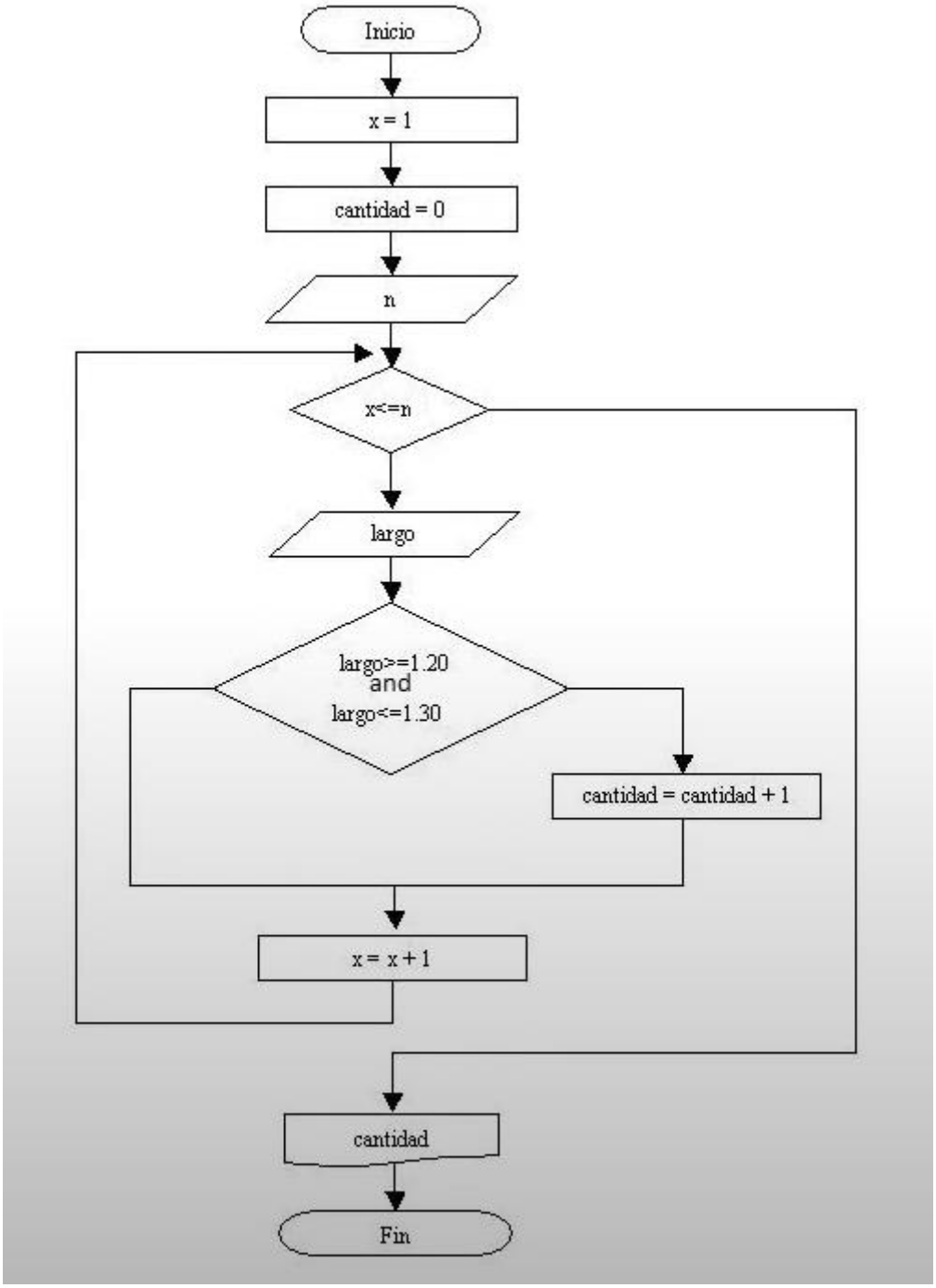
```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
   Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
   AMD64)] on win32
   Type "help", "copyright", "credits" or "license()" for more information.
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio31.py =============
   Ingrese un valor:1
   Ingrese un valor:2
   Ingrese un valor:3
   Ingrese un valor:4
   Ingrese un valor:5
   Ingrese un valor:6
   Ingrese un valor:7
   Ingrese un valor:8
   Ingrese un valor:9
   Ingrese un valor:10
   La suma de los 10 valores es
   55
   El promedio es
   5.5
>>>
                                                                    Ln: 19  Col: 0
```

Capítulo 32.- Estructura repetitiva while – 4

Problema:

Una planta que fabrica perfiles de hierro posee un lote de n piezas.

Confeccionar un programa que pida ingresar por teclado la cantidad de piezas a procesar y luego ingrese la longitud de cada perfil; sabiendo que la pieza cuya longitud esté comprendida en el rango de 1.20 y 1.30 son aptas. Imprimir por pantalla la cantidad de piezas aptas que hay en el lote.



Vamos a programar:

```
*ejercicio32.py - D:/Curso completo de Python/ejercicio32.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
x=1
cantidad=0
n=int(input("Ingrese cuantas piezas va a procesar:"))
while x<=n:
    largo=float(input("Ingrese la medida de la pieza:"))
    if largo>=1.20 and largo<=1.30:
        cantidad=cantidad+1
    x=x+1
print("Cantidad de piezas aptas")
print(cantidad)
```

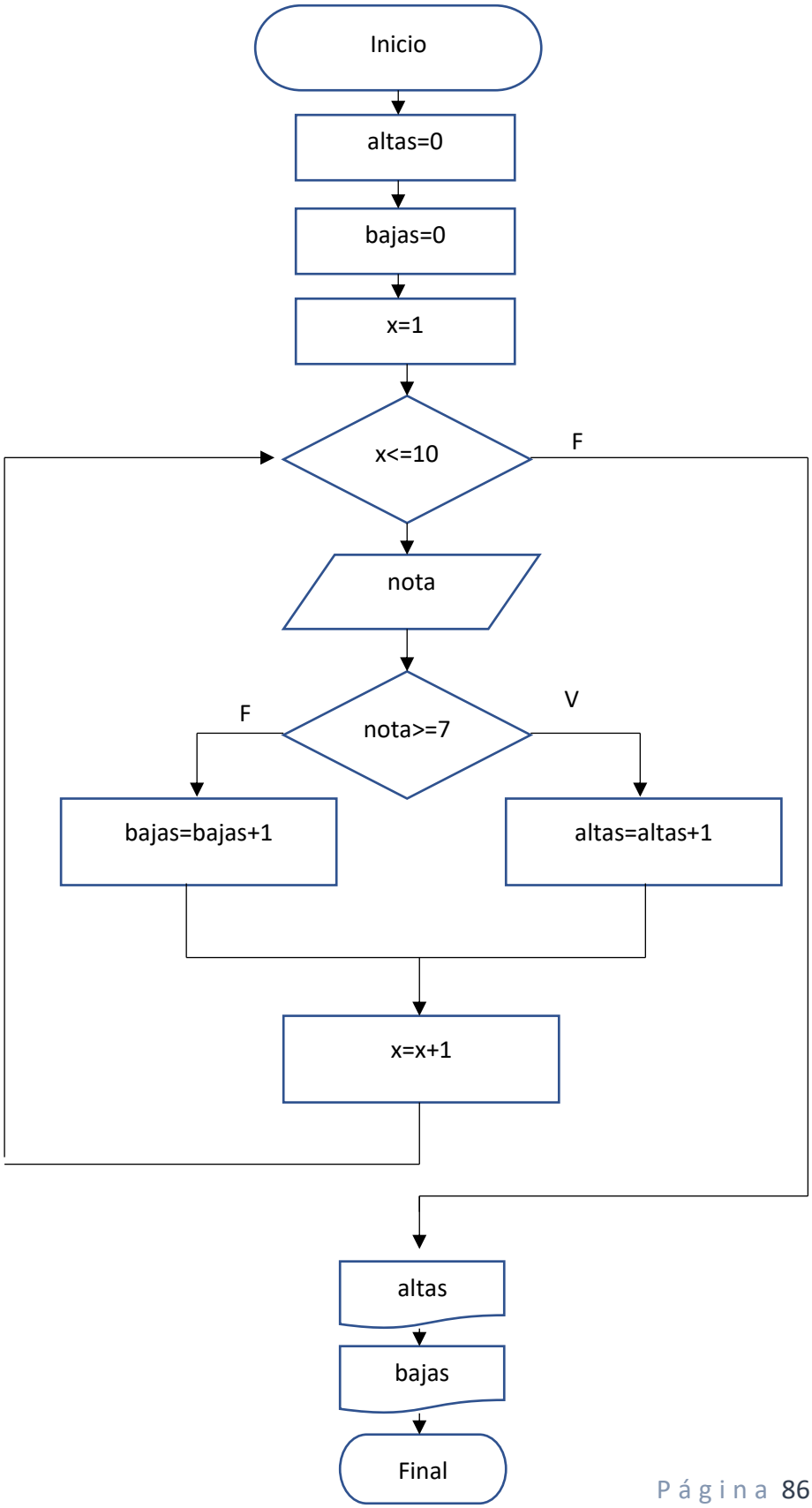
Lo guardamos con el nombre de Ejercicio32.py y a continuación lo ejecutamos.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio32.py =============
Ingrese cuantas piezas va a procesar:4
Ingrese la medida de la pieza:1.14
Ingrese la medida de la pieza:1.22
Ingrese la medida de la pieza:1.25
Ingrese la medida de la pieza:1.79
Cantidad de piezas aptas
2
>>>
```

Capítulo 33.- Estructura repetitiva while – 5

Problema propuesto:

Escribir un programa que solicite ingresar 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuantos menores.



Vamos a programar:

ejercicio33.py - D:/Curso completo de Python/ejercicio33.py (3.11.1)

File Edit Format Run Options Window Help

```
1  altas=0
2  bajas=0
3  x=1
4  while x<=10:
5      nota=int(input("Ingreso nota:"))
6      if nota>=7:
7          altas=altas+1
8      else:
9          bajas=bajas+1
10     x=x+1
11 print("Cantidad de alumnos con notas mayores o iguales a 7")
12 print(altas)
13 print("Cantidad d ealumnos con notas menores a 7")
14 print(bajas)
15
```

Guarda el programa con el nombre de Ejercicio33.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

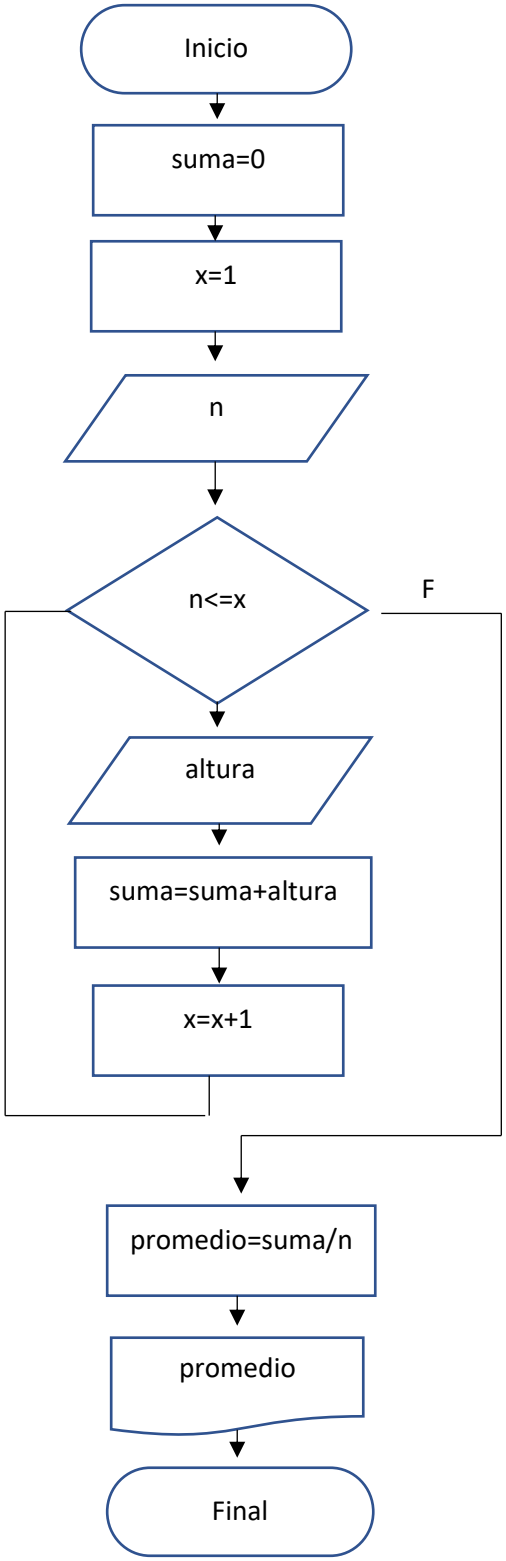
File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio33.py =============
Ingreso nota:1
Ingreso nota:2
Ingreso nota:3
Ingreso nota:4
Ingreso nota:5
Ingreso nota:6
Ingreso nota:7
Ingreso nota:8
Ingreso nota:9
Ingreso nota:10
Cantidad de alumnos con notas mayores o iguales a 7
4
Cantidad d ealumnos con notas menores a 7
6
>>>
```

Capítulo 34.- Estructura repetitiva while – 6

Problema propuesto:

Se ingresan un conjunto de n alturas de personas por teclado. Mostrar la altura promedio de las personas.



Vamos a programar:

```
*ejercicio34.py - D:/Curso completo de Python/ejercicio34.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1  suma=0
2  x=1
3  n=int(input("Cuantas personas se van a medir:"))
4  while x<=n:
5      altura=float(input("Ingrese la altura:"))
6      suma=suma+altura
7      x=x+1
8  promedio=suma/n
9  print("Altura promedio")
10 print(promedio)
11
```

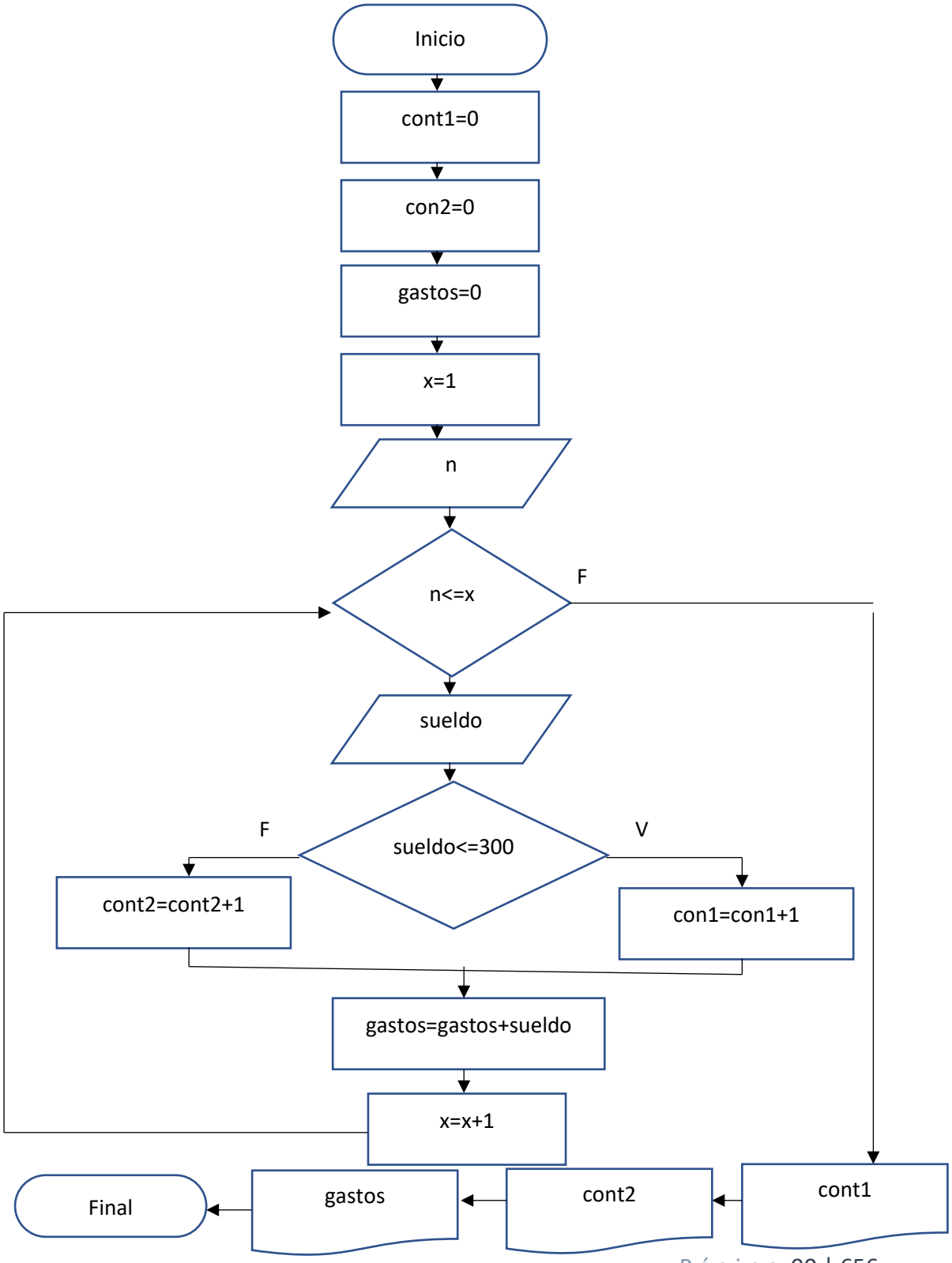
Lo guardamos con el nombre d eEjercicio34.py y a continuación lo ejecutamos.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
   Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
   AMD64)] on win32
   Type "help", "copyright", "credits" or "license()" for more information.
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio34.py =============
   Cuantas personas se van a medir:5
   Ingrese la altura:1.67
   Ingrese la altura:1.69
   Ingrese la altura:1.92
   Ingrese la altura:1.80
   Ingrese la altura:1.50
   Altura promedio
   1.7159999999999997
>>>
```

Capítulo 35.- Estructura repetitiva while – 7

Problema propuesto:

En una empresa trabajan n empleados cuyos sueldos oscilan entre $100 y $500, realiza un programa que lea los sueldos que cobra cada empleado e informe cuántos empleados cobran entre $100 y $300 y cuantos más de $300. Además el programa deberá informar el importe que gasta la empresa en sueldos al personal.



Vamos a programar:

ejercicio35.py - D:/Curso completo de Python/ejercicio35.py (3.11.1)

File Edit Format Run Options Window Help

```
cont1=0
cont2=0
gastos=0
x=1
n=int(input("Cuantos empleados tiene la empresa:"))
while x<=n:
    sueldo=float(input("Ingrese el sueldo del empleado:"))
    if sueldo<=300:
        cont1=cont1+1
    else:
        cont2=cont2+1
    gastos=gastos+sueldo
    x=x+1
print("Cantidad de empleados con un sueldos entre 100 y 300")
print(cont1)
print("Cantidad de empleados con sueldos mayores a 300")
print(cont2)
print("Los gastos de la empresa en sueldos")
print(gastos)
```

Guardamos el programa con el nombre de Ejercicio35.py y a continuación ejecutamos:

IDLE Shell 3.11.1

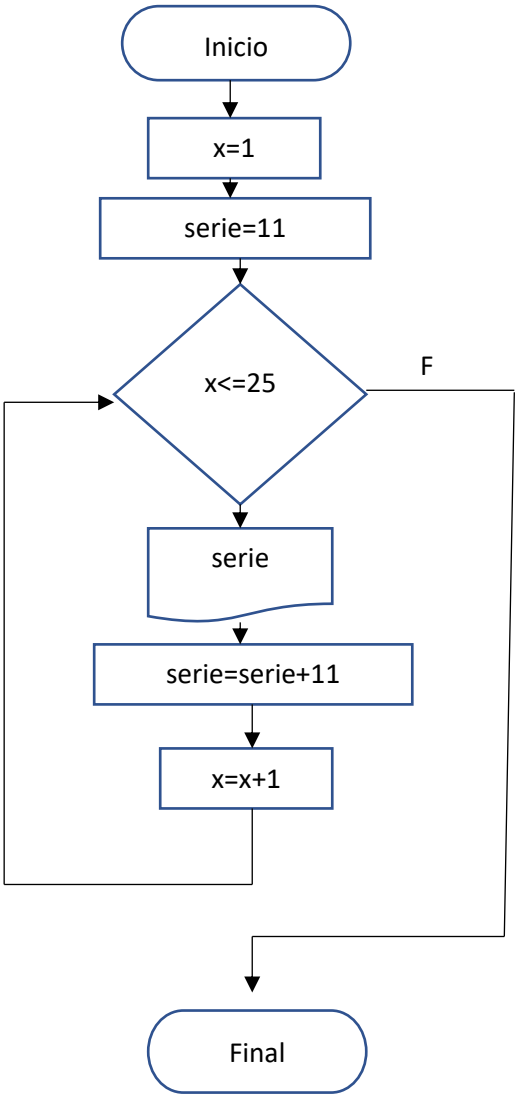
File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio35.py =============
Cuantos empleados tiene la empresa:5
Ingrese el sueldo del empleado:400
Ingrese el sueldo del empleado:450
Ingrese el sueldo del empleado:200
Ingrese el sueldo del empleado:200
Ingrese el sueldo del empleado:200
Cantidad de empleados con un sueldos entre 100 y 300
3
Cantidad de empleados con sueldos mayores a 300
2
Los gastos de la empresa en sueldos
1450.0
>>>
```

capítulo 36.- Estructura repetitiva while – 8

Problema propuesto:

Realiza un programa que imprima 25 términos de serie 11 – 22 – 33 – 44, etc. (No se ingresan valores por teclado).



Vamos a programar:

```
*ejercicio36.py - D:/Curso completo de Python/ejercicio36.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 x=1
2 serie=11
3 while x<=25:
4     print(serie)
5     serie=serie+11
6     x=x+1
```

Guardamos el programa con el nombre de Ejercicio36.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio36.py =============
11
22
33
44
55
66
77
88
99
110
121
132
143
154
165
176
187
198
209
220
231
242
253
264
275
>>>
                                                                    Ln: 30  Col: 0
```

Otra forma de hacerlo para obtener el mismo resultado:

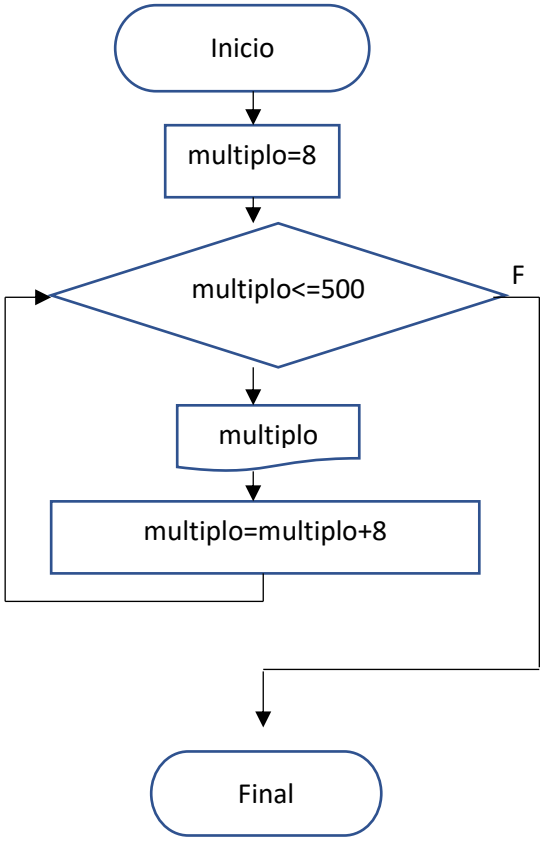
```
ejercicio36.py - D:/Curso completo de Python/ejercicio36.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 x=1
2 while x<=25:
3     serie=x*11
4     print(serie)
5     x=x+1
6
```

Esto significa que pueden haber varios algoritmos para llegar a la misma solución.

Capítulo 37.- Estructura repetitiva while – 9

Problema propuesto:

Mostrar los múltiplos de 8 hasta 500. Debe aparecer en pantalla 8 – 16 – 24, etc.



Vamos a programar:

```
*ejercicio37.py - D:/Curso completo de Python/ejercicio37.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 multiplo=8
2 while multiplo<=500:
3     print(multiplo)
4     multiplo=multiplo+8
5
```

Lo guardamos con el nombre de Ejercicio37.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio37.py =============
8
16
24
32
40
48
56
```

. . .

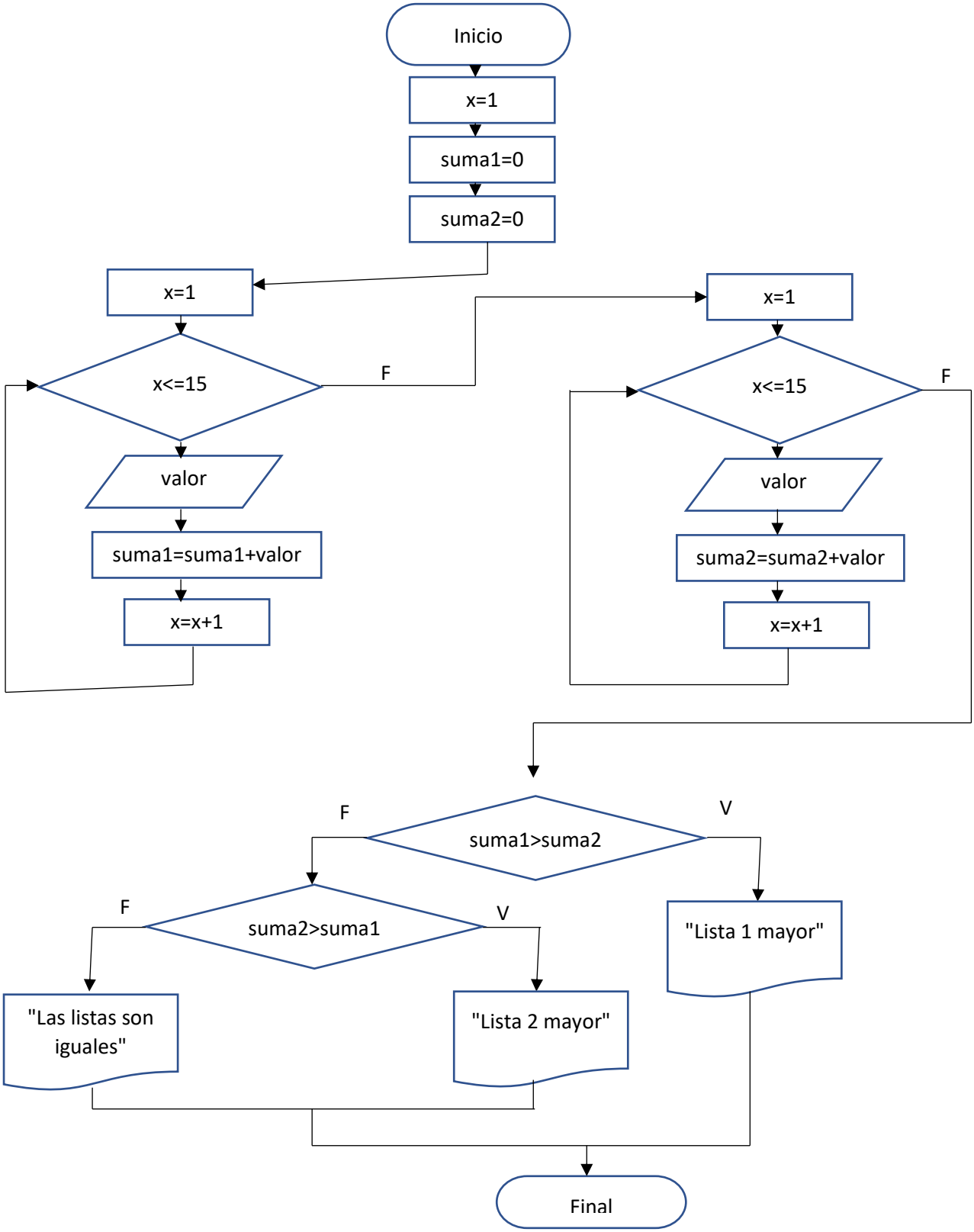
```
432
440
448
456
464
472
480
488
496
>>>
```

Ln: 67 Col: 0

Capítulo 38.- Estructura repetitiva while – 10

Realiza un programa que permita cargar dos listas de 15 valores cada una. Informar con un mensaje cuál de las dos listas tiene un valor acumulado mayor (mensaje "Lista 1 mayor", "Lista 2 mayor", "Listas iguales")

Tener en cuenta que puede haber dos o más estructuras repetitivas en un algoritmo.



Vamos a programar:

ejercicio38.py - D:/Curso completo de Python/ejercicio38.py (3.11.1)

File Edit Format Run Options Window Help

```
suma1=0
suma2=0
x=1
print("Carga de la primera lista")
while x<=15:
    valor=int(input("Ingrese valor: "))
    suma1=suma1+valor
    x=x+1
x=1
print("Carga de la segunda lista")
while x<=15:
    valor=int(input("Ingrese valor: "))
    suma2=suma2+valor
    x=x+1
if suma1>suma2:
    print("La lista 1 tiene un valor acumulado mayor")
else:
    if suma2>suma1:
        print("La lista 2 tiene un vamor acumulado mayor")
    else:
        print("Las listas tienen un valor acumulado igual")
```

Guardamos el programa con el nombre Ejercicio38.py y a continuación lo ejecutamos:

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio38.py =============
Carga de la primera lista
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Carga de la segunda lista
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
Ingrese valor: 2
La lista 2 tiene un vamor acumulado mayor
```

Ejecutamos de nuevo:

```
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio38.py =============
    Carga de la primera lista
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Carga de la segunda lista
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Ingrese valor: 1
    Las listas tienen un valor acumulado igual
```

Ejecutamos de nuevo:

```
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio38.py =============
    Carga de la primera lista
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
    Ingrese valor: 5
```

```
Carga de la segunda lista
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
Ingrese valor: 1
La lista 1 tiene un valor acumulado mayor
```

Capítulo 39.- Estructura repetitiva while – 11

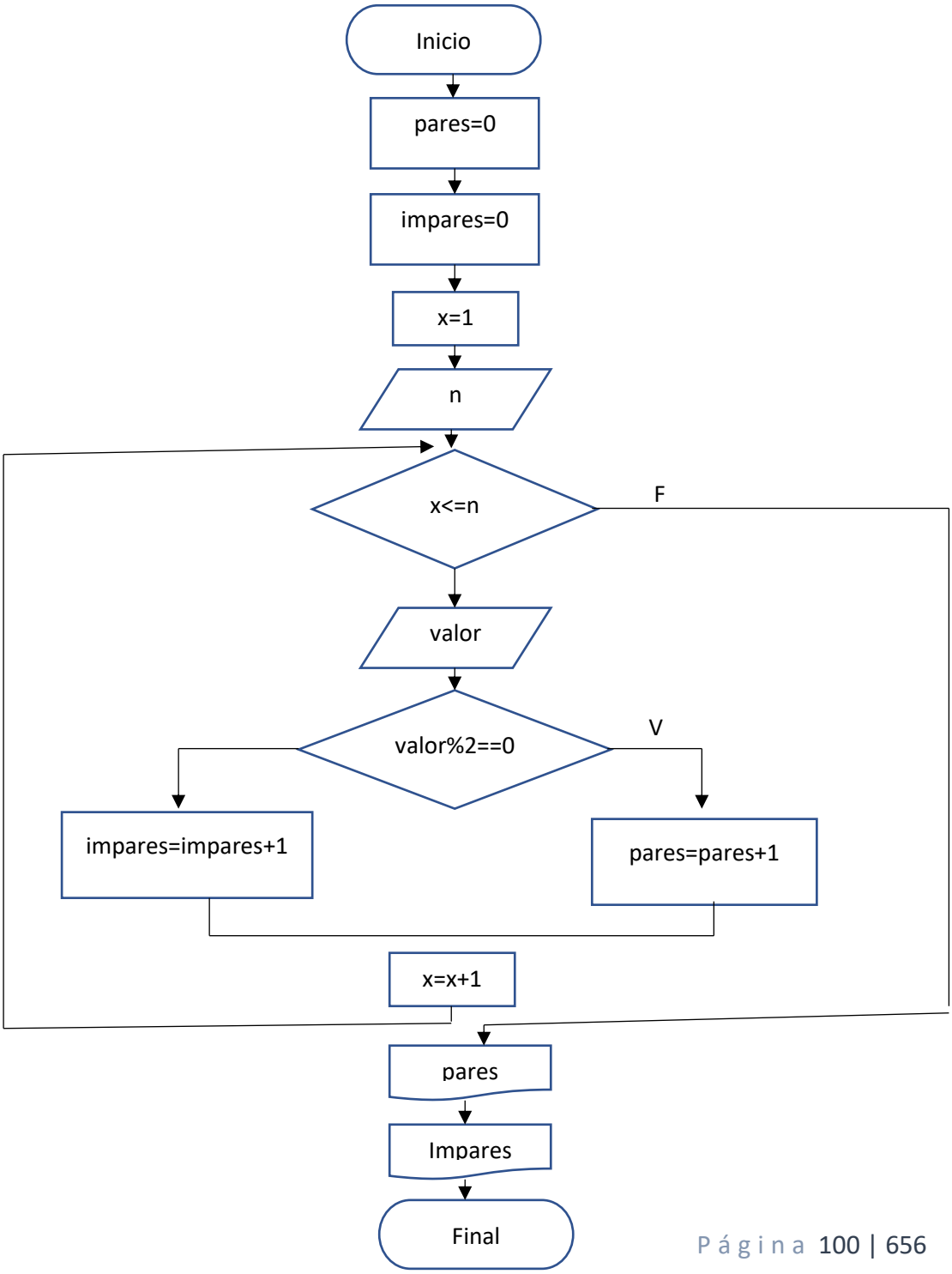
Problema propuesto:

Desarrollar un programa que permita cargar n números enteros y luego nos informe cuántos valores fueron pares y cuantos impares.

Emplear el operador "%" en la condición de la estructura condicional (este operador retorna el resto de la división de los valores, por ejemplo 11%2 retorna un 1):

if valor%2==0: (valor es par)

if valor%2==1: (valor es impar)



Vamos a programar:

```
*ejercicio39.py - D:/Curso completo de Python/ejercicio39.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
pares=0
impares=0
x=1
n=int(input("Cuantos números va a ingresar:"))
while x<=n:
    valor=int(input("Ingrese el valor:"))
    if valor%2==0:
        pares=pares+1
    else:
        impares=impares+1
    x=x+1
print("Cantidad de pares")
print(pares)
print("Cantidad de impares")
print(impares)
```

Guarde el programa con el nombre de Ejercicio39.py y a continuación ejecútelo:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio39.py =============
Cuantos números va a ingresar:5
Ingrese el valor:1
Ingrese el valor:2
Ingrese el valor:3
Ingrese el valor:4
Ingrese el valor:5
Cantidad de pares
2
Cantidad de impares
3
>>>
```

Resumiendo utilizando el operador del % conseguimos el siguiente resultado.

21%2	20%2
21 | 2 1 | 10	20 | 2 0 | 10

El valor que obtenemos es el resto, cuando el resto da 0 el número es par.

Capítulo 40.- Estructura repetitiva for – 1

Cualquier problema que requiera una estructura repetitiva se puede resolver empleando la estructura while, pero hay otra estructura repetitiva cuyo planteo es más sencillo en ciertas situaciones que tenemos que recorrer una lista de datos o repetir una cantidad definida de veces un bloque.

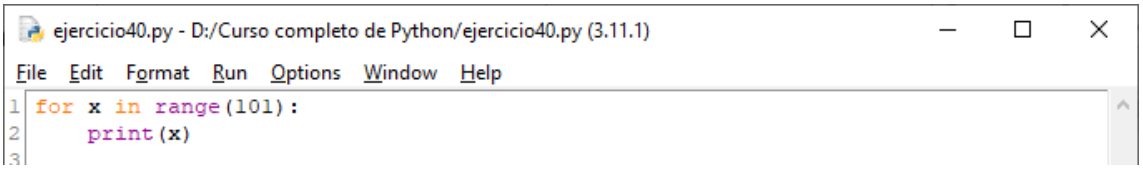
En general, la estructura repetitiva for se usa en aquellas situaciones en las cuales queremos que una variable vaya tomando un valor de una lista definida por valores.

```
for x in range(101)
    print(x)
```

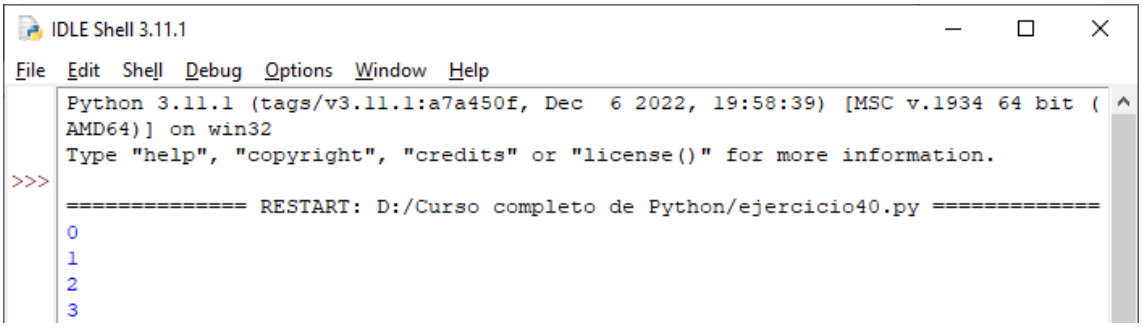
Problema 1:

Realiza un programa que imprima en pantalla los números del 0 al 100. Este problema lo podemos resolver perfectamente con el ciclo while pero en esta situación lo resolveremos empleando el for.

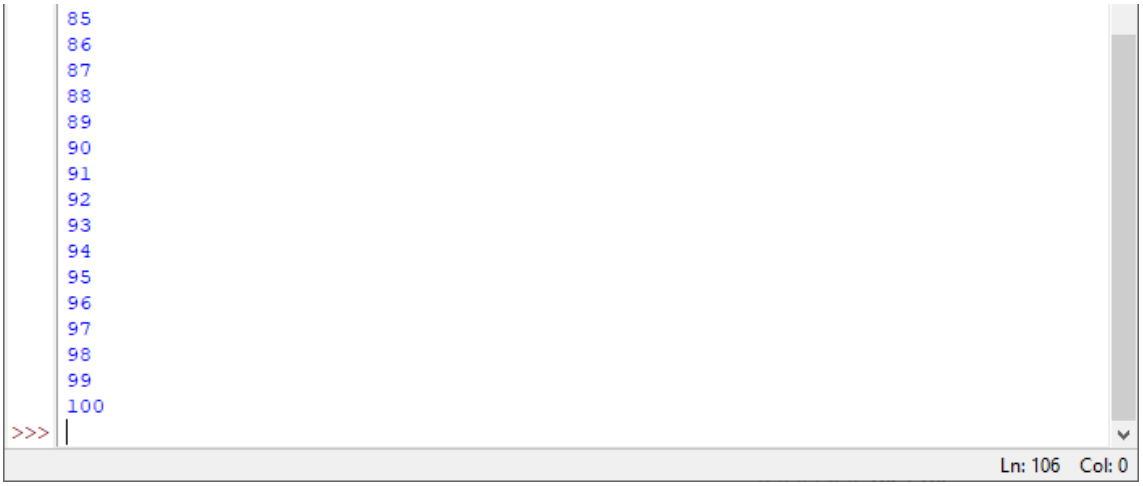
Vamos a programar:



Guardamos el programa con el nombre Ejercicio40.py y a continuación lo ejecutamos:



. . .



Recorre todos los valores hasta el valor asignado menos 1.

Con el ciclo while este sería el código:

```
*ejercicio40.py - D:/Curso completo de Python/ejercicio40.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 x=0
2 while x<=100:
3     print(x)
4     x=x+1
5
```

Si queremos imprimir los valores del 20 hasta el 30.

```
*ejercicio40.py - D:/Curso completo de Python/ejercicio40.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 for x in range(20,31):
2     print(x)
3
```

Vamos a ejecutar:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>> 
    ============== RESTART: D:/Curso completo de Python/ejercicio40.py =============
    20
    21
    22
    23
    24
    25
    26
    27
    28
    29
    30
>>> 
                                                                    Ln: 16   Col: 0
```

Para imprimir todos los valores impares desde 1 hasta 100.

```
*ejercicio40.py - D:/Curso completo de Python/ejercicio40.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 for x in range(1,101,2):
2     print(x)
3
```

Vamos a ejecutar:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>> 
    ============== RESTART: D:/Curso completo de Python/ejercicio40.py =============
    1
    3
```

. . .

```
87
89
91
93
95
97
99
>>>
```

Ln: 55 Col: 0

Capítulo 41.- Estructura repetitiva for – 2

Problema:

Desarrollar un programa que permita la carga de 10 valores por teclado y nos muestre posteriormente la suma de los valores ingresados y su promedio. Este problema ya lo desarrollamos, lo resolveremos empleando la estructura for para repetir la caga de los diez valores por teclado.

Vamos a programar:

```
*ejercicio41.py - D:/Curso completo de Python/ejercicio41.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1  suma=0
2  for x in range(10):
3      valor=int(input("Ingrese un valor: "))
4      suma=suma+valor
5  print("La suma de los 10 valores es")
6  print(suma)
7  promedio=suma/10
8  print("El promedio es")
9  print(promedio)
10
```

Guardamos el programa con el nombre de Ejercicio41.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
   Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
   AMD64)] on win32
   Type "help", "copyright", "credits" or "license()" for more information.
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio41.py =============
   Ingrese un valor: 1
   Ingrese un valor: 2
   Ingrese un valor: 3
   Ingrese un valor: 4
   Ingrese un valor: 5
   Ingrese un valor: 6
   Ingrese un valor: 7
   Ingrese un valor: 8
   Ingrese un valor: 9
   Ingrese un valor: 10
   La suma de los 10 valores es
   55
   El promedio es
   5.5
>>>
```

Vamos a cambiar el código:

```
*ejercicio41.py - D:/Curso completo de Python/ejercicio41.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1  suma=0
2  for x in range(10):
3      valor=int(input("Ingrese un valor: "))
4      suma=suma+valor
5  print("La suma de los 10 valores es")
6  print(suma)
7  promedio=suma//10
8  print("El promedio es")
9  print(promedio)
10
```

en la línea 10 hemos cambiado el operador de dividir "/" por el de por el operador matemático división enteros "//", para obtener un resultado sin la parte decimal.

Vamos a ejecutar:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio41.py =============
    Ingrese un valor: 1
    Ingrese un valor: 2
    Ingrese un valor: 3
    Ingrese un valor: 4
    Ingrese un valor: 5
    Ingrese un valor: 6
    Ingrese un valor: 7
    Ingrese un valor: 8
    Ingrese un valor: 9
    Ingrese un valor: 10
    La suma de los 10 valores es
    55
    El promedio es
    5 ←
>>>
```

Capítulo 42.- Estructura repetitiva for – 3

Problema:

Escribir un programa que solicite por teclado 10 notas de alumnos y nos informe cuántos tienen notas mayores o iguales a 7 y cuantos menores.

ejercicio42.py - D:/Curso completo de Python/ejercicio42.py (3.11.1)

File Edit Format Run Options Window Help

```
aprobados=0
reprobados=0
for x in range(10):
    nota=int(input("Introduzca la nota: "))
    if nota>=7:
        aprobados=aprobados+1
    else:
        reprobados=reprobados+1
print("Cantidad de aprobados")
print(aprobados)
print("Cantidad de reprobados")
print(reprobados)
```

El programa los guardamos con el nombre de Ejercicio42.py y a continuación lo ejecutamos.

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio42.py =============
Introduzca la nota: 1
Introduzca la nota: 2
Introduzca la nota: 3
Introduzca la nota: 4
Introduzca la nota: 5
Introduzca la nota: 6
Introduzca la nota: 7
Introduzca la nota: 8
Introduzca la nota: 9
Introduzca la nota: 10
Cantidad de aprobados
4
Cantidad de reprobados
6
>>>
```

Capítulo 43.- Estructura repetitiva for – 4

Problema:

Escribir un programa que lea 10 números y luego muestre cuántos valores ingresados fueron múltiplos de 3 y cuanto de 5. Debemos tener en cuenta que hay números que son múltiplos de 3 y 5 a la vez.

ejercicio43.py - D:/Curso completo de Python/ejercicio43.py (3.11.1)

File Edit Format Run Options Window Help

```
mul3=0
mul5=0
for x in range(10):
    valor=int(input("Ingrese un número: "))
    if valor%3==0:
        mul3=mul3+1
    if valor%5==0:
        mul5=mul5+1
print("Cantidad de valores multiplos de 3")
print(mul3)
print("Cantidad de valores multiplos de 5")
print(mul5)
```

Vamos a guardar el programa con el nombre Ejercicio43.py y a continuación vamos a ejecutarlo.

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio43.py =============
Ingrese un número: 1
Ingrese un número: 2
Ingrese un número: 3
Ingrese un número: 4
Ingrese un número: 5
Ingrese un número: 6
Ingrese un número: 7
Ingrese un número: 8
Ingrese un número: 9
Ingrese un número: 15
Cantidad de valores multiplos de 3
4
Cantidad de valores multiplos de 5
2
>>>
```

Capítulo 44.- Estructura repetitiva for – 5

Problema:

Codificar un programa que lea n números enteros y calcule la cantidad de valores mayores o iguales a 1000 (n se carga por teclado).

Este tipo de problemas también se pueden resolver empleando la estructura repetitiva for. Lo primero que se hace es cargar una variable que indique la cantidad de valores a ingresar. Dicha variable se carga antes de entrar en la estructura for.

```
*ejercicio44.py - D:/Curso completo de Python/ejercicio44.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 cantidad=0
2 n=int(input("Cuantos valores vas a ingresar:"))
3 for f in range(n):
4     valor=int(input("Ingrese un valor:"))
5     if valor>=1000:
6         cantidad=cantidad+1
7 print("Los valores ingresados mayores o igual a 1000 son")
8 print(cantidad)
9
```

Lo guardamos con el nombre de Ejercicio44.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio44.py =============
Cuantos valores vas a ingresar:5
Ingrese un valor:1
Ingrese un valor:2
Ingrese un valor:3
Ingrese un valor:1200
Ingrese un valor:1500
Los valores ingresados mayores o igual a 1000 son
2
>>>
```

Ejecutamos de nuevo:

```
>>>
============== RESTART: D:/Curso completo de Python/ejercicio44.py =============
Cuantos valores vas a ingresar:10
Ingrese un valor:1
Ingrese un valor:2
Ingrese un valor:3
Ingrese un valor:4
Ingrese un valor:5
Ingrese un valor:6
Ingrese un valor:7
Ingrese un valor:8
Ingrese un valor:1211
Ingrese un valor:2222
Los valores ingresados mayores o igual a 1000 son
2
>>>
```

Capítulo 45.- Estructura repetitiva for – 6

Problema propuesto:

Confeccionar un programa que lea n pares de datos, cada par de datos corresponde a la medida de la base y la altura de un triángulo. El programador deberá informar:

a) De cada triángulo la medida de su base, su altura y su superficie (la superficie se calcula con el producto de la base por la altura entre 2).
b) La cantidad de triángulos cuya superficie sea mayor a 12.

ejercicio45.py - D:/Curso completo de Python/ejercicio45.py (3.11.1)

File Edit Format Run Options Window Help

```
cantidad=0
n=int(input("Cuantos pares de datos desea introducir"))
for x in range(n):
    b=int(input("base:"))
    a=int(input("altura:"))
    superficie=b*a/2
    print("superficie")
    print(superficie)
    if superficie>12:
        cantidad=cantidad+1
print("La cantidad de triangulos con una superficie mayor a 12 son")
print(cantidad)
```

Lo guardamos con el nombre de Ejercicio45.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio45.py =============
Cuantos pares de datos desea introducir3
base:10
altura:10
superficie
50.0
base:3
altura:5
superficie
7.5
base:40
altura:20
superficie
400.0
La cantidad de triangulos con una superficie mayor a 12 son
2
>>>
```

Capítulo 46.- Estructura repetitiva for - 7

Problema propuesto:

Desarrolla un programa que solicite la carga de 10 números e imprima la suma de los últimos 5 valores ingresados:

```
*ejercicio46.py - D:/Curso completo de Python/ejercicio46.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 sumar=0
2 for x in range(10):
3     valor=int(input("Ingrese un valor:"))
4     if x>=5:
5         sumar=sumar+valor
6 print("La suma de los últimos 5 valores ingresados es")
7 print(sumar)
8
```

Lo guardamos con el nombre de Ejercicio46.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio46.py =============
Ingrese un valor:1
Ingrese un valor:1
Ingrese un valor:1
Ingrese un valor:1
Ingrese un valor:1
Ingrese un valor:1
Ingrese un valor:1
Ingrese un valor:1
Ingrese un valor:1
Ingrese un valor:1
La suma de los últimos 5 valores ingresados es
5
>>>
```

Capítulo 47.- Estructura repetitiva for – 8

Problema propuesto:

Desarrolla un programa a que muestre la tabla de multiplicar del 5 (del 5 al 50)

Vamos a programarlo:

```
*ejercicio47.py - D:/Curso completo de Python/ejercicio47.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
for tabla in range(5, 51, 5):
    print(tabla)
```

Guardamos el programa con el nombre de Eejrcicio47.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio47.py =============
5
10
15
20
25
30
35
40
45
50
>>>
```

Intenta realizar el mismo resultado utilizando el while:

```
ejercicio47.py - D:/Curso completo de Python/ejercicio47.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
tabla=5
while tabla<=50:
    print(tabla)
    tabla=tabla+5
```

Capítulo 48.- Estructura repetitiva for – 9

Problema propuesto:

Confeccionar un programa que permita ingresar un valor del 1 al 10 y nos muestre la tabla de multiplicar del mismo (los primeros 12 términos)

Ejemplo: Si ingreso 3 deberá aparecer en pantalla los valores 3, 6, 9, hasta 36.

Vamos a programar:

```
ejercicio48.py - D:/Curso completo de Python/ejercicio48.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 valor=int(input("Que tablas quieres que te muestre del 1 al 10:"))
2 for v in range(1,13):
3     tabla=valor*v
4     print(tabla)
```

Guardamos el programa con el nombre de Ejercicio48.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio48.py =============
    Que tablas quieres que te muestre del 1 al 10:3
    3
    6
    9
    12
    15
    18
    21
    24
    27
    30
    33
    36
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio48.py =============
    Que tablas quieres que te muestre del 1 al 10:5
    5
    10
    15
    20
    25
    30
    35
    40
    45
    50
    55
    60
>>>
                                                                        Ln: 33  Col: 0
```

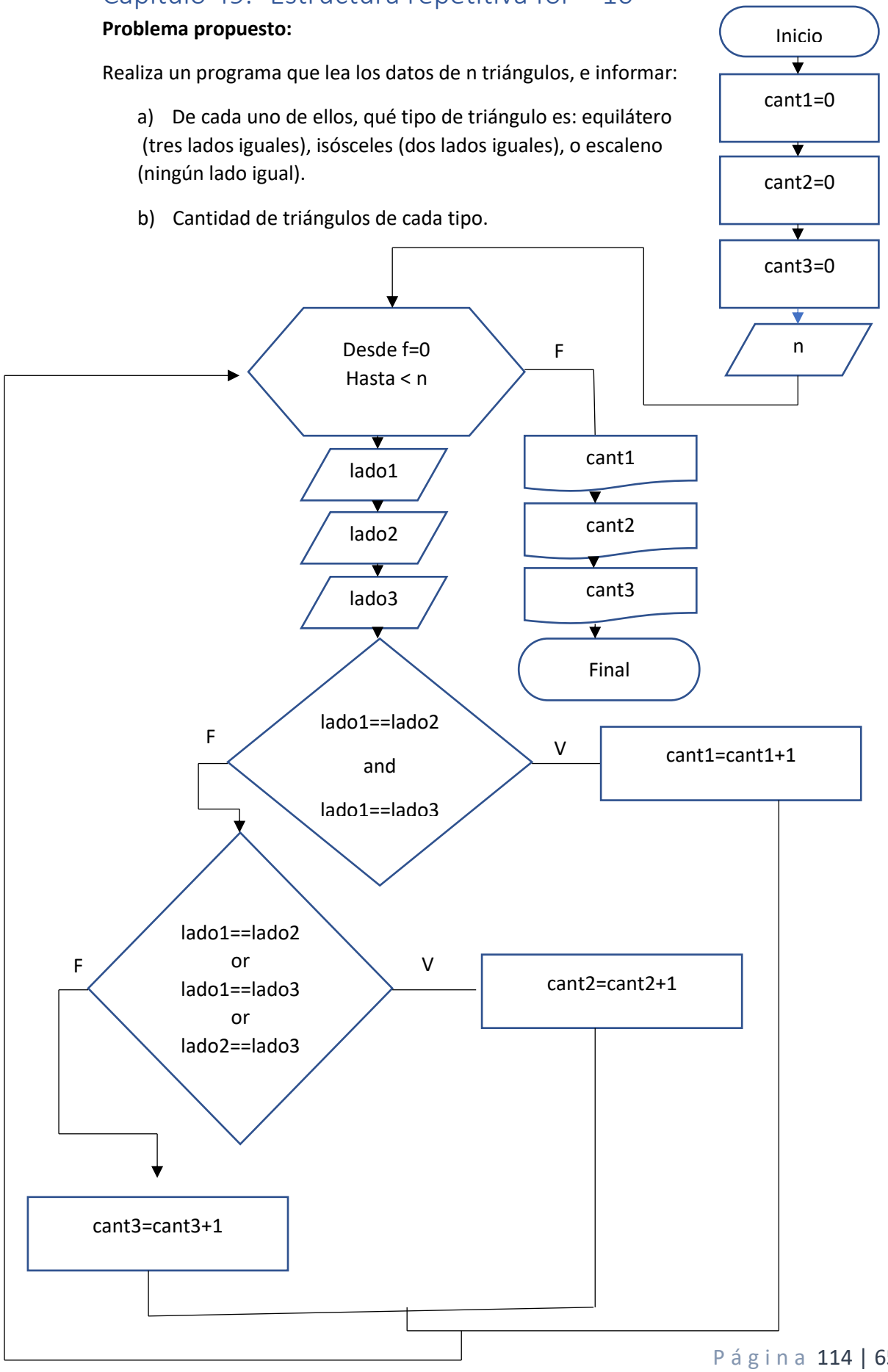
Capítulo 49.- Estructura repetitiva for – 10

Problema propuesto:

Realiza un programa que lea los datos de n triángulos, e informar:

a) De cada uno de ellos, qué tipo de triángulo es: equilátero (tres lados iguales), isósceles (dos lados iguales), o escaleno (ningún lado igual).

b) Cantidad de triángulos de cada tipo.



Vamos a programar:

ejercicio49.py - D:/Curso completo de Python/ejercicio49.py (3.11.1)

File Edit Format Run Options Window Help

```
cant1=0
cant2=0
cant3=0
n=int(input("Ingrese la cantidad de triángulos:"))
for f in range(n):
    lado1=int(input("Ingrese lado 1:"))
    lado2=int(input("Ingrese lado 2:"))
    lado3=int(input("Ingrese lado 3:"))
    if lado1==lado2 and lado1==lado3:
        print("Triangulo equilátero")
        cant1=cant1+1
    else:
        if lado1==lado2 or lado1==lado3 or lado2==lado3:
            print("Triangulo isósceles")
            cant2=cant2+1
        else:
            print("Triangulo escaleno")
            cant3=cant3+1
print("Cantidad de triangulos equilateros")
print(cant1)
print("Cantidad de triangulos isosceles")
print(cant2)
print("Cantidad de triangulos escalenos")
print(cant3)
```

Guardamos el programa con el nombre Ejercicio49.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

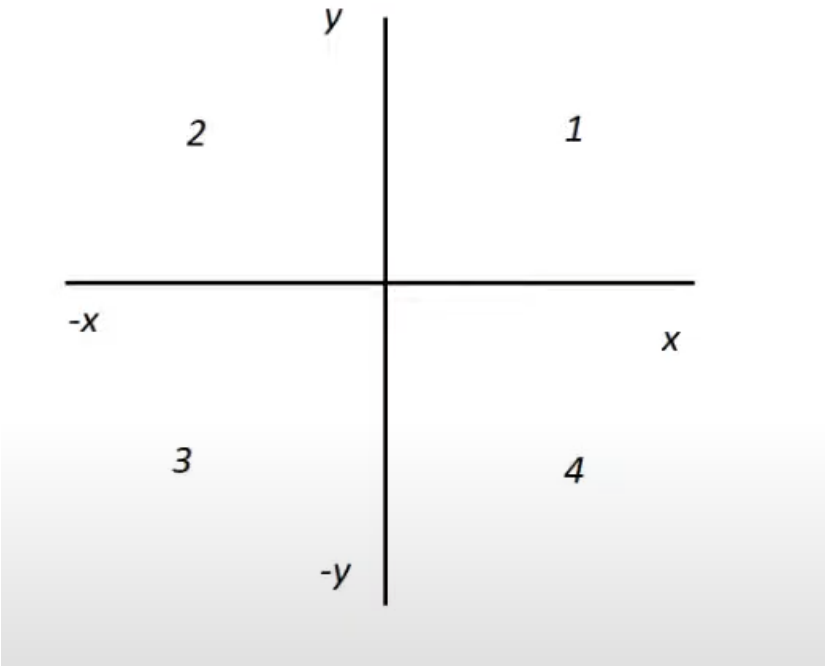
File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio49.py =============
Ingrese la cantidad de triángulos:5
Ingrese lado 1:4
Ingrese lado 2:4
Ingrese lado 3:4
Triangulo equilátero
Ingrese lado 1:6
Ingrese lado 2:6
Ingrese lado 3:2
Triangulo isósceles
Ingrese lado 1:1
Ingrese lado 2:2
Ingrese lado 3:3
Triangulo escaleno
Ingrese lado 1:7
Ingrese lado 2:7
Ingrese lado 3:7
Triangulo equilátero
Ingrese lado 1:6
Ingrese lado 2:6
Ingrese lado 3:2
Triangulo isósceles
Cantidad de triangulos equilateros
2
Cantidad de triangulos isosceles
2
Cantidad de triangulos escalenos
1
>>>
```

Capítulo 50.- Estructura repetitiva for – 11

Problema propuesto:

Escribir un programa que pida ingresar coordenadas (x,y) que representan puntos en el plano. Informar cuántos puntos se han ingresado en el primer plano, segundo, tercer y cuarto cuadrante. Al comenzar el programa se pide que ingrese la cantidad de puntos a procesar.



Vamos a programar:

ejercicio50.py - D:/Curso completo de Python/ejercicio50.py (3.11.1)

File Edit Format Run Options Window Help

```
cant1=0
cant2=0
cant3=0
cant4=0
n=int(input("Cantidad de puntos:"))
for f in range(n):
    x=int(input("Ingrese coordenada x:"))
    y=int(input("Ingrese coordenada y:"))
    if x>0 and y>0:
        cant1=cant1+1
    else:
        if x<0 and y>0:
            cant2=cant2+1
        else:
            if x<0 and y<0:
                cant3=cant3+1
            else:
                if x>0 and y<0:
                    cant4=cant4+1
print("Cantidad de puntos ingresados en el primer cuadrante")
print(cant1)
print("Cantidad de puntos ingresados en el segundo cuadrante")
print(cant2)
print("Cantidad de puntos ingresados en el tercer cuadrante")
print(cant3)
print("Cantidad de puntos ingresados en el cuarto cuadrante")
print(cant4)
```

Vamos a guardar el programa con el nombre Ejercicio50.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio50.py =============
Cantidad de puntos:4
Ingrese coordenada x:3
Ingrese coordenada y:4
Ingrese coordenada x:-4
Ingrese coordenada y:4
Ingrese coordenada x:-1
Ingrese coordenada y:-1
Ingrese coordenada x:4
Ingrese coordenada y:-8
Cantidad de puntos ingresados en el primer cuadrante
1
Cantidad de puntos ingresados en el segundo cuadrante
1
Cantidad de puntos ingresados en el tercer cuadrante
1
Cantidad de puntos ingresados en el cuarto cuadrante
1
>>>
```

Capítulo 51.- Estructura repetitiva for – 12

Problema propuesto:

Se realiza la carga de 10 valores enteros por teclado. Se desea conocer:

a) La cantidad de valores ingresados negativos.
b) La cantidad de valores ingresados positivos.
c) La cantidad de múltiplos de 15.
d) El valor acumulado de los números ingresados que son pares.

Vamos a programar:

ejercicio51.py - D:/Curso completo de Python/ejercicio51.py (3.11.1)

File Edit Format Run Options Window Help

```
neg=0
pos=0
mult15=0
suma_pares=0
for v in range(10):
    valor=int(input("Ingrese un valor: "))
    if valor>0:
        pos=pos+1
    else:
        if valor<0:
            neg=neg+1
    if valor%15==0:
        mult15=mult15+1
    if valor%2==0:
        suma_pares=suma_pares+valor
print("Números positivos")
print(pos)
print("Números negativos")
print(neg)
print("Números múltiplos de 15")
print(mult15)
print("La suma de los números pares")
print(suma_pares)
```

Guardamos el programa con el nombre de Ejercicio51.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio51.py =============
Ingrese un valor: -3
Ingrese un valor: -5
Ingrese un valor: 5
Ingrese un valor: 6
Ingrese un valor: 7
Ingrese un valor: 8
Ingrese un valor: 9
Ingrese un valor: 10
Ingrese un valor: 11
Ingrese un valor: 12
Números positivos
8
Números negativos
2
Números múltiplos de 15
0
La suma de los números pares
36
```

Volvemos a ejecutar:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio51.py =============
Ingrese un valor: 15
Ingrese un valor: 30
Ingrese un valor: 1
Ingrese un valor: 1
Ingrese un valor: 1
Ingrese un valor: 1
Ingrese un valor: 1
Ingrese un valor: 1
Ingrese un valor: 1
Ingrese un valor: 1
Números positivos
10
Números negativos
0
Números múltiplos de 15
2
La suma de los números pares
30
>>>
```

Capítulo 52.- Estructura repetitiva for – 13

Problema propuesto:

Se cuenta con la siguiente información:
Las edades de 5 estudiantes del turno de mañana.
Las edades de 6 estudiantes del turno de tarde.
Las edades de 11 estudiantes del turno de noche.
Las edades de cada estudiante deben ingresarse por teclado.

a) Obtener el promedio de edades de cada turno (tres promedios)
b) Imprimir dichos promedios (promedios de cada turno)
c) Mostrar por pantalla un mensaje que indique cual de los tres turnos tiene un promedio de edades mayor.

ejercicio52.py - D:/Curso completo de Python/ejercicio52.py (3.11.1)

File Edit Format Run Options Window Help

```
pro_mañana=0
pro_tarde=0
pro_noche=0
suma=0
print("Turno de mañana")
for x in range(5):
    nota=int(input("Ingrese nota:"))
    suma=suma+nota
pro_mañana=suma/5
print("Promedio turno mañana")
print(pro_mañana)
suma=0
print("Turno de tarde")
for x in range(6):
    nota=int(input("Ingrese nota:"))
    suma=suma+nota
pro_tarde=suma/6
print("Promedio turno tarde")
print(pro_tarde)
suma=0
print("Turno de noche")
for x in range(11):
    nota=int(input("Ingrese nota:"))
    suma=suma+nota
pro_noche=suma/11
print("Promedio turno noche")
print(pro_noche)
print("El turno con el promedio de mayor edad es")
if pro_mañana>pro_tarde and pro_mañana>pro_noche:
    print("Turno de mañana")
else:
    if pro_tarde>pro_mañana and pro_tarde>pro_noche:
        print("Turno de tarde")
    else:
        print("Turno de noche")
```

Guardamos el programa con el nombre de Ejercicio52.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio52.py =============
Turno de mañana
Ingrese nota:6
Ingrese nota:6
Ingrese nota:6
Ingrese nota:6
Ingrese nota:6
Promedio turno mañana
6.0
Turno de tarde
Ingrese nota:8
Ingrese nota:8
Ingrese nota:8
Ingrese nota:9
Ingrese nota:9
Ingrese nota:9
Promedio turno tarde
8.5
Turno de noche
Ingrese nota:1
Ingrese nota:2
Ingrese nota:22
Ingrese nota:33
Ingrese nota:44
Ingrese nota:3
Ingrese nota:33
Ingrese nota:33
Ingrese nota:33
Ingrese nota:33
Ingrese nota:33
Promedio turno noche
24.545454545454547
El turno con el promedio de mayor edad es
Turno de noche
>>>
Ln: 38  Col: 0
```

Capítulo 53.- Definición de comentarios en el código fuente – 1

Un programa en Python puede definir además del algoritmo propiamente dicho una serie de comentarios en el código fuente que sirvan para aclarar los objetivos de ciertas partes del programa.

Tengamos en cuenta que un programa puede requerir mantenimiento del mismo en un futuro.

Cuando hay que implementar cambios es bueno encontrar en el programa comentarios sobre el objetivo de las distintas partes del algoritmo, sobre todo si es complejo.

Existen dos formas de definir comentarios en Python:

- Comentarios de una sola línea, se emplea el carácter #:

```
# Define tres contadores
conta1=0
conta2=0
conta3=0
```

Todo lo que disponemos después del carácter # no se ejecuta.

- Comentario de varias líneas.

```
"""Definimos tres contadores
   que se muestran si son distintos a cero"""
conta1=0
conta2=0
conta3=0
```

Se deben utilizar tres comillas simples o dobles seguidas al principio y al final del comentario.

Problema:

Mostrar la tabla de multiplicar del 5 empleando primero un while y seguidamente de nuevo empleado el for. (delimitar un comentario previo a cada algoritmo y un comentario del bloque al principio del programa).
Vamos a programar:

ejercicio53.py - D:/Curso completo de Python/ejercicio53.py (3.11.1)

File Edit Format Run Options Window Help

```
1  """
2  Mostrar la tabla de multiplicar de 5 empleando primero el while y seguidamente
3  de nuevo con el for.
4  (definir un comentario previo a cada algoritmo y un comentario de bloque al
5  principio del programa)
6  """
7  print("Tabla de multiplicar del 5")
8  # algoritmo para mostrar la tabla de multiplicar del 5 con while
9  x=5
10 while x<=50:
11     print(x)
12     x=x+5
13
14 print("Tabla del multiplicar del 5")
15 # algoritmo con el for
16 for x in range(5,51, 5):
17     print(x)
```

El programa lo guardamos con el nombre de Ejercicio53.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio53.py =============
Tabla de multiplicar del 5
5
10
15
20
25
30
35
40
45
50
Tabla del multiplicar del 5
5
10
15
20
25
30
35
40
45
50
>>>
                                                                    Ln: 27  Col: 0
```

Capítulo 54.- Definición de comentarios en el código fuente – 2

Problema propuesto:

Realiza un programa que solicite la carga de valores enteros por teclado y los sume. Finalizar la carga al ingresar el valor -1. Deja como comentario dentro del código fuente el enunciado del problema y algún otro comentario de línea pertinente.

ejercicio54.py - D:/Curso completo de Python/ejercicio54.py (3.11.1)

File Edit Format Run Options Window Help

```
1  """
2  Realiza un programa que solicite la carga de valores enteros por
3  teclado y los sume. Finalizar la carga al ingresar el valor -1.
4  Deja como comentario dentro del código fuente el enunciado del
5  problema y algún otro comentario de línea pertinente.
6  """
7  sumar=0
8  valor=int(input("Ingrese un valor -1 para terminar: "))
9  # Ingresando el valor -1 se sale del while.
10 while valor!=-1:
11     sumar=sumar+valor
12     valor=int(input("Ingrese un valor -1 para terminar: "))
13 print("La suma de todos los valores es")
14 print(sumar)
15
```

Guardamos el programa con el nombre Ejercicio54.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio54.py =============
Ingrese un valor -1 para terminar: 10
Ingrese un valor -1 para terminar: 20
Ingrese un valor -1 para terminar: 30
Ingrese un valor -1 para terminar: 40
Ingrese un valor -1 para terminar: 50
Ingrese un valor -1 para terminar: -1
La suma de todos los valores es
150
>>>
```

Capítulo 55.- Definición de comentarios en el código fuente – 3

Problema propuesto:

Confeccionar un programa que solicite la carga de 10 valores reales por teclado. Mostrar al final su suma. Definir varias líneas de comentario indicando el nombre del programa, el programador y la fecha de la última modificación. Utiliza el carácter # para los comentarios.

ejercicio55.py - D:/Curso completo de Python/ejercicio55.py (3.11.1)

File Edit Format Run Options Window Help

```
# Programa: Carga de 10 números reales
# Programador: Diego Moisset
# Fecha de la última modificación: 25/04/2020

sumar=0
for x in range(10):
    valor=float(input("Ingrese un valor: "))
    sumar=sumar+valor
print("La suma de los 10 valores")
print(sumar)
```

Guardamos el programa con el nombre Ejercicio55.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio55.py =============
Ingrese un valor: 1.25
Ingrese un valor: 3.62
Ingrese un valor: 4.08
Ingrese un valor: 10.5
Ingrese un valor: 3.15
Ingrese un valor: 6.02
Ingrese un valor: 2.25
Ingrese un valor: 6.35
Ingrese un valor: 1.85
Ingrese un valor: 10.23
La suma de los 10 valores
49.3
>>>
```

Capítulo 56.- Variables enteras, flotantes y cadena de caracteres – 1

Hasta este momento hemos visto como definir variables enteras y flotantes. Realizar su carga por asignación y por teclado.

Para inicializarlas por asignación utilizamos el operador =

```
# definición de una variable entera
cantidad=20
# definición de una variable flotante
altura=1.92
```

Para realizar la carga por teclado utilizando la función input debemos llamar a la función int o float para convertir el dato devuelto por el input.

```
cantidad=int(input("Ingresar la cantidad de personas:")
altura=float(input("Ingresar la altura de la persona en metros ej: 1.70:"))
```

A estos dos tipos de datos fundamentales (int y float) se suma un tipo de dato muy utilizado que son las cadenas de caracteres.

Una cadena de caracteres está compuesta por uno o más caracteres. También podemos iniciar una cadena de caracteres por asignación o ingresarla por teclado.

Iniciación de una cadena por asignación:

```
# definición e inicio de una cadena de caracteres
dia="lunes"
```

Igual resultado obtenemos si utilizamos la comilla simple:

```
# definición e inicio de una cadena de caracteres
dia='lunes'
```

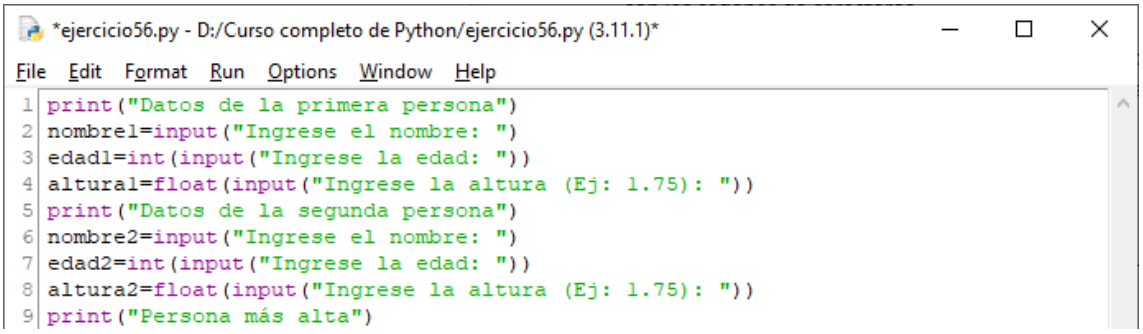
Para la carga por teclado de una cadena de caracteres utilizamos la función input que retorna una cadena de caracteres:

```
nombre=input("Ingrese su nombre:")
```

Problema:

Realiza la carga por teclado del nombre, edad y altura de dos personas. Mostrar por pantalla el nombre de la persona con mayor altura.

Vamos a programar:



```
10 if altura1>altura2:
11     print(nombre1)
12 else:
13     print(nombre2)
14
```

Guardamos el programa con el nombre de Ejercicio56.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio56.py =============
Datos de la primera persona
Ingrese el nombre: Juan
Ingrese la edad: 34
Ingrese la altura (Ej: 1.75): 1.72
Datos de la segunda persona
Ingrese el nombre: Ana
Ingrese la edad: 23
Ingrese la altura (Ej: 1.75): 1.62
Persona más alta
Juan
>>>
```

Capítulo 57.- Variables enteras, flotantes y cadena de caracteres – 2

Problema:

Realizar la carga de dos nombres por teclado. Mostrar cuál de los dos es mayor alfabéticamente o si son iguales.

==	Igualdad
>	Mayor
>=	Mayor o igual
<	Menor
<=	Menor o igual
!=	Distinto

Vamos a programar:

ejercicio57.py - D:/Curso completo de Python/ejercicio57.py (3.11.1)

File Edit Format Run Options Window Help

```
nombre1=input("Ingrese un primer nombre: ")
nombre2=input("Ingrese un segundo nombre: ")
if nombre1>nombre2:
    print(nombre1)
    print("es mayor alfabéticamente")
else:
    if nombre2>nombre1:
        print(nombre2)
        print("es mayor alfabeticamente")
    else:
        print("Son iguales")
```

Guardar el programa con el nombre de Ejercicio57.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio57.py =============
Ingrese un primer nombre: Juan
Ingrese un segundo nombre: Juan
Son iguales
>>>
============== RESTART: D:/Curso completo de Python/ejercicio57.py =============
Ingrese un primer nombre: Luis
Ingrese un segundo nombre: Ana
Luis
es mayor alfabéticamente
>>>
============== RESTART: D:/Curso completo de Python/ejercicio57.py =============
Ingrese un primer nombre: Carlos
Ingrese un segundo nombre: Luis
Luis
es mayor alfabeticamente
>>>
```

Si introducimos dos nombres iguales pero uno en mayúsculas y el otro en minúsculas:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio57.py =============
Ingrese un primer nombre: Ana
Ingrese un segundo nombre: ana
ana
es mayor alfabeticamente
>>>
```

Según la tabla de los caracteres ASCII las letras en minúscula tienen un valor mayor que las mayúsculas.

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Capítulo 58.- Variables enteras, flotantes y cadena de caracteres – 3

Problema:

Realiza la carga de enteros por teclado. Preguntar después que ingresa el valor si se desea cargar otro valor debiendo el operador ingresar la cadena 'si' o 'no' por teclado.

Mostrar la suma de los valores ingresados.

Vamos a programar:

```
*ejercicio58.py - D:/Curso completo de Python/ejercicio58.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 sumar=0
2 opcion='si'
3 while opcion=='si':
4     n=int(input("Ingresa un valor: "))
5     sumar=sumar+n
6     opcion=input("Desea introducir otro valor si/no: ")
7 print("La suma de los valores ingresados es")
8 print(sumar)
9
```

Guardamos el programa con el nombre Ejercicio58.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
  Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
  AMD64)] on win32
  Type "help", "copyright", "credits" or "license()" for more information.
>>>
  ============== RESTART: D:/Curso completo de Python/ejercicio58.py =============
  Ingresa un valor: 10
  Desea introducir otro valor si/no: si
  Ingresa un valor: 20
  Desea introducir otro valor si/no: si
  Ingresa un valor: 30
  Desea introducir otro valor si/no: si
  Ingresa un valor: 40
  Desea introducir otro valor si/no: si
  Ingresa un valor: 50
  Desea introducir otro valor si/no: no
  La suma de los valores ingresados es
  150
>>>
```

Capítulo 59.- Variables enteras, flotantes y cadena de caracteres – 4

Problema propuesto:

Realizar la carga de dos nombres de personas distintos. Mostrar por pantalla luego ordenados en forma alfabética.

Vamos a programar:

```
*ejercicio59.py - D:/Curso completo de Python/ejercicio59.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1  nombre1=input("Dime un primer nombre: ")
2  nombre2=input("Dime un segundo nombre: ")
3  print("Listado en orden alfabético")
4  if nombre1<nombre2:
5      print(nombre1)
6      print(nombre2)
7  else:
8      print(nombre2)
9      print(nombre1)
10
```

Guarda el programa con el nombre de Ejercicio59.py y a continuación lo vamos a ejecutar:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
   Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
   AMD64)] on win32
   Type "help", "copyright", "credits" or "license()" for more information.
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio59.py =============
   Dime un primer nombre: Carlos
   Dime un segundo nombre: Ana
   Listado en orden alfabético
   Ana
   Carlos
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio59.py =============
   Dime un primer nombre: Catalina
   Dime un segundo nombre: Jorge
   Listado en orden alfabético
   Catalina
   Jorge
>>>
```

Capítulo 60.- Procesar cadenas de caracteres – 1

Ya hemos visto que podemos cargar una cadena de caracteres por asignación:

```
# Con doble comillas
cadena1="juan"
# El resultado es igual con simples comillas
cadena2='ana'
```

También podemos cargar por teclado:

```
nombre=input("Ingrese su nombre:")
```

Podemos utilizar los operadores relacionales para identificar si dos cadenas son iguales, distintas o cual es la mayor alfabética:

==	Igualdad
>	Mayor
>=	Mayor o igual
<	Menor
<=	Menor o igual
!=	Distinto

Como su nombre lo indica una cadena de caracteres está formada generalmente por varios caracteres (de todos modos podría tener solo un carácter o ser una cadena vacía).

Podemos acceder en forma individual a cada carácter del string mediante un subíndice:

```
nombre='juan'
print(nombre[0])        # Se imprime una j
if nombre[0]=="j":      # Verificamos si el primer carácter del string es una j
    print(nombre)
    print("Comienza con la letra j")
```

Los subíndices comienzan a numerarse a partir del cero.

Si queremos conocer la longitud de un string en Python disponemos de una función llamada len que retorna la cantidad de caracteres que contiene.

```
nombre='juan'
print(len(nombre))
```

El programa anterior imprime un 4 ya que la cadena nombre almacena 'juan' que tiene cuatro caracteres.

Problema:

Solicita la carga del nombre de una persona en minúsculas. Mostrar un mensaje si comienza con vocal dicho nombre.

Vamos a programar:

```
*ejercicio60.py - D:/Curso completo de Python/ejercicio60.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 nombre=input("Ingrese un nombre de una persona en minúsculas: ")
2 if nombre[0]=='a' or nombre[0]=='e' or nombre[0]=='i' or nombre[0]=='o' or nombre[0]=='u':
3     print("El nombre ingresado comienza con vocal")
4 else:
5     print("El nombre ingresado no comienza con vocal")
6
```

Guardamos el programa con el nombre Ejercicio60.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio60.py =============
    Ingrese un nombre de una persona en minúsculas: ana
    El nombre ingresado comienza con vocal
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio60.py =============
    Ingrese un nombre de una persona en minúsculas: pedro
    El nombre ingresado no comienza con vocal
>>> |
```

Capítulo 61.- Procesar cadenas de caracteres – 2

Problema:

Ingresa un mail por teclado. Verificar si el string ingresado contiene solo un carácter "@".

Vamos a programar:

```
*ejercicio61.py - D:/Curso completo de Python/ejercicio61.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1  cantidad=0
2  arroba=input("Ingrese su correo electrónico: ")
3  for x in range(len(arroba)):
4      if arroba[x]=='@':
5          cantidad=cantidad+1
6  if cantidad==1:
7      print("Contiene solo un caracter @ el mail ingresado")
8  else:
9      print("Incorrecto")
10
```

Vamos a guardar el programa con el nombre Ejercicio61.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
   Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
   AMD64)] on win32
   Type "help", "copyright", "credits" or "license()" for more information.
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio61.py =============
   Ingrese su correo electrónico: diego@hotmail.@gmail
   Incorrecto
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio61.py =============
   Ingrese su correo electrónico: diego@hotmail.com
   Contiene solo un caracter @ el mail ingresado
>>>
```

Otra forma de realizar el programa obteniendo el mismo resultado:

```
ejercicio61.py - D:/Curso completo de Python/ejercicio61.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1  cantidad=0
2  x=0
3  arroba=input("Ingrese su correo electrónico: ")
4  while x<(len(arroba)):
5      if arroba[x]=='@':
6          cantidad=cantidad+1
7      x=x+1
8  if cantidad==1:
9      print("Contiene solo un caracter @ el mail ingresado")
10 else:
11     print("Incorrecto")
12
```

Capítulo 62.- Procesar cadenas de caracteres – 3

Métodos propios de las cadenas de caracteres

Los string en Python son inmutables, esto quiere decir que una vez que los inicialicemos no podemos modificar su contenido:

```
nombre="juan"
nombre[0]="m"   # Esto no se puede hacer.
```

No hay que confundir cambiar parte el string con realizar la asignación de otro string a la misma variable, luego si es correcto asignar otro valor a un string.

```
numbre="juan"
print(nombre)
nombre="ana"
print(nombre)
```

Los string tienen una serie de métodos que nos facilita la creación de nuestros programas.

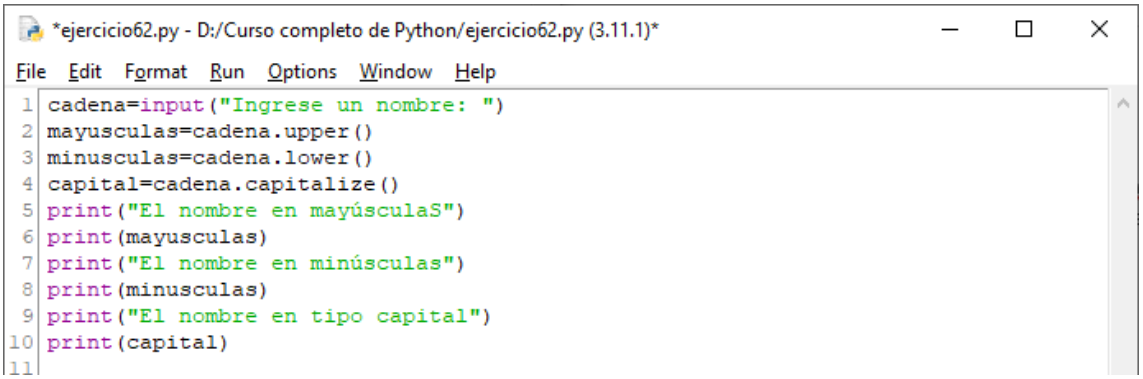
Los primeros tres métodos que veremos se llaman: lower, upper y capitalize.

- upper(): Devuelve una cadena de caracteres convertida todos los caracteres a mayúscula.
- lower(): Devuelve una cadena de caracteres convertida todos sus caracteres a minúsculas.
- capitalize(): Devuelve una cadena de caracteres convertida a mayúsculas solo su primer carácter y todos los demás a minúsculas.

Problema:

Inicializar un string con la cadena "mAriA" luego llamar a sus métodos upper(), lower() y capitalize(), guardar los datos retornados en otros string y mostrar en pantalla.

Vamos a programar:



Guardamos el programa con el nombre Ejercicio62.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio62.py =============
Ingrese un nombre: mAriA
El nombre en mayúsculaS
MARIA
El nombre en minúsculas
maria
El nombre en tipo capital
Maria
>>>
```

El valor de la variable cadena se mantiene con el valor de mAriA.

Capítulo 63.- Procesar cadenas de caracteres – 4

Problema propuesto:

Carga una oración por teclado. Mostrar luego cuantos espacios en blanco ingresamos. Tener en cuenta que un espacio en blanco es igual a " ", en cambio una cadena vacía es "".

Vamos a programar:

```
*ejercicio63.py - D:/Curso completo de Python/ejercicio63.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1  # Método while
2  oracion=input("Ingrese una oración: ")
3  cantidad=0
4  x=0
5  while x<len(oracion):
6      if oracion[x]==" ":
7          cantidad=cantidad+1
8      x=x+1
9  print("El número de espacios en blanco es")
10 print(cantidad)
11
12 # Método for
13 oracion=input("Ingrese una oración: ")
14 cantidad=0
15 for x in range(len(oracion)):
16     if oracion[x]==" ":
17         cantidad=cantidad+1
18 print("El número de espacios en blanco es")
19 print(cantidad)
20
```

Guardamos el programa con el nombre Ejercicio63.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio63.py =============
Ingrese una oración: La casa de la montaña
El número de espacios en blanco es
4
Ingrese una oración: La casa de la montaña
El número de espacios en blanco es
4
>>>
```

Capítulo 64.- Procesar cadenas de caracteres – 5

Problema propuesto:

Ingresar una oración que puede tener letras tanto en mayúsculas como minúsculas. Contar la cantidad de vocales.

Crear un segundo string con toda la oración en minúsculas para que sea más fácil disponer la condición que verifica que es una vocal.

Vamos a programar:

ejercicio64.py - D:/Curso completo de Python/ejercicio64.py (3.11.1)

File Edit Format Run Options Window Help

```
oracion1=input("Ingrese una oración intercalando mayúsculas y minúsculas: ")
oracion2=oracion1.lower()
# Método for
vocal=0
for x in range(len(oracion2)):
    if oracion2[x]=='a' or oracion2[x]=='e' or oracion2[x]=='i'or oracion2[x]=='o' or oracion2[x]=='u':
        vocal=vocal+1
print("El número de vocales son")
print(vocal)

# Método while
x=0
vocal=0
while x<len(oracion2):
    if oracion2[x]=='a' or oracion2[x]=='e' or oracion2[x]=='i'or oracion2[x]=='o' or oracion2[x]=='u':
        vocal=vocal+1
    x=x+1
print("El número de vocales son")
print(vocal)
```

El programa lo guardamos con el nombre de Ejercicio64.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio64.py =============
Ingrese una oración intercalando mayúsculas y minúsculas: Alberto, Antonio y Ana
son los amigos de la infancia.
El número de vocales son
20
El número de vocales son
20
>>>
```

Capítulo 65.- Procesar cadenas de caracteres – 6

Problema propuesto:

Solicitar el ingreso de una clave por teclado y almacenarla en una cadena de caracteres. Controlar que el string ingresado tenga entre 10 y 20 caracteres par que este sea válido, en caso contrario mostrar un mensaje de error.

Vamos a programar:

```
*ejercicio65.py - D:/Curso completo de Python/ejercicio65.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 clave=input("Ingrese una clave de entre 10 y 20 caracteres: ")
2 if len(clave)>=10 and len(clave)<=20:
3     print("Clave correcta")
4 else:
5     print("Clave incorrecta")
6
```

Guardamos el programa con el nombre de Ejercicio65.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio65.py =============
Ingrese una clave de entre 10 y 20 caracteres: casa
Clave incorrecta
>>>
============== RESTART: D:/Curso completo de Python/ejercicio65.py =============
Ingrese una clave de entre 10 y 20 caracteres: supercalifrastilistico
Clave incorrecta
>>>
============== RESTART: D:/Curso completo de Python/ejercicio65.py =============
Ingrese una clave de entre 10 y 20 caracteres: estoesgrande
Clave correcta
>>>
```

Capítulo 66.- Estructura de datos tipo lista – 1

Hasta ahora hemos trabajado con variables que permiten almacenar un único valor:

edad=12
altura=1.79
nombre="juan"

En Python existe un tipo de variable que permite almacenar una colección de datos y luego acceder por medio de un subíndice (similar a los strings).

Creación de la lista por asignación

Para crear una lista por asignación debemos indicar sus elementos encerrados entre corchetes y separados por coma.

```
lista1=[10, 5, 3]                          # Lista de enteros
lista2=[1.78, 2.66, 1.55, 89.4]            # Lista de valores float
lista3=["lunes", "martes", "miércoles"]    # Lista de string
lista4=["juan", 45, 1.92]                  # Lista con elementos de distinto tipo
```

Si queremos conocer la cantidad de elementos de una lista podemos llamar a la función len:

```
lista1=[10, 5, 3]        # Lista de enteros
print(len(lista1))       # Imprime un 3
```

Problema

Definir una lista que almacene 5 enteros. Sumar todos sus elementos y mostrar dicha suma.

Vamos a programar:

ejercicio66.py - D:/Curso completo de Python/ejercicio66.py (3.11.1)

File Edit Format Run Options Window Help

```
1  lista=[10, 20, 30, 40, 50]
2  # Método for
3  suma=0
4  for x in range(len(lista)):
5      suma=suma+lista[x]
6  print("Los elementos de la lista")
7  print(lista)
8  print("La suma de sus elementos")
9  print(suma)
10
11 # Método while
12 suma=0
13 x=1
14 while x<len(lista):
15     suma=suma+lista[x]
16     x=x+1
17 print("Los elementos de la lista")
18 print(lista)
19 print("La suma de sus elementos")
20 print(suma)
21
```

Guardamos el programa con el nombre de Ejercicio66.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio66.py =============
Los elementos de la lista
[10, 20, 30, 40, 50]
La suma de sus elementos
150
Los elementos de la lista
[10, 20, 30, 40, 50]
La suma de sus elementos
140
>>>
```

Capítulo 67.- Estructura de datos tipo lista – 2

Problema

Definir una lista por asignación que almacene los nombres de los primeros cuatro meses del año.

Mostrar el primer y último elemento de la lista solamente.

Vamos a programar:

```
*ejercicio67.py - D:/Curso completo de Python/ejercicio67.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 meses=["Enero", "Febrero", "Marzo", "Abril"]
2 print(meses)
3 print("Primer mes del año")
4 print(meses[0])
5 print("El cuarto mes del año")
6 print(meses[3])
7
```

Guardar el nombre el programa como Ejercicio67.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio67.py =============
    ['Enero', 'Febrero', 'Marzo', 'Abril']
    Primer mes del año
    Enero
    El cuarto mes del año
    Abril
>>>
```

Si en la línea 6 ponemos print(meses[4]) observaríamos el siguiente error:

```
['Enero', 'Febrero', 'Marzo', 'Abril']
Primer mes del año
Enero
El cuarto mes del año
Traceback (most recent call last):
  File "D:/Curso completo de Python/ejercicio67.py", line 6, in <module>
    print(meses[4])
IndexError: list index out of range
```

Índice fuera de rango.

Capítulo 68.- Estructura de datos tipo lista – 3

Problema:

Definir una lista por asignación que almacene el la primer componente el nombre de un alumno y en las dos siguientes sus notas. Imprimir luego el nombre y el promedio de las dos notas.

Vamos a programar:

ejercicio68.py - D:/Curso completo de Python/ejercicio68.py (3.11.1)

File Edit Format Run Options Window Help

```
alumno=["Juan", 6, 9]
# Primer método
print("El nombre del alumno es")
print(alumno[0])
print("Promedio de su dos notas")
promedio=(alumno[1]+alumno[2])/2
# Segundo método
suma=0
for x in range(1, len(alumno)):
    suma=suma+alumno[x]
print("El nombre del alumno es")
print(alumno[0])
print("El promedio de sus dos notas")
print(suma/2)
# Tercer método
x=1
suma=0
while x<len(alumno):
    suma=suma+alumno[x]
    x=x+1
print("El nombre del alumno es")
print(alumno[0])
print("El promedio de sus dos notas")
print(suma/2)
```

Guardamos el programa con el nombre de Ejercicio68.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio68.py =============
El nombre del alumno es
Juan
Promedio de su dos notas
El nombre del alumno es
Juan
El promedio de sus dos notas
7.5
El nombre del alumno es
Juan
El promedio de sus dos notas
7.5
>>>
```

Capítulo 69.- Estructura de datos tipo lista – 4

Problema propuesto:

Definir por asignación una lista con 8 elementos enteros. Contar cuantos de dichos valores almacenan un valor superior a 100.

Vamos a programar:

ejercicio69.py - D:/Curso completo de Python/ejercicio69.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[65, 110, 30, 125, 40, 168, 48, 210]
# Prime método (for)
cantidad=0
print(lista)
for x in range(len(lista)):
    if lista[x]>100:
        cantidad=cantidad+1
print("La lista contiene los valores")
print(lista)
print("Los elementos con valores superiores a 100 hay")
print(cantidad)

# Segundo método (while)
cantidad=0
x=0
while x<len(lista):
    if lista[x]>100:
        cantidad=cantidad+1
    x=x+1
print("La lista contiene los valores")
print(lista)
print("Los elementos con valores superiores a 100 hay")
print(cantidad)

```

Guardamos el programa con el nombre de Ejercicio69.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio69.py ==============
[65, 110, 30, 125, 40, 168, 48, 210]
La lista contiene los valores
[65, 110, 30, 125, 40, 168, 48, 210]
Los elementos con valores superiores a 100 hay
4
La lista contiene los valores
[65, 110, 30, 125, 40, 168, 48, 210]
Los elementos con valores superiores a 100 hay
4
>>>
>>>
```

Capítulo 70.- Estructura de datos tipo lista – 5

Problema propuesto:

Definir una lista por asignación con 5 enteros. Mostrar por pantalla solo los elementos con valor iguales o superiores a 7.

Vamos a programar:

ejercicio70.py - D:/Curso completo de Python/ejercicio70.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[9, 3, 5, 8, 7]
print(lista)
# Primer método (for)
print("Método for")
for x in range(len(lista)):
    if lista[x]>=7:
        print(lista[x])

# Segundo método (while)
print("Método while")
x=0
while x<len(lista):
    if lista[x]>=7:
        print(lista[x])
    x=x+1
```

Guardamos el programa con el nombre de Ejercicio70.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio70.py =============
[9, 3, 5, 8, 7]
Método for
9
8
7
Método while
9
8
7
>>>
```

Capítulo 71.- Estructuras de datos tipo lista – 6

Problema propuesto:

Definir una lista que almacene por asignación los nombres de 5 personas. Contar cuantos de esos nombres tienen más de 5 caracteres.

Vamos a programar:

ejercicio71.py - D:/Curso completo de Python/ejercicio71.py (3.11.1)

File Edit Format Run Options Window Help

```python
lista=["Antoio", "Juan", "Dolores", "Luis", "Federico"]
print(lista)

# Método for
print("Con el método for")
cantidad=0
for x in range(len(lista)):
    if len(lista[x])>=5:
        cantidad=cantidad+1
print("Los nombres que tienen 5 o más caracteres son")
print(cantidad)

# Método while
print("Con el método while")
cantidad=0
x=0
while x<len(lista):
    if len(lista[x])>=5:
        cantidad=cantidad+1
    x=x+1
print("Los nombres que tienen 5 o más caracteres son")
print(cantidad)

```

Guardamos el programa con el nombre de Ejercicio71.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio71.py =============
['Antoio', 'Juan', 'Dolores', 'Luis', 'Federico']
Con el método for
Los nombres que tienen 5 o más caracteres son
3
Con el método while
Los nombres que tienen 5 o más caracteres son
3
>>>
```

Capítulo 72.- Listas: carga por teclado de sus elementos – 1

Una lista en Python es una estructura mutable (es decir puede ir cambiando durante la ejecución del programa)

Hemos visto que podemos definir una lista por asignación indicando entre corchete los valores a almacenar:

lista=[10, 20, 30]

Una lista luego de definirla podemos agregarle nuevos elementos a la colección. La primera forma que veremos para que nuestra lista crezca es utilizar el método append que tiene la lista y pasar como parámetro el nuevo elemento:

```
lista=[10, 20, 30]
print(len(lista))        # Imprime un 3
lista.append(100)
print(len(lista))        # Imprime un 4
print(lista[0])          # Imprime un 10
print(lista[3]           # Imprime un 100
```

Problema:

Definir una lista vacía y luego solicitar la carga de 5 enteros por teclado y añadirlos a la lista. Imprimir la lista generada.

Vamos a programar:

ejercicio72.py - D:/Curso completo de Python/ejercicio72.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[]
# Método for
print("Con el método for")
for x in range(5):
    valor=int(input("Ingrese un valor: "))
    lista.append(valor)
print(lista)

# Método while
print("Con el método while")
x=0
while x<5:
    valor=int(input("Ingrese un valor: "))
    lista.append(valor)
    x=x+1
print(lista)

```

Guardamos el programa con el nombre de Ejercicio72.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

```
============== RESTART: D:/Curso completo de Python/ejercicio72.py =============
Con el método for
Ingrese un valor: 7
Ingrese un valor: 9
Ingrese un valor: 12
Ingrese un valor: 14
Ingrese un valor: 3
[7, 9, 12, 14, 3]
Con el método while
Ingrese un valor: 12
Ingrese un valor: 25
Ingrese un valor: 34
Ingrese un valor: 9
Ingrese un valor: 10
[7, 9, 12, 14, 3, 12, 25, 34, 9, 10]
>>>
```

ejercicio72.py - D:/Curso completo de Python/ejercicio72.py (3.11.1)

File Edit Format Run Options Window Help

```
1  lista=[]
2  # Método for
3  print("Con el método for")
4  for x in range(5):
5      valor=int(input("Ingrese un valor: "))
6      lista.append(valor)
7  print(lista)
8
9  # Método while
10 print("Con el método while")
11 x=0
12 while x<5:
13     valor=int(input("Ingrese un valor: "))
14     lista.append(valor)
15     x=x+1
16 print(lista)
17 print(len(lista))
18
```

En la línea 17 nos muestra el número de elementos que tiene la lista.

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio72.py =============
Con el método for
Ingrese un valor: 1
Ingrese un valor: 2
Ingrese un valor: 3
Ingrese un valor: 4
Ingrese un valor: 5
[1, 2, 3, 4, 5]
Con el método while
Ingrese un valor: 6
Ingrese un valor: 7
Ingrese un valor: 8
Ingrese un valor: 9
Ingrese un valor: 10
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
10
>>>
```

Capítulo 73.- Listas: carga por teclado de sus elementos – 2

Problema:

Realiza una carga de valores enteros por teclado, almacenarlos en una lista. Finalizar la carga de enteros al ingresar el cero. Mostrar finalmente el tamaño de la lista y el contenido de la lista.

Vamos a programar:

```
*ejercicio73.py - D:/Curso completo de Python/ejercicio73.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
lista=[]
valor=int(input("Ingresa un valor: "))
while valor!=0:
    lista.append(valor)
    valor=int(input("Ingresa un valor: "))
print("Tamaño de la lista")
print(len(lista))
print("Contenido de la lista")
print(lista)
```

Guardamos el programa con el nombre de Ejercicio73.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio73.py =============
Ingresa un valor: 10
Ingresa un valor: 20
Ingresa un valor: 30
Ingresa un valor: 40
Ingresa un valor: 50
Ingresa un valor: 0
Tamaño de la lista
5
Contenido de la lista
[10, 20, 30, 40, 50]
>>>
```

Capítulo 74.- Listas: carga por teclado de sus elementos – 3

Problema propuesto:

Almacenar en una lista los sueldos (valores float) de 5 operarios. Imprimir la lista y el promedio de sueldos.

Vamos a programar:

```
ejercicio74.py - D:/Curso completo de Python/ejercicio74.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1  sueldos=[]
2  suma=0
3  for x in range(5):
4      valor=float(input("Ingrese el sueldo: "))
5      sueldos.append(valor)
6  print(sueldos)
7  for x in range(len(sueldos)):
8      suma=suma+sueldos[x]
9  print("El prodemdio es")
10 print(suma/len(sueldos))
11
```

Guardamos el programa con el nombre de Ejercicio74.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
   Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
   AMD64)] on win32
   Type "help", "copyright", "credits" or "license()" for more information.
>>>
   ============== RESTART: D:/Curso completo de Python/ejercicio74.py =============
   Ingrese el sueldo: 1000
   Ingrese el sueldo: 2000
   Ingrese el sueldo: 3000
   Ingrese el sueldo: 4000
   Ingrese el sueldo: 5000
   [1000.0, 2000.0, 3000.0, 4000.0, 5000.0]
   El prodemdio es
   3000.0
>>>
```

Capítulo 75.- Listas: carga por teclado de sus elementos – 4

Problema propuesto:

Cargar por teclado y almacenar en una lista las alturas de 5 personas (valores float). Obtener el promedio de las mismas. Contar cuantas personas son más altas que el promedio y cuantas más bajas.

Vamos a programar:

ejercicio75.py - D:/Curso completo de Python/ejercicio75.py (3.11.1)

File Edit Format Run Options Window Help

```
alturas=[]
cantidad=0
suma=0
altas=0
bajas=0
for x in range(5):
    valor=float(input("Ingrese la altura. Ej.: 1.82: "))
    alturas.append(valor)
    suma=suma+valor
promedio=suma/5
for x in range(5):
    if alturas[x]>promedio:
        altas=altas+1
    else:
        bajas=bajas+1
print("Las alturas de las personas")
print(alturas)
print("El promedio de las alturas es")
print(promedio)
print("El número de alturas mayores al promedio son: ")
print(altas)
print("El número de alturas menores al promedio son: ")
print(bajas)
```

Guardamos el programa con el nombre de Ejercicio75.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio75.py =============
Ingrese la altura. Ej.: 1.82: 1.56
Ingrese la altura. Ej.: 1.82: 1.68
Ingrese la altura. Ej.: 1.82: 1.92
Ingrese la altura. Ej.: 1.82: 1.68
Ingrese la altura. Ej.: 1.82: 1.72
Las alturas de las personas
[1.56, 1.68, 1.92, 1.68, 1.72]
El promedio de las alturas es
1.7120000000000002
El número de alturas mayores al promedio son:
2
El número de alturas menores al promedio son:
3
>>>
```

Capítulo 76.- Listas: carga por teclado de sus elementos – 5

Problema propuesto:

Una empresa tiene dos turnos (mañana y tarde) en los que trabajan 8 empleados (4 por la mañana y 4 por la tarde). Confeccionar un programa que permita almacenar los sueldos de los empleados agrupados en dos listas.

Imprimir las dos listas de sueldos.

Vamos a programar:

ejercicio76.py - D:/Curso completo de Python/ejercicio76.py (3.11.1)

File Edit Format Run Options Window Help

```
mañana=[]
tarde=[]
print("Turno de mañana:")
for x in range(4):
    valor=float(input("Ingrese el sueldo: "))
    mañana.append(valor)
print("Turno de tarde:")
for x in range(4):
    valor=float(input("Ingrese el sueldo: "))
    tarde.append(valor)
print("Sueldos turno de mañana")
print(mañana)
print("Sueldos turno de tarde")
print(tarde)
```

Guardamos el programa con el nombre de Ejercicio76.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio76.py =============
Turno de mañana:
Ingrese el sueldo: 1000
Ingrese el sueldo: 2000
Ingrese el sueldo: 1200
Ingrese el sueldo: 1700
Turno de tarde:
Ingrese el sueldo: 4000
Ingrese el sueldo: 4300
Ingrese el sueldo: 2000
Ingrese el sueldo: 1700
Sueldos turno de mañana
[1000.0, 2000.0, 1200.0, 1700.0]
Sueldos turno de tarde
[4000.0, 4300.0, 2000.0, 1700.0]
>>>
```

Capítulo 77.- Listas: mayor y menor elemento – 1

Es una actividad muy común la búsqueda del mayor y menor elemento de una lista.

Es necesario que la lista tenga valores del mismo tipo por ejemplo enteros. Pueden ser de tipo cadena de caracteres y se busque cual es mayor o menor alfabéticamente, pero no podemos buscar el mayor o menor si la lista tiene enteros y cadenas de caracteres al mismo tiempo.

Problema:

Crear y cargar una lista de 5 enteros. Implementar un algoritmo que identifique el mayor valor de la lista.

Vamos a programar:

ejercicio77.py - D:/Curso completo de Python/ejercicio77.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[]
for x in range(5):
    valor=int(input("Ingrese valor: "))
    lista.append(valor)
mayor=lista[0]
for x in range(5):
    if lista[x]>mayor:
        mayor=lista[x]
print("Lista completa")
print(lista)
print("El valor mayor es")
print(mayor)
```

Guardar el programa con el nombre de Ejercicio77.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio77.py =============
Ingrese valor: 56
Ingrese valor: 77
Ingrese valor: 444
Ingrese valor: 3
Ingrese valor: 4
Lista completa
[56, 77, 444, 3, 4]
El valor mayor es
444
>>>
```

Capítulo 78.- Listas: mayor y menor elemento – 2

Problema:

Crear y cargar una lista con 5 enteros por teclado. Implementar un algoritmo que identifique el menor valor de la lista y la posición donde se encuentra.

Vamos a programar:

ejercicio78.py - D:/Curso completo de Python/ejercicio78.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[]
posicion=0
for x in range(5):
    valor=int(input("Ingrese valor: "))
    lista.append(valor)
menor=lista[0]
for x in range(5):
    if lista[x]<menor:
        menor=lista[x]
        posicion=x
print("Lista completa")
print(lista)
print("El valor menor es")
print(menor)
print("Su posición en la lista es")
print(posicion)
```

Guardamos el programa con el nombre de Ejercicio78.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio78.py =============
Ingrese valor: 200
Ingrese valor: 150
Ingrese valor: 100
Ingrese valor: 700
Ingrese valor: 600
Lista completa
[200, 150, 100, 700, 600]
El valor menor es
100
Su posición en la lista es
2
>>>
```

Capítulo 79.- Listas: mayor y menor elemento – 3

Problema propuesto:

Ingresar por teclado los nombres de 5 personas y almacenarlos en una lista. Mostrar el nombre de la persona menor en orden alfabético.

Vamos a programar:

```
ejercicio79.py - D:/Curso completo de Python/ejercicio79.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1  nombres=[]
2  for x in range(5):
3      valor=(input("Ingrese valor: "))
4      nombres.append(valor)
5  menor=nombres[0]
6  for x in range(5):
7      if nombres[x]<menor:
8          menor=nombres[x]
9  print("Lista completa")
10 print(nombres)
11 print("El nombre menor en orden alfabético")
12 print(menor)
13
```

Guardar el programa con el nombre de Ejercicio79.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio79.py =============
Ingrese valor: Carlos
Ingrese valor: Ana
Ingrese valor: Luis
Ingrese valor: Pedro
Ingrese valor: Juan
Lista completa
['Carlos', 'Ana', 'Luis', 'Pedro', 'Juan']
El nombre menor en orden alfabético
Ana
>>>
```

Capítulo 80.- Listas: mayor y menor elemento – 4

Problema propuesto:

Cargar una lista con 5 elementos enteros. Imprimir el mayor y un mensaje si se repite dentro de la lista(es decir si dicho valor se encuentra en 2 o más posiciones de la lista)

Vamos a programar:

ejercicio80.py - D:/Curso completo de Python/ejercicio80.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[]
cantidad=0
for x in range(5):
    valor=int(input("Ingrese un valor: "))
    lista.append(valor)
mayor=lista[0]
print("Valores de la lista")
print(lista)
for x in range(5):
    if lista[x]>mayor:
        mayor=lista[x]
print("El valor mayor es")
print(mayor)
for x in range(5):
    if lista[x]==mayor:
        cantidad=cantidad+1
    if cantidad>1:
        print("El valor se encuentra repetido en la lista")
```

Guardar el programa con el nombre de Ejercicio80.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio80.py =============
Ingrese un valor: 10
Ingrese un valor: 100
Ingrese un valor: 60
Ingrese un valor: 20
Ingrese un valor: 100
Valores de la lista
[10, 100, 60, 20, 100]
El valor mayor es
100
El valor se encuentra repetido en la lista
>>>
============== RESTART: D:/Curso completo de Python/ejercicio80.py =============
Ingrese un valor: 1
Ingrese un valor: 2
Ingrese un valor: 3
Ingrese un valor: 4
Ingrese un valor: 5
Valores de la lista
[1, 2, 3, 4, 5]
El valor mayor es
5
>>>
```

Capítulo 81.- Listas paralelas – 1

Podemos decir que dos listas son paralelas cuando hay una relación entre los componentes de igual subíndice (misma posición) de una lista y otra.

nombres	Juan	Ana	Marcos	Pablo	Laura
edades	12	21	27	14	21

Si tenemos dos listas que ya hemos inicializado con 5 elementos cada una. En una se almacenan los nombres de personas en la otra las edades de dichas personas.
Decimos que la lista nombres es paralela a la lista edades si en el componente 0 de cada lista se almacena información relacionada a una persona (Juan – 12 años).
Es decir hay una relación entre cada componente de las dos listas.
Esta relación la conoce únicamente el programador y se hace para facilitar el desarrollo de algoritmos que procesen los datos almacenados en la estructura de datos.

Problema:
Desarrollar un programa que permita cargar 5 nombres de personas y sus edades. Luego de realizar la carga por teclado de todos los datos imprimir los nombres de las personas mayores de edad (mayores o iguales a 18 años).

Vamos a programar:

ejercicio81.py - D:/Curso completo de Python/ejercicio81.py (3.11.1)

File Edit Format Run Options Window Help

```
nombres=[]
edades=[]
for x in range(5):
    nom=input("Ingrese el nombre: ")
    nombres.append(nom)
    eda=int(input("Ingrese su edad: "))
    edades.append(eda)
print("Listado de las personas mayores de edad")
for x in range(5):
    if edades[x]>=18:
        print(nombres[x])
```

Guardamos el programa con el nombre de Ejercicio81.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio81.py =============
Ingrese el nombre: Ana
Ingrese su edad: 12
Ingrese el nombre: Juan
Ingrese su edad: 45
Ingrese el nombre: Pedro
Ingrese su edad: 23
Ingrese el nombre: Carlos
Ingrese su edad: 17
Ingrese el nombre: Luis
Ingrese su edad: 78
Listado de las personas mayores de edad
Juan
Pedro
Luis
```

Capítulo 82.- Listas paralelas – 2

Problema propuesto:

Crear y cargar dos listas con los nombres de 5 productos en una y sus respectivos precios en otra.

Definir dos listas paralelas. Mostrar cuantos productos tienen un precio mayor al primer producto ingresado.

Vamos a programar:

ejercicio81.py - D:/Curso completo de Python/ejercicio81.py (3.11.1)

File Edit Format Run Options Window Help

```
productos=[]
precios=[]
for x in range(5):
    producto=input("Ingrese el producto: ")
    productos.append(producto)
    precio=float(input("Ingrese el precio de dicho producto: "))
    precios.append(precio)
cantidad=0
for x in range(1, 5):
    if precios[x]>precios[0]:
        cantidad=cantidad+1
print("Cantidad de productos con un precio")
print("mayor al primer producto ingresado")
print(cantidad)
```

Guardamos el programa con el nombre de Ejercicio82.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio81.py =============
Ingrese el producto: Patatas
Ingrese el precio de dicho producto: 30
Ingrese el producto: manzanas
Ingrese el precio de dicho producto: 50
Ingrese el producto: peras
Ingrese el precio de dicho producto: 70
Ingrese el producto: cebollas
Ingrese el precio de dicho producto: 20
Ingrese el producto: lechuga
Ingrese el precio de dicho producto: 15
Cantidad de productos con un precio
mayor al primer producto ingresado
2
>>>
```

Capítulo 83.- Listas paralelas – 3

Problema propuesto:

En un curso de 4 alumnos se registraron las notas de sus exámenes y se deben procesar de acuerdo a lo siguiente:

a) Ingresar nombre y nota de cada alumno (almacenar los datos en dos listas paralelas)
b) Realizar un listado que muestre el nombre, notas y condición del alumno. En la condición, colocar "Muy bueno" si la nota es mayor o igual a 8, "Bueno" si la nota está entre 4 y 7, y colocar "Insuficiente" si la nota es inferior a 4.
c) Imprimir cuantos alumnos tienen la leyenda "Muy Bueno".

Vamos a programar:

ejercicio83.py - D:/Curso completo de Python/ejercicio83.py (3.11.1)

File Edit Format Run Options Window Help

```
alumnos=[]
notas=[]
for x in range(4):
    alumno=input("Ingrese el nombre del alumno: ")
    alumnos.append(alumno)
    nota=int(input("Ingrese la nota del alumno: "))
    notas.append(nota)
cantidad=0
for x in range(4):
    print(alumnos[x])
    print(notas[x])
    if notas[x]>=8:
        print("Muy bueno")
        cantidad=cantidad+1
    else:
        if notas[x]>=4:
            print("Bueno")
        else:
            print("Insuficiente")
print("Cantidad de alumnos muy buenos")
print(cantidad)
```

Guardamos el programa con el nombre de Ejercicio83.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio83.py ==============
Ingrese el nombre del alumno: Juan
Ingrese la nota del alumno: 9
Ingrese el nombre del alumno: Luis
Ingrese la nota del alumno: 6
Ingrese el nombre del alumno: Carlos
Ingrese la nota del alumno: 4
Ingrese el nombre del alumno: Ana
Ingrese la nota del alumno: 2
```

```
Juan
9
Muy bueno
Luis
6
Bueno
Carlos
4
Bueno
Ana
2
Insuficiente
Cantidad de alumnos muy buenos
1
```

Capítulo 84.- Listas paralelas – 4

Problema propuesto:

Realizar un programa que pida la carga de dos listas numéricas entreras de 4 elementos cada una.

Generar una tercera lista que surja de la suma de los elementos de la misma posición de cada lista.

Mostar esta tercera lista.

Vamos a programar:

ejercicio84.py - D:/Curso completo de Python/ejercicio84.py (3.11.1)

File Edit Format Run Options Window Help

```
lista1=[]
lista2=[]
lista3=[]
print("Introducir los valores a la primera lista")
for x in range(4):
    lis1=int(input("Ingrese un número entero: "))
    lista1.append(lis1)
print("Introducir los valores a la segunda lista")
for x in range(4):
    lis2=int(input("Ingrese un número entero: "))
    lista2.append(lis2)
print("Crando la tercera lista")
for x in range(4):
    lista3.append(lista1[x]+lista2[x])
print("Mostrando la primera lista")
print(lista1)
print("Mostrando la segunda lista")
print(lista2)
print("Mostrando la tercera lista")
print(lista3)
```

Guardamos el programa con el nombre de Ejercicio84.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio84.py =============
Introducir los valores a la primera lista
Ingrese un número entero: 1
Ingrese un número entero: 2
Ingrese un número entero: 3
Ingrese un número entero: 4
Introducir los valores a la segunda lista
Ingrese un número entero: 5
Ingrese un número entero: 5
Ingrese un número entero: 5
Ingrese un número entero: 5
Crando la tercera lista
Mostrando la primera lista
[1, 2, 3, 4]
Mostrando la segunda lista
[5, 5, 5, 5]
Mostrando la tercera lista
[6, 7, 8, 9]
>>>
```

Capítulo 85.- Listas: ordenamiento de sus elementos – 1

Otro algoritmo muy común que debe conocer y entender un programador es el ordenamiento de una lista de datos.

El ordenamiento de una lista se logra intercambiando los componentes de manera que:

lista[0] <= lista[1] <= lista[2] etc.

El contenido de la componente lista[0] sea menor o igual al contenido de la componente lista[1] y así sucesivamente.

Si se cumple lo dicho anteriormente decimos que la lista está ordenada de menor a mayor. Igualmente podemos ordenar la lista de mayor a menor.

Tengamos en cuenta que la estructura de datos lista en Python es mutable, eso significa que podemos modificar su elementos por otros.

Se puede ordenar tanto listas con componentes de tipo int, float como cadena de caracteres. En este último caso el ordenamiento es alfabético.

Problema:

se debe crear y cargar una lista donde almacenar 5 sueldos. Desplazar el valor mayor de la lista a la última posición.

La primera aproximación para llegar en el próximo problema al ordenamiento completo de una lista tiene por objetivo analizar los intercambios de elementos dentro de la lista y dejar el mayor en la última posición.

El algoritmo consiste en comparar si el primer componente es mayor al segundo, en caso que la condición sea verdadera, intercambiamos los contenidos de los componentes.

Vamos a suponer que se ingresaron los siguientes valores por teclado:

1200
750
820
550
490

```
Cuando k = 0
                x = 0          x = 1          x = 2          x = 3
                750            750            750            750
                1200           820            820            820
                820            1200           550            550
                550            550            1200           490
                490            490            490            1200

Cuando k = 1
                x = 0          x = 1          x = 2          x = 3
                750            750            750            750
                820            550            550            550
                550            820            490            490
                490            490            820            820
                1200           1200           1200           1200
```

```
Cuando k = 2
                x = 0          x = 1          x  = 2         x = 3
                550            550            550            550
                750            490            490            490
                490            750            750            750
                820            820            820            820
                1200           1200           1200           1200


Cuando k = 3
                x = 0          x = 1          x  = 2         x = 3
                490            490            490            490
                550            550            550            550
                750            750            750            750
                820            820            820            820
                1200           1200           1200           1200
```

Vamos a programar:

ejercicio85.py - D:/Curso completo de Python/ejercicio85.py (3.11.1)

File Edit Format Run Options Window Help

```python
sueldos=[]
for x in range(5):
    valor=int(input("Ingrese sueldo:"))
    sueldos.append(valor)
print("Lista sin ordenar")
print(sueldos)
for k in range(4):
    for x in range(4):
        if sueldos[x]>sueldos[x+1]:
            aux=sueldos[x]
            sueldos[x]=sueldos[x+1]
            sueldos[x+1]=aux
print("Lista ordenada")
print(sueldos)
```

Guardamos el programa con el nombre de Ejercicio85.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio85.py =============
Ingrese sueldo:1200
Ingrese sueldo:750
Ingrese sueldo:820
Ingrese sueldo:550
Ingrese sueldo:490
Lista sin ordenar
[1200, 750, 820, 550, 490]
Lista ordenada
[490, 550, 750, 820, 1200]
>>>
```

Capítulo 86.- Listas: ordenamiento de sus elementos – 2

Una última consideración a este ALGORITMO de ordenamiento es que los elementos que se van ordenando continuamos comparándolos.

Ejemplo: En la primera ejecución del for interno el valor 1200 queda ubicado en la posición 4 de la lista. En la segunda ejecución comparamos si el 820 es mayor a 1200, lo cual seguramente será falso.

Podemos concluir que la primera vez debemos hacer para este ejemplo 4 comparaciones, en la segunda ejecución del for interno debemos hacer 3 comparaciones y en general debemos ir reduciendo en uno la cantidad de comparaciones.

Si bien el algoritmo planteado funciona, un algoritmo más eficiente, que se deriva del anterior es el plantear un for interno con la siguiente estructura:

ejercicio85.py - D:/Curso completo de Python/ejercicio85.py (3.11.1)

File Edit Format Run Options Window Help

```
sueldos=[]
for x in range(5):
    valor=int(input("Ingrese sueldo:"))
    sueldos.append(valor)
print("Lista sin ordenar")
print(sueldos)
for k in range(4):
    for x in range(4-k):
        if sueldos[x]>sueldos[x+1]:
            aux=sueldos[x]
            sueldos[x]=sueldos[x+1]
            sueldos[x+1]=aux
print("Lista ordenada")
print(sueldos)
```

Vamos a ejecutar de nuevo:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio85.py =============
Ingrese sueldo:1200
Ingrese sueldo:750
Ingrese sueldo:820
Ingrese sueldo:550
Ingrese sueldo:490
Lista sin ordenar
[1200, 750, 820, 550, 490]
Lista ordenada
[490, 550, 750, 820, 1200]
>>>
```

Este método se llama el método de la burbuja.

Capítulo 87.- Listas: ordenamiento de sus elementos – 3

Problema propuesto:

Crea una lista y almacenar los nombres de 5 países. Ordenar alfabéticamente la lista e imprimirla.

Vamos a programar:

ejercicio87.py - D:/Curso completo de Python/ejercicio87.py (3.11.1)

File Edit Format Run Options Window Help

```
paises=[]
for x in range(5):
    pais=input("Ingrese un país:  ")
    paises.append(pais)
print(paises)
for k in range(4):
    for j in range(4-k):
        if paises[j]>paises[j+1]:
            aux=paises[j]
            paises[j]=paises[j+1]
            paises[j+1]=aux
print("Lista ordenada alfabéticamente")
print(paises)
```

Guardamos el programa con el nombre de Ejercicio87.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio87.py =============
Ingrese un país:  Uruguay
Ingrese un país:  Chile
Ingrese un país:  Argentina
Ingrese un país:  Brasil
Ingrese un país:  Colombia
['Uruguay', 'Chile', 'Argentina', 'Brasil', 'Colombia']
Lista ordenada alfabéticamente
['Argentina', 'Brasil', 'Chile', 'Colombia', 'Uruguay']
>>>
```

Capítulo 88.- Listas: ordenamiento de sus elementos – 4

Problema propuesto:

Solicitar por teclado la cantidad de empleados que tiene su empresa. Crear y cargar una lista con todos los sueldos de dichos empleados. Imprimir la lista de sueldos ordenados de menor a mayor.

Vamos a programar:

ejercicio88.py - D:/Curso completo de Python/ejercicio88.py (3.11.1)

File Edit Format Run Options Window Help

```
sueldos=[]
empleados=int(input("Cuantos empleados tiene su empresa:"))
for x in range(empleados):
    sueldo=int(input("Ingrese el sueldo: "))
    sueldos.append(sueldo)
k=empleados-1
print("Lista sin ordenar")
print(sueldos)
for x in range(k):
    for y in range(k-x):
        if sueldos[y]>sueldos[y+1]:
            aux=sueldos[y]
            sueldos[y]=sueldos[y+1]
            sueldos[y+1]=aux
print("Lista ordenada")
print(sueldos)
```

Guardar el programa con el nombre de Ejercicio88.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio88.py =============
Cuantos empleados tiene su empresa:4
Ingrese el sueldo: 1000
Ingrese el sueldo: 700
Ingrese el sueldo: 900
Ingrese el sueldo: 200
Lista sin ordenar
[1000, 700, 900, 200]
Lista ordenada
[200, 700, 900, 1000]
>>>
```

Ejecutamos de nuevo:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio88.py =============
Cuantos empleados tiene su empresa:6
Ingrese el sueldo: 6000
Ingrese el sueldo: 4000
Ingrese el sueldo: 5000
```

```
Ingrese el sueldo: 5550
Ingrese el sueldo: 2000
Ingrese el sueldo: 1600
Lista sin ordenar
[6000, 4000, 5000, 5550, 2000, 1600]
Lista ordenada
[1600, 2000, 4000, 5000, 5550, 6000]
>>>
```

Capítulo 89.- Listas: ordenamiento de sus elementos – 5

Problema propuesto:

Cargar una lista con 5 elementos enteros. Ordenarla de menor a mayor y mostrarla por pantalla, luego ordenarla de mayor a menor e imprimir nuevamente.

Vamos a programar:

ejercicio89.py - D:/Curso completo de Python/ejercicio89.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[]
for x in range(5):
    valor=int(input("Ingrese un valor: "))
    lista.append(valor)
print("Mostrar lista sin ordenar")
print(lista)
for x in range(4):
    for y in range(4-x):
        if lista[y]>lista[y+1]:
            aux=lista[y+1]
            lista[y+1]=lista[y]
            lista[y]=aux
print("Mostrar lista ordenada en modo creciente")
print(lista)
for x in range(4):
    for y in range(4-x):
        if lista[y]<lista[y+1]:
            aux=lista[y+1]
            lista[y+1]=lista[y]
            lista[y]=aux
print("Mostrar lista ordenada en modo decreciente")
print(lista)
```

Guardamos el programa con el nombre de Ejercicio89.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio89.py ==============
Ingrese un valor: 100
Ingrese un valor: 50
Ingrese un valor: 200
Ingrese un valor: 7
Ingrese un valor: 20
Mostrar lista sin ordenar
[100, 50, 200, 7, 20]
Mostrar lista ordenada en modo creciente
[7, 20, 50, 100, 200]
Mostrar lista ordenada en modo decreciente
[200, 100, 50, 20, 7]
>>>
```

Capítulo 90.- Listas: ordenamiento con listas paralelas – 1

Cuando se tienen listas paralelas y se ordenan los elementos de una de ellas hay que tener la precaución de intercambiar los elementos de las listas paralelas.

Problema:

Confeccionar un programa que permita cargar los nombres de 5 alumnos y sus notas respectivas.
Luego ordena las notas de mayor a menor. Imprimir las notas y los nombres de los alumnos.
Debe quedar claro que cuando intercambiemos las notas para dejarlas ordenadas de mayor a menor debemos intercambiar los nombres para que las listas continúen paralelas. (es decir que en los mismos subíndices de cada lista continúe la información relacionada).

Vamos a programar:

ejercicio90.py - D:/Curso completo de Python/ejercicio90.py (3.11.1)

File Edit Format Run Options Window Help

```
alumnos=[]
notas=[]
for x in range(5):
    alumno=input("Ingrese el nombre del alumno: ")
    alumnos.append(alumno)
    nota=int(input("Ingrese la nota del alumno: "))
    notas.append(nota)
print("Mostrar las dos listas")
print(alumnos)
print(notas)
for x in range(4):
    for y in range(4-x):
        if notas[y]<notas[y+1]:
            aux=notas[y+1]
            notas[y+1]=notas[y]
            notas[y]=aux
            aux=alumnos[y+1]
            alumnos[y+1]=alumnos[y]
            alumnos[y]=aux
print("Mostrar listas ordenadas por notas en modo decreciente")
print(alumnos)
print(notas)
```

Guardamos el programa con el nombre de Ejercicio90.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio90.py =============
Ingrese el nombre del alumno: Luis
Ingrese la nota del alumno: 7
Ingrese el nombre del alumno: Carlos
Ingrese la nota del alumno: 4
Ingrese el nombre del alumno: Ana
Ingrese la nota del alumno: 10
Ingrese el nombre del alumno: Mauricio
Ingrese la nota del alumno: 8
Ingrese el nombre del alumno: Esteban
Ingrese la nota del alumno: 5
```

```
Mostrar las dos listas
['Luis', 'Carlos', 'Ana', 'Mauricio', 'Esteban']
[7, 4, 10, 8, 5]
Mostrar listas ordenadas por notas en modo decreciente
['Ana', 'Mauricio', 'Luis', 'Esteban', 'Carlos']
[10, 8, 7, 5, 4]
>>>
```

Vamos a modifica el código:

ejercicio90.py - D:/Curso completo de Python/ejercicio90.py (3.11.1)

File Edit Format Run Options Window Help

```
alumnos=[]
notas=[]
for x in range(5):
    alumno=input("Ingrese el nombre del alumno: ")
    alumnos.append(alumno)
    nota=int(input("Ingrese la nota del alumno: "))
    notas.append(nota)
print("Mostrar las dos listas")
print(alumnos)
print(notas)
for x in range(4):
    for y in range(4-x):
        if notas[y]<notas[y+1]:
            aux=notas[y+1]
            notas[y+1]=notas[y]
            notas[y]=aux
            aux=alumnos[y+1]
            alumnos[y+1]=alumnos[y]
            alumnos[y]=aux
print("Mostrar listas ordenadas por notas en modo decreciente")
for x in range(5):
    print(alumnos[x], notas[x])
```

Ejecutamos de nuevo:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio90.py =============
Ingrese el nombre del alumno: Luis
Ingrese la nota del alumno: 7
Ingrese el nombre del alumno: Carlos
Ingrese la nota del alumno: 4
Ingrese el nombre del alumno: Ana
Ingrese la nota del alumno: 10
Ingrese el nombre del alumno: Mauricio
Ingrese la nota del alumno: 8
Ingrese el nombre del alumno: Esteban
Ingrese la nota del alumno: 5
Mostrar las dos listas
['Luis', 'Carlos', 'Ana', 'Mauricio', 'Esteban']
[7, 4, 10, 8, 5]
Mostrar listas ordenadas por notas en modo decreciente
Ana 10
Mauricio 8
Luis 7
Esteban 5
Carlos 4
>>>
```

Capítulo 91.- Listas: ordenamiento con listas paralelas – 2

Problema propuesto:

Crear y cargar una lista de nombres de 5 países y en otra lista paralela la cantidad de habitantes del mismo. Ordenar alfabéticamente e imprimir los resultados. Por último ordenar con respecto a la cantidad de habitantes (de mayor a menor) e imprimir nuevamente.

Vamos a programar:

ejercicio91.py - D:/Curso completo de Python/ejercicio91.py (3.11.1)

File Edit Format Run Options Window Help

```
paises=[]
habitantes=[]
for x in range(5):
    pais=input("Ingrese un pais: ")
    paises.append(pais)
    habitante=int(input("Ingrese el número de habitantes: "))
    habitantes.append(habitante)
for y in range(4):
    for z in range(4-y):
        if paises[z]>paises[z+1]:
            aux=paises[z+1]
            paises[z+1]=paises[z]
            paises[z]=aux
            aux=habitantes[z+1]
            habitantes[z+1]=habitantes[z]
            habitantes[z]=aux
print("Listado de paises en orden alfabético")
for x in range(5):
    print(paises[x], habitantes[x])
for y in range(4):
    for z in range(4-y):
         if habitantes[z]<habitantes[z+1]:
            aux=paises[z+1]
            paises[z+1]=paises[z]
            paises[z]=aux
            aux=habitantes[z+1]
            habitantes[z+1]=habitantes[z]
            habitantes[z]=aux
print("Listado de paises por cantidad de habitantes")
for x in range(5):
    print(paises[x], habitantes[x])
```

Guardamos el programa con el nombre de Ejercicio91.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio91.py =============
Ingrese un pais: Argentina
Ingrese el número de habitantes: 40000000
Ingrese un pais: Chile
Ingrese el número de habitantes: 20000000
Ingrese un pais: Brasil
Ingrese el número de habitantes: 180000000
Ingrese un pais: Uruguay
Ingrese el número de habitantes: 3000000
Ingrese un pais: Colombia
Ingrese el número de habitantes: 46000000
```

```
    Listado de paises en orden alfabético
    Argentina 40000000
    Brasil 180000000
    Chile 20000000
    Colombia 46000000
    Uruguay 3000000
    Listado de paises por cantidad de habitantes
    Brasil 180000000
    Colombia 46000000
    Argentina 40000000
    Chile 20000000
    Uruguay 3000000
>>>
```

Capítulo 92.- Listas: componentes de tipo lista – 1

Hasta ahora hemos trabajado con listas cuyos componentes so de tipos:

enteros
flotantes
cadena de caracteres

Ejemplo

notas=[8, 6, 8]
alturas=[1.73, 1.55, 1.92]
días=["lunes", "martes", "miércoles"]

Pero lo que la hace tan flexible a esta estructura de datos es que podemos almacenar componentes de tipo LISTA.

notas=[[5,5], [6,9], [7,3]]

En la línea anterior hemos definido una lista de tres elementos de tipo lista, el primer elemento de la lista es otra lista de dos elementos de tipo entero. De forma similar los otros dos elementos de la lista notas son listas de dos elementos de tipo entero.

Problema:

Crear una lista por asignación. La lista tiene que tener cuatro elementos. Cada elemento debe ser una lista de 3 enteros.

Imprimir sus elementos accediendo de diferentes modos.

Vamos a programar:

ejercicio92.py - D:/Curso completo de Python/ejercicio92.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[[1,2,3], [4,5,6], [7,8,9], [10,11,12]]
print(lista)
# Imprimir el primer componente
print(lista[0])
# Imprimir el primer elemento del primer componente
print(lista[0][0])
print("---------")
for x in range(len(lista[0])):
      print(lista[0][x])
print("---------")
for k in range(len(lista)):
    for x in range(len(lista[k])):
        print(lista[k][x])
```

Guardamos el programa con el nombre de Ejercicio92.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio92.py =============
[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]
[1, 2, 3]
```

```
    1
    ---------
    1
    2
    3
    ---------
    1
    2
    3
    4
    5
    6
    7
    8
    9
    10
    11
    12
>>>
```

Capítulo 93.- Listas: componentes de tipo lista – 2

Problema:

Crear una lista por asignación. La lista tiene que tener 2 elementos. Cada elemento debe ser una lista de 5 enteros.

Calcular y mostrar la suma de cada lista contenida en la lista principal.

lista=[[1,1,1,1,1], [2,2,2,2,2]]

Vamos a programar:

ejercicio93.py - D:/Curso completo de Python/ejercicio93.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[[1,1,1,1,1], [2,2,2,2,2]]
# Método 1
print("Primer método")
suma1=lista[0][0]+lista[0][1]+lista[0][2]+lista[0][3]+lista[0][4]
suma2=lista[1][0]+lista[1][1]+lista[1][2]+lista[1][3]+lista[1][4]
print("La suma del primer elemento")
print(suma1)
print("La sema del segundo elemento")
print(suma2)
# Método 2
print("Segundo método")
suma1=0
suma2=0
for x in range(len(lista[0])):
    suma1=suma1+lista[0][x]
    suma2=suma2+lista[1][x]
print("La suma del primer elemento")
print(suma1)
print("La sema del segundo elemento")
print(suma2)
# Método 3
print("Tercer método")
for k in range(len(lista)):
    suma=0
    for x in range(len(lista[0])):
        suma=suma+lista[k][x]
    print(suma)
```

El programa lo guardamos como Ejercicio93.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio93.py =============
Primer método
La suma del primer elemento
5
La sema del segundo elemento
10
Segundo método
La suma del primer elemento
5
La sema del segundo elemento
10
Tercer método
5
10
>>>
```

Capítulo 94.- Listas: componentes de tipo lista – 3

Problema:

Crear una lista por asignación. La lista tiene que tener 5 elementos. Cada elemento debe ser una lista, la primera lista tiene que tener un elemento, la segunda dos elementos, la tercera tres elementos y así sucesivamente.

Sumar todos los valores de las listas.

Vamos a programar:

ejercicio94.py - D:/Curso completo de Python/ejercicio94.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[[1], [1,2], [1,2,3], [1,2,3,4], [1,2,3,4,5]]
sumatotal=0
for x in range(len(lista)):
    suma=0
    for k in range(len(lista[x])):
        suma=suma+lista[x][k]
        sumatotal=sumatotal+lista[x][k]
    print("La suma por lista es")
    print(suma)
print("La suma total es")
print(sumatotal)
```

Guardamos el programa con el nombre de Ejercicio94.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio94.py =============
La suma por lista es
1
La suma por lista es
3
La suma por lista es
6
La suma por lista es
10
La suma por lista es
15
La suma total es
35
>>>
```

Capítulo 95.- Listas: componentes de tipo lista – 4

Problema propuesto:

Se tiene la siguiente lista:

lista=[[100, 7, 85, 8], [4,8, 56,25], [67,89, 23,1], [78,56]]

Imprimir la lista. Luego fijar con el valor cero todos los elementos mayores a 50 del primer elemento de "lista".

Volver a imprimir la lista.

Vamos a programar:

ejercicio95.py - D:/Curso completo de Python/ejercicio95.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[[100,7,85,8], [4,8,56,25], [67,89,23,1], [78,56]]
print(lista)
for x in range(len(lista[0])):
    if lista[0][x]>50:
        lista[0][x]=0
print(lista)
```

Guardamos el programa con el nombre de Ejercicio95.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio95.py =============
[[100, 7, 85, 8], [4, 8, 56, 25], [67, 89, 23, 1], [78, 56]]
[[0, 7, 0, 8], [4, 8, 56, 25], [67, 89, 23, 1], [78, 56]]
>>>
```

Capítulo 96.- Listas: componentes de tipo lista – 5

Problema propuesto:

Se tiene la siguiente lista:

lista=[[4,12,5,66], [14,6,25], [3, 4, 5, 67,89,23,1], [78,56]]

Imprime la lista, Luego fijar con el valor cero a todos los elementos mayores a 10 contenidos en todos los elementos de la variable "lista".

Volver a imprimir la lista.

Vamos a programar:

```
*ejercicio96.py - D:/Curso completo de Python/ejercicio96.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 lista=[[4,12,5,66], [14,6,25], [3,4,5,67,89,23,1], [78,56]]
2 print(lista)
3 for x in range(len(lista)):
4     for y in range(len(lista[x])):
5         if lista[x][y]>10:
6             lista[x][y]=0
7 print(lista)
8
9
```

Guardamos el programa con el nombre de Ejercicio96.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio96.py =============
[[4, 12, 5, 66], [14, 6, 25], [3, 4, 5, 67, 89, 23, 1], [78, 56]]
[[4, 0, 5, 0], [0, 6, 0], [3, 4, 5, 0, 0, 0, 1], [0, 0]]
>>>
```

Capítulo 97.- Listas: componentes de tipo lista – 6

Problema propuesto:

Crear una lista por asignación con la cantidad de elementos de tipo lista que usted desee. Luego imprimir el último elemento de la lista principal.

Vamos a programar:

```
ejercicio97.py - D:/Curso completo de Python/ejercicio97.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 lista=[["Juan", "Ana"], ["Luis"], ["Pedro", "María", "Carlos"]]
2 print(lista)
3 print(lista[len(lista)-1])
4
```

Guardamos el programa con el nombre de Ejercicio97.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio97.py =============
    [['Juan', 'Ana'], ['Luis'], ['Pedro', 'María', 'Carlos']]
    ['Pedro', 'María', 'Carlos']
>>>
```

Si queremos imprimir el primer elemento de la lista:

```
ejercicio97.py - D:/Curso completo de Python/ejercicio97.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 lista=[["Juan", "Ana"], ["Luis"], ["Pedro", "María", "Carlos"]]
2 print(lista)
3 print(lista[len(lista)-1])
4 print(lista[0])
5
```

Ejecutamos de nuevo:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio97.py =============
    [['Juan', 'Ana'], ['Luis'], ['Pedro', 'María', 'Carlos']]
    ['Pedro', 'María', 'Carlos']
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio97.py =============
    [['Juan', 'Ana'], ['Luis'], ['Pedro', 'María', 'Carlos']]
    ['Pedro', 'María', 'Carlos']
    ['Juan', 'Ana']  <----
>>>
```

Capítulo 98.- Listas: carga por teclado de componentes de tipo lista – 1

En el concepto anterior vimos que fácilmente te podemos definir por asignación una lista cuyos componentes sean también listas:

lista=[[1,2,3], [7,4], [9,2]]

En muchas situaciones debemos crear una nueva lista ingresando datos por teclado.

Problema

Crea y carga una lista con los nombres de tres alumnos. Cada alumno tiene dos notas, almacenar las notas en una lista paralela. Cada componente de la lista paralela debe ser también una lista con las dos notas. Imprimir luego cada nombre y sus dos notas.

Si cargamos los datos por asignación sería algo parecido a esto:

alumnos=["Juan", "Ana", "Luis"]

notas=[[10,8], [6,5], [2,8]]

En el componente 0 de la lista alumnos tenemos almacenado el nombre "Juan" y como son listas paralelas en el componente 0 de la lista notas tenemos almacenado una lista con las notas de 10 y 8.

Nuestro objetivo como pide el problema es cargar los datos por teclado.

Vamos a programar:

ejercicio98.py - D:/Curso completo de Python/ejercicio98.py (3.11.1)

File Edit Format Run Options Window Help

```
# Primer método
print("Primer método")
alumnos=[]
notas=[]
for x in range(3):
    alumno=input("Ingrese el nombre del alumno: ")
    alumnos.append(alumno)
    nota1=int(input("Ingrese la primera nota: "))
    nota2=int(input("Ingrese la segunda nota: "))
    notas.append([nota1, nota1])
print(alumnos)
print(notas)
# Segundo método
print("Segundo método")
alumnos=[]
notas=[]
nota=[0,0]
for x in range(3):
    alumno=input("Ingrese el nombre del alumno: ")
    alumnos.append(alumno)
    for y in range(2):
        nota[y]=int(input("Ingrese su nota: "))
    notas.append([nota[0], nota[1]])
print(alumnos)
print(notas)
```

Guardamos el programa con el nombre de Ejercicio98.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio98.py =============
Primer método
Ingrese el nombre del alumno: Juan
Ingrese la primera nota: 10
Ingrese la segunda nota: 8
Ingrese el nombre del alumno: Ana
Ingrese la primera nota: 6
Ingrese la segunda nota: 5
Ingrese el nombre del alumno: Luis
Ingrese la primera nota: 2
Ingrese la segunda nota: 8
['Juan', 'Ana', 'Luis']
[[10, 10], [6, 6], [2, 2]]
Segundo método
Ingrese el nombre del alumno: Juan
Ingrese su nota: 10
Ingrese su nota: 8
Ingrese el nombre del alumno: Ana
Ingrese su nota: 6
Ingrese su nota: 5
Ingrese el nombre del alumno: Luis
Ingrese su nota: 2
Ingrese su nota: 8
['Juan', 'Ana', 'Luis']
[[10, 8], [6, 5], [2, 8]]
>>>
```

Vamos a mostrar el listado de otra forma:

```
*ejercicio98.py - D:/Curso completo de Python/ejercicio98.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1  # Primer método
2  print("Primer método")
3  alumnos=[]
4  notas=[]
5  for x in range(3):
6      alumno=input("Ingrese el nombre del alumno: ")
7      alumnos.append(alumno)
8      nota1=int(input("Ingrese la primera nota: "))
9      nota2=int(input("Ingrese la segunda nota: "))
10     notas.append([nota1, nota1])
11 print(alumnos)
12 print(notas)
13 # Segundo método
14 print("Segundo método")
15 alumnos=[]
16 notas=[]
17 nota=[0,0]
18 for x in range(3):
19     alumno=input("Ingrese el nombre del alumno: ")
20     alumnos.append(alumno)
21     for y in range(2):
22         nota[y]=int(input("Ingrese su nota: "))
23     notas.append([nota[0], nota[1]])
24 for x in range(3):
25     print(alumnos[x], notas[x][0], notas[x][1])
26
```

Vamos a ejecutar de nuevo:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: D:/Curso completo de Python/ejercicio98.py =============
Primer método
Ingrese el nombre del alumno: Juan
Ingrese la primera nota: 10
Ingrese la segunda nota: 8
Ingrese el nombre del alumno: Ana
Ingrese la primera nota: 6
Ingrese la segunda nota: 5
Ingrese el nombre del alumno: Luis
Ingrese la primera nota: 2
Ingrese la segunda nota: 8
['Juan', 'Ana', 'Luis']
[[10, 10], [6, 6], [2, 2]]
Segundo método
Ingrese el nombre del alumno: Juan
Ingrese su nota: 10
Ingrese su nota: 8
Ingrese el nombre del alumno: Ana
Ingrese su nota: 5
Ingrese su nota: 6
Ingrese el nombre del alumno: Luis
Ingrese su nota: 2
Ingrese su nota: 8
Juan 10 8
Ana 5 6
Luis 2 8
>>>
```

Capítulo 99.- Listas: carga por teclado de componentes de tipo lista – 2

Problema:

Se tiene que cargar la siguiente información:

- Nombre de 3 empleados.
- Ingresos en concepto de sueldo, cobrado por cada empleado, en los últimos 3 meses.

Confeccionar el programa para:

a) Realizar la carga de los nombres de empleados y los tres sueldos por cada empleado.
b) Generar una lista que contenga el ingreso acumulado en sueldo de los últimos 3 meses para cada empleado.
c) Mostrar por pantalla el total pagado en sueldos a cada empleado en los últimos 3 meses.
d) Obtener el nombre del empleado que tuvo el mayor ingreso acumulado.

En este problema definiremos tres listas:

Una lista para almacenar el nombre de los empleados, por ejemplo si lo cargamos por asignación:

nombre=["Juan", "Ana", "Luis"]

Una lista paralela a la anterior para almacenar los sueldos en los últimos tres meses por cada empleado:

sueldo=[[5000, 5100, 5000], [7000, 6500, 6000], [2500, 2500, 2500]]

Otra lista paralela donde almacenaremos el total de sueldos cobrados por cada empleado que resulta de sumar los tres sueldos de cada empleado contenidos en la lista sueldos:

totalsueldos=[15100, 1950, 7500]

Es importante notar que estos datos no los cargaremos por asignación sino los cargaremos por teclado a las dos primeras listas y la tercera la generaremos extrayendo los datos de la lista sueldos.

Vamos a programar:

ejercicio99.py - D:/Curso completo de Python/ejercicio99.py (3.11.1)

File Edit Format Run Options Window Help

```
# Definir y asignar valores por teclado de
# las listas nombres y suledos
nombres=[]
sueldos=[]
for x in range(3):
    nombre=input("Ingrese el nombre del empleado: ")
    nombres.append(nombre)
    sueldo1=int(input("Ingrese el primer suledo: "))
    sueldo2=int(input("Ingrese el segundo sueldo: "))
    sueldo3=int(input("Ingrese el tercer sueldo: "))
    sueldos.append([sueldo1, sueldo2, sueldo3])
print("Mostramos el contenido de las listas nombres y sueldos")
print(nombres)
print(sueldos)
```

```
15 # Definimos la lista totalsueldos y asingamos valores
16 totalsueldos=[]
17 for x in range(3):
18     total=0
19     for y in range(3):
20         total=total+sueldos[x][y]
21     totalsueldos.append(total)
22 print("Mostrar el contenido de la lista totalsueldos")
23 print(totalsueldos)
24 # listar el sueldo de cada empleado
25 print("----------")
26 for x in range(3):
27     print(nombres[x], totalsueldos[x])
28 print("---------")
29 # Calcular el valor mayor de la lista totalsueldos
30 valormax=totalsueldos[0]
31 for x in range(1,3):
32     if totalsueldos[x]>valormax:
33         valormax=totalsueldos[x]
34 print("El sueldo acumulado mayor es de")
35 print(valormax)
36 # Buscar el empleado con mayor sueldo
37 for x in range(3):
38     if totalsueldos[x]==valormax:
39         print("El empleado de mayor sueldo es")
40         print(nombres[x])
41
```

Guardamos el programa con el nombre de Ejercicio99.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: D:/Curso completo de Python/ejercicio99.py =============
    Ingrese el nombre del empleado: Juan
    Ingrese el primer suledo: 5000
    Ingrese el segundo sueldo: 5100
    Ingrese el tercer sueldo: 5000
    Ingrese el nombre del empleado: Ana
    Ingrese el primer suledo: 7000
    Ingrese el segundo sueldo: 6500
    Ingrese el tercer sueldo: 6000
    Ingrese el nombre del empleado: Luis
    Ingrese el primer suledo: 2500
    Ingrese el segundo sueldo: 2500
    Ingrese el tercer sueldo: 2500
    Mostramos el contenido de las listas nombres y sueldos
    ['Juan', 'Ana', 'Luis']
    [[5000, 5100, 5000], [7000, 6500, 6000], [2500, 2500, 2500]]
    Mostrar el contenido de la lista totalsueldos
    [15100, 19500, 7500]
    ----------
    Juan 15100
    Ana 19500
    Luis 7500
    ---------
    El sueldo acumulado mayor es de
    19500
    El empleado de mayor sueldo es
    Ana
>>>
```

Capítulo 100.- Listas: carga por teclado de componentes de tipo lista – 3

Problema:

Solicitar por teclado dos enteros. El primer valor indica la cantidad de elementos que crearemos en la lista. El segundo valor indica la cantidad de elementos que tendrá cada una de las listas internas a la lista principal.

Mostrar la lista y la suma de todos sus elementos.

Por ejemplo si el operador carga un 2 y un 4 significa que debemos crear una lista similar a: lista=[[1,1,1,1], [1,1,1,1]]. Vamos a programar.

ejercicio100.py - D:/Curso completo de Python/ejercicio100.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[]
suma=0
elementos=int(input("Ingrese la cantida de elementos: "))
elementoslista=int(input("Ingrese la cantidad de elementos de la lista: "))
for x in range(elementos):
    lista.append([])
    print("Lista", x+1)
    for y in range(elementoslista):
        num=int(input("Ingrese un número: "))
        lista[x].append(num)
print("Lista")
print("-----")
print(lista)
# Sumar todos los elementos
for x in range(elementos):
    for y in range(elementoslista):
        suma=suma+lista[x][y]
print("La suma de todos los elementos es")
print(suma)
```

Guardamos el programa con el nombre de Ejercicio100.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio100.py =============
Ingrese la cantida de elementos: 3
Ingrese la cantidad de elementos de la lista: 4
Lista 1
Ingrese un número: 4
Ingrese un número: 7
Ingrese un número: 2
Ingrese un número: 9
Lista 2
Ingrese un número: 2
Ingrese un número: 9
Ingrese un número: 4
Ingrese un número: 6
Lista 3
Ingrese un número: 7
Ingrese un número: 1
Ingrese un número: 9
Ingrese un número: 3
Lista
-----
[[4, 7, 2, 9], [2, 9, 4, 6], [7, 1, 9, 3]]
La suma de todos los elementos es
63
```

Capítulo 101.- Listas: carga por teclado de componentes de tipo lista – 4

Problema:

Definir dos listas de 3 elementos.

La primera lista cada elemento es una sublista con el nombre del padre y la madre de una familia.

La segunda lista está constituida por listas con los nombres de los hijos de cada familia. Puede haber familias si hijos.

Imprimir los nombres del padre, la madre y sus hijos.

También imprimir solo el nombre del padre y la cantidad de hijos que tiene dicho padre.

Un ejemplo si se define por asignación:

padres=[["Juan", "Ana"], ["Carlos", "María"], ["Pedro", "Laura"]]

hijos=[["Marcos", "Alberto", "Silvia"], [], ["Oscar"]]

Como son listas paralelas podemos decir que la familia cuyos padres son "Juan" y "Ana" tienen tres hijos llamados "Marcos", "Alberto", "Silvia". La segunda familia no tiene hijos y la tercera tiene solo uno.

Vamos a programar:

ejercicio101.py - D:/Curso completo de Python/ejercicio101.py (3.11.1)

File Edit Format Run Options Window Help

```
1  padres=[]
2  hijos=[]
3  familias=int(input("Cuantas familias: "))
4  for x in range(familias):
5      padre=input("Nombre del padre: ")
6      madre=input("nombre de la madre: ")
7      padres.append([padre, madre])
8      numhijos=int(input("Cuantos hijos: "))
9      hijos.append([])
10     for y in range(numhijos):
11         hijo=input("Nombre del hijo: ")
12         hijos[x].append(hijo)
13 # Listado del padre la madre y los hijos
14 print("Listado de Padre, Madre e hijos")
15 for x in range(familias):
16     print("Padre:", padres[x][0])
17     print("Madre:", padres[x][1])
18     for y in range(len(hijos[x])):
19         print("hijo: ", hijos[x][y])
20     print("-----------------")
21 # listado de Padres y la cantidad de hijos
22 print("Listado del Padre y número de hijos")
23 for x in range(familias):
24     print("Padre:", padres[x][0])
25     print("Número de hijos: ", len(hijos[x]))
26
```

Guardamos el programa con el nombre de Ejercicio101.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio101.py =============
Cuantas familias: 3
Nombre del padre: Juan
nombre de la madre: Ana
Cuantos hijos: 3
Nombre del hijo: Marcos
Nombre del hijo: Alberto
Nombre del hijo: Silvia
Nombre del padre: Carlos
nombre de la madre: María
Cuantos hijos: 0
Nombre del padre: Pedro
nombre de la madre: Laura
Cuantos hijos: 1
Nombre del hijo: Oscar
Listado de Padre, Madre e hijos
Padre: Juan
Madre: Ana
hijo:  Marcos
hijo:  Alberto
hijo:  Silvia
-----------------
Padre: Carlos
Madre: María
-----------------
Padre: Pedro
Madre: Laura
hijo:  Oscar
-----------------
Listado del Padre y número de hijos
Padre: Juan
Número de hijos:  3
Padre: Carlos
Número de hijos:  0
Padre: Pedro
Número de hijos:  1
>>>
Ln: 40  Col: 0
```

Capítulo 102.- Listas: carga por teclado de componentes de tipo lista – 5

Problema propuesto:

Se desea saber la temperatura media trimestral de cuatro países. Para ello se tiene como dato las temperaturas medias mensuales de dichos países.

Se debe ingresar el nombre del país y seguidamente las tres temperaturas medias mensuales.

Seleccionar las estructuras de datos adecuados para el almacenamiento de los datos en memoria.

a) Cargar por teclado los nombres de los países y las temperaturas medias mensuales.
b) Imprimir los nombres de los países y las temperaturas medias mensuales.
c) Calcular la temperatura media trimestral de cada país.
d) Imprimir los nombres de los países y las temperaturas medias trimestrales.
e) Imprimir el nombre del país con la temperatura media trimestral mayor.

Vamos a programar:

ejercicio102.py - D:/Curso completo de Python/ejercicio102.py (3.11.1)

File Edit Format Run Options Window Help

```
1  paises=[]
2  temperaturas=[]
3  temperaturasmedias=[]
4  for x in range(4):
5      pais=input("Ingrese el país: ")
6      paises.append(pais)
7      temperaturas.append([])
8      for y in range(3):
9          temperatura=float(input("Ingrese la temperatura media por mes: "))
10         temperaturas[x].append(temperatura)
11 print("Nombres de los paises y temperaturas medias mensuales")
12 for x in range(3):
13     print(paises[x], temperaturas[x][0], temperaturas[x][1], temperaturas[x][2])
14     suma=temperaturas[x][0]+temperaturas[x][1]+temperaturas[x][2]
15     media=suma/3
16     temperaturasmedias.append(media)
17 print("Temperaturas medias por pais")
18 for x in range(3):
19     print(paises[x], temperaturasmedias[x])
20 tempmax=temperaturasmedias[0]
21 # Mostrar el pais con la temperatura media más alta
22 for x in range(1, 3):
23     if temperaturasmedias[x]>tempmax:
24         tempmax=temperaturasmedias[x]
25 print("País con temperaturas medias más altas")
26 for x in range(3):
27     if temperaturasmedias[x]==tempmax:
28         print("Pais con temperatura media más alta ", paises[x])
29
```

Guardamos el programa con el nombre de Ejercicio102.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio102.py =============
Ingrese el país: Chile
Ingrese la temperatura media por mes: 22
Ingrese la temperatura media por mes: 22
Ingrese la temperatura media por mes: 22
Ingrese el país: Argentina
Ingrese la temperatura media por mes: 2
Ingrese la temperatura media por mes: 3
Ingrese la temperatura media por mes: 3
Ingrese el país: Brasil
Ingrese la temperatura media por mes: 5
Ingrese la temperatura media por mes: 5
Ingrese la temperatura media por mes: 5
Ingrese el país: Uruguay
Ingrese la temperatura media por mes: 7
Ingrese la temperatura media por mes: 7
Ingrese la temperatura media por mes: 7
Nombres de los países y temperaturas medias mensuales
Chile 22.0 22.0 22.0
Argentina 2.0 3.0 3.0
Brasil 5.0 5.0 5.0
Temperaturas medias por país
Chile 22.0
Argentina 2.666666666666665
Brasil 5.0
País con temperaturas medias más altas
País con temperatura media más alta  Chile
>>>
Ln: 31  Col: 0
```

Capítulo 103.- Listas: carga por teclado de componentes de tipo lista – 6

Problema propuesto:

Definir una lista y almacenar los nombres de 3 empleados.
Por otro lado definir otra lista y almacenar en cada elemento una sublista con los números de días del mes que el empleado faltó.
Imprimir los nombres de empleados y los días que faltó.
Mostrar los empleados con la cantidad de inasistencias.
Finalmente mostrar el nombre o los nombres de empleados que faltaron menos días.

Vamos a programar:

ejercicio103.py - D:/Curso completo de Python/ejercicio103.py (3.11.1)

File Edit Format Run Options Window Help

```
empleados=[]
diasfalta=[]
for x in range(3):
    nombre=input("Nombre del empleado: ")
    empleados.append(nombre)
    diasfalta.append([])
    dias=int(input("Días que ha faltado al trabajo: "))
    for y in range(dias):
        dia=int(input("Día del mes que no ha ido al trabajo: "))
        diasfalta[x].append(dia)
for x in range(3):
    print("------------------------")
    print(empleados[x])
    print("------------------------")
    for y in range(len(diasfalta[x])):
        print("día: ", diasfalta[x][y])
for x in range(3):
    print(empleados[x], len(diasfalta[x]))

min=diasfalta[0]
for x in range(1,3):
    if diasfalta[x]<min:
        min=diasfalta[x]

for x in range(3):
    if diasfalta[x]==min:
        print("El empleado con menos faltas es ", empleados[x])

```

Guardar el programa con el nombre de Ejercicio103.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio103.py =============
Nombre del empleado: Ana
Días que ha faltado al trabajo: 2
Día del mes que no ha ido al trabajo: 1
Día del mes que no ha ido al trabajo: 2
Nombre del empleado: Juan
Días que ha faltado al trabajo: 0
```

```
Nombre del empleado: Carlos
Días que ha faltado al trabajo: 4
Día del mes que no ha ido al trabajo: 4
Día del mes que no ha ido al trabajo: 5
Día del mes que no ha ido al trabajo: 6
Día del mes que no ha ido al trabajo: 7
------------------------
Ana
------------------------
día:  1
día:  2
------------------------
Juan
------------------------
------------------------
Carlos
------------------------
día:  4
día:  5
día:  6
día:  7
Ana 2
Juan 0
Carlos 4
El empleado con menos faltas es  Juan
>>>
```

Capítulo 104.- Listas: carga por teclado de componentes de tipo lista – 7

Problema propuesto:

Desarrollar un programa que cree una lista de 50 elementos. El primer elemento es una lista con un elemento entero, el segundo elemento es una lista de dos elementos, etc.

La lista debería tener esta estructura y asignarles esos valores a medida que se crean elementos:

[[1], [1,2], [1,2,3], [1,2,3,4], [1,2,3,4,5], etc . . . .]

Vamos a programar:

ejercicio104.py - D:/Curso completo de Python/ejercicio104.py (3.11.1)

File Edit Format Run Options Window Help

```
# Método 1
lista=[]
cant=1
for k in range(50):
    lista.append([])
    valor=1
    for x in range(cant):
        lista[k].append(valor)
        valor=valor+1
    cant=cant+1
print(lista)

# Método 2
lista=[]
cant=1
while cant<=50:
    lista.append([])
    for y in range(cant):
        lista[cant-1].append(y+1)
    cant=cant+1
print(lista)
```

Guardamos el programa con el nombre de Ejercicio104.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio104.py =============
[[1], [1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2, 3, 4, 5], [1, 2, 3, 4, 5, 6], [1,
2, 3, 4, 5, 6, 7], [1, 2, 3, 4, 5, 6, 7, 8], [1, 2, 3, 4, 5, 6, 7, 8, 9], [1, 2,
 3, 4, 5, 6, 7, 8, 9, 10], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], [1, 2, 3, 4, 5,
6, 7, 8, 9, 10, 11, 12], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13], [1, 2, 3,
4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1
3, 14, 15], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], [1, 2, 3, 4
, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10
, 11, 12, 13, 14, 15, 16, 17, 18], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1
4, 15, 16, 17, 18, 19], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
```

. . .

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
, 30], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 2
1, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10
, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30
, 31, 32, 33, 34], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1
8, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35], [1, 2, 3
, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
 31, 32, 33, 34, 35, 36, 37], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35
, 36, 37, 38], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1
9, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 3
9], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40], [1,
 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41], [1, 2,
 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24
, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42], [1, 2
, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 2
4, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43],
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,
23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42,
43, 44], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
 41, 42, 43, 44, 45], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37
, 38, 39, 40, 41, 42, 43, 44, 45, 46], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1
3, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 3
3, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47], [1, 2, 3, 4, 5, 6, 7
, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27,
28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
48], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
 42, 43, 44, 45, 46, 47, 48, 49], [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14
, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34
, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]]
>>>
```

Ln: 7 Col: 0

Capítulo 105.- Lista: eliminación de elementos – 1

Hemos visto que una lista la podemos iniciar por asignación indicando sus elementos.

lista=[10, 20, 30, 40]

También podemos agregarle elementos al final mediante el método append:

lista.append(120)

Si ahora imprimimos la lista tenemos como resultado:

[10, 20, 30, 40, 120]

Otra característica fundamental de las listas en Python es que podemos eliminar cualquiera de sus componentes llamando al método pop e indicando la posición del elemento a borrar:

lista.pop(0)

Ahora si imprimimos la lista luego de eliminar el primer elemento el resultado es:

[20, 30, 40, 120]

Otra cosa que hay que hacer notar que cuando un elemento de la lista se elimina no queda una posición vacía, sino que se desplazan todos los elementos de la derecha una posición.

El método pop retorna el valor almacenado en la lista en la posición indicada, aparte de borrarlo.

lista=[10, 20, 30, 40]

print(lista.pop(0)) # Imprime un 10.

Problema:

Crea una lista por asignación con 5 enteros. Elimina el primero, el tercero y el último de la lista.

Guardamos el programa con el nombre de Ejercicio105.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio105.py =============
[10, 20, 30, 40, 50]
50
30
10
[20, 40]
[10, 20, 30, 40, 50]
10
30
50
[20, 40]
>>>
```

En el primer método como empezamos a borrar por el final los elementos no se desplazan, pero en el segundo método al borrar desde el principio los elementos a su derecha son desplazados a la izquierda una posición.

Capítulo 106.- Listas: eliminación de elementos – 2

Problema:

Crear una lista y almacenar 10 enteros pedidos por teclado. Eliminar todos los elementos que sean iguales al número entero 5.

Vamos a programar:

ejercicio106.py - D:/Curso completo de Python/ejercicio106.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[]
for x in range(10):
    valor=int(input("Ingrese un valor:"))
    lista.append(valor)
print(lista)
y=0
while y<=(len(lista)-1):
    if lista[y]==5:
        lista.pop(y)
    else:
        y=y+1
print(lista)
```

Guardamos el programa con el nombre de Ejercicio106.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio106.py =============
Ingrese un valor:5
Ingrese un valor:1
Ingrese un valor:5
Ingrese un valor:2
Ingrese un valor:5
Ingrese un valor:3
Ingrese un valor:5
Ingrese un valor:4
Ingrese un valor:5
Ingrese un valor:6
[5, 1, 5, 2, 5, 3, 5, 4, 5, 6]
[1, 2, 3, 4, 6]
>>>
```

Capítulo 107.- Listas: Eliminación de elementos – 3

Problema propuesto:

Crea dos listas paralelas. En la primera ingresa los nombres de empleados y en la segunda los sueldos de cada empleado.

Ingresar por teclado cuando inicia el programa la cantidad de empleados en la empresa.

Borra luego todos los empleados que tienen un sueldo mayor a 1000 (tanto el sueldo como su nombre).

Vamos a programar:

ejercicio107.py - D:/Curso completo de Python/ejercicio107.py (3.11.1)

File Edit Format Run Options Window Help

```
1  empleados=[]
2  sueldos=[]
3  cantidad=int(input("Cantidad de empleados: "))
4  for x in range(cantidad):
5      empleado=input("Ingrese el nombre del empleado: ")
6      empleados.append(empleado)
7      sueldo=int(input("Ingrese el sueldo del empleado: "))
8      sueldos.append(sueldo)
9  print("Nombres de los empleados")
10 print(empleados)
11 print("Sueldo de los empleados")
12 print(sueldos)
13 cant=0
14 while cant<len(sueldos):
15     if sueldos[cant]>1000:
16         sueldos.pop(cant)
17         empleados.pop(cant)
18     else:
19         cant=cant+1
20 print("Nombres de los empleados que cobrar 1000 o menos")
21 for x in range(len(empleados)):
22     print(empleados[x], sueldos[x])
23
```

Guardamos el programa con el nombre de Ejercicio107.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio107.py =============
Cantidad de empleados: 3
Ingrese el nombre del empleado: Juan
Ingrese el sueldo del empleado: 1200
Ingrese el nombre del empleado: Ana
Ingrese el sueldo del empleado: 950
Ingrese el nombre del empleado: Carlos
Ingrese el sueldo del empleado: 1000
Nombres de los empleados
['Juan', 'Ana', 'Carlos']
Sueldo de los empleados
[1200, 950, 1000]
Nombres de los empleados que cobrar 1000 o menos
Ana 950
Carlos 1000
>>>
```

Capítulo 108.- Listas: Eliminación de elementos – 4

Problema propuesto:

Crear una lista de 5 enteros y cargarlos por teclado. Borrar los elementos mayores o iguales a 10 y generar una nueva lista con dichos valores.

Vamos a programar:

ejercicio108.py - D:/Curso completo de Python/ejercicio108.py (3.11.1)

File Edit Format Run Options Window Help

```
lista=[]
borrados=[]
for x in range(5):
    valor=int(input("Ingrese un valor: "))
    lista.append(valor)
print("Listas originales")
print(lista)
print(borrados)
cant=0
while cant<len(lista):
    if lista[cant]>=10:
        borrados.append(lista[cant])
        lista.pop(cant)
    else:
        cant=cant+1
print("Lista con los elementos eliminados")
print(lista)
print("Lista generada con los valores mayores o iguales a 10")
print(borrados)
```

Guardamos el programa con el nombre Ejercicio108.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio108.py =============
Ingrese un valor: 23
Ingrese un valor: 2
Ingrese un valor: 45
Ingrese un valor: 5
Ingrese un valor: 6
Listas originales
[23, 2, 45, 5, 6]
[]
Lista con los elementos eliminados
[2, 5, 6]
Lista generada con los valores mayores o iguales a 10
[23, 45]
>>>
```

Capítulo 109.- Concepto de funciones: Programación estructurada – 1

Hasta ahora hemos trabajado con una metodología de programación lineal. Todas las instrucciones de nuestro archivo *.py se ejecutan en forma secuencial de principio a fin.

Esta forma de organizar un programa solo puede ser llevado a cabo si el mismo se muy pequeño (decenas de líneas)

Cuando los problemas a resolver tienden a ser más grandes la metodología de programación lineal se vuelve ineficiente y compleja.

El segundo paradigma de programación que veremos en la programación estructurada.

La programación estructurada busca dividir o descomponer un problema complejo en pequeños problemas. La solución de cada uno de esos pequeños problemas nos trae la solución del problema complejo.

En Python el planteo de esas pequeñas soluciones al problema complejo se hace dividiendo el programa en funciones.

Una función es un conjunto de instrucciones en Python que resuelve un problema específico.

El lenguaje Python ya tiene incorporada algunas funciones básicas. Algunas de ellas ya las utilizamos en conceptos anteriores como son las funciones: pint, len y range.

Veamos ahora como crear nuestras propias funciones.

El tema de funciones es en un principio puede presentar dificultades para entenderlo y ver sus ventajas ante la metodología de programación lineal que veníamos trabajando en conceptos anteriores.

Los primeros problemas que presentamos nos pueden parecer que sea más conveniente utilizar la programación lineal en vez de la programación estructurada por funciones.

A medida que avancemos veremos que si un programa empieza a ser más complejo (cientos de líneas, miles de líneas o más) la división en pequeñas funciones nos permitirá tener el programa más ordenado y fácil de entender y por lo tanto en mantener.

Problema:

Confeccionar una aplicación que muestre una presentación en pantalla del programa. Solicite la carga de dos valores y nos muestre la suma. Mostrar finalmente un mensaje de despedida del programa.

Implementar estas actividades en tres funciones.

Vamos a programar:

```
*ejercicio109.py - D:/Curso completo de Python/ejercicio109.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1  def presentacion():
2      print("Programa que permite cargar por teclado dos valores")
3      print("Efectua la suma de los valores")
4      print("Muestra el resultado de la suma")
5      print("****************************************************")
6
7  def carga_suma():
8      valor1=int(input("Ingrese el primer valor: "))
9      valor2=int(input("Ingrese el segundo valor: "))
10     suma=valor1+valor2
11     print("La suma de los dos valores es", suma)
12
13 def finalizacion():
14     print("************************************")
15     print("Gracias por utilizar este programa.")
16
17 # Bloque principal de nuestro programa
18 presentacion()
19 carga_suma()
20 finalizacion()
21
```

Con la palabra def se declaran las funciones.

Guardamos el programa con el nombre de Ejercicio109.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
   Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
   AMD64)] on win32
   Type "help", "copyright", "credits" or "license()" for more information.
>>>
   ============= RESTART: D:/Curso completo de Python/ejercicio109.py =============
   Programa que permite cargar por teclado dos valores
   Efectua la suma de los valores
   Muestra el resultado de la suma
   ****************************************************
   Ingrese el primer valor: 5
   Ingrese el segundo valor: 7
   La suma de los dos valores es 12
   ************************************
   Gracias por utilizar este programa.
>>>
```

Dos pasos importante uno es declara la función y un segundo paso llamar a la función.

Las funciones puede ser llamadas múltiples veces, de este modo evitamos codigo repetitivo.

```
# Bloque principal de nuestro programa
presentacion()
carga_suma()
carga_suma()
finalizacion()
```

Nos preguntará dos veces por dos valores y calculará la suma de dichos valores.

Capítulo 110.- Concepto de funciones: Programación estructurada – 2

Problema:

Confeccionar una aplicación que solicite la carga de dos valores enteros y muestre la suma.

Repetir la carga e impresión de la suma 5 veces.

Mostrar una línea separatoria después de cada vez que cargamos dos valores y su suma.

Vamos a programar:

ejercicio110.py - D:/Curso completo de Python/ejercicio110.py (3.11.1)

File Edit Format Run Options Window Help

```
def sumar_valores():
    valor1=int(input("Ingrese el primer valor: "))
    valor2=int(input("Ingrese el segundo valor: "))
    suma=valor1+valor2
    print("La suma de los dos valores es", suma)
    print("*************************************")

# Bloque principal
# Método 2
sumar_valores()
sumar_valores()


# Método 2
for x in range(3):
    sumar_valores()

```

Guardamos el programa con el nombre de Ejercicio110.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio110.py =============
Ingrese el primer valor: 12
Ingrese el segundo valor: 4
La suma de los dos valores es 16
*************************************
Ingrese el primer valor: 7
Ingrese el segundo valor: 9
La suma de los dos valores es 16
*************************************
Ingrese el primer valor: 5
Ingrese el segundo valor: 6
La suma de los dos valores es 11
*************************************
Ingrese el primer valor: 14
Ingrese el segundo valor: 21
La suma de los dos valores es 35
*************************************
Ingrese el primer valor: 33
Ingrese el segundo valor: 10
La suma de los dos valores es 43
*************************************
>>>
```

Otro forma de realizarlo:

ejercicio110.py - D:/Curso completo de Python/ejercicio110.py (3.11.1)

File Edit Format Run Options Window Help

```python
def sumar_valores():
    valor1=int(input("Ingrese el primer valor: "))
    valor2=int(input("Ingrese el segundo valor: "))
    suma=valor1+valor2
    print("La suma de los dos valores es", suma)

def separacion():
    print("*************************************")

# Bloque principal
# Método 2
sumar_valores()
separacion()
sumar_valores()
separacion()

# Método 2
for x in range(3):
    sumar_valores()
    separacion()
```

Hemos creado 2 funciones una llamada sumar_valores() y una segunda llamada separacion() que iremos llamando dentro del Bloque principal.

Capítulo 111.- Concepto de funciones: Programación estructurada – 3

Problema propuesto:

Desarrollar un programa con dos funciones. La primera que solicite el ingreso de un entero y muestre el cuadrado de dicho valor. La segunda que solicite la caga de dos valores y muestre el producto de los mismos. Llamar desde el bloque del programa principal ambas funciones.

Vamos a programar:

ejercicio111.py - D:/Curso completo de Python/ejercicio111.py (3.11.1)

File Edit Format Run Options Window Help

```
def calcular_cuadrado():
    num=int(input("Ingrese un número: "))
    cuadrado=num*num
    print("El cuadrado de dicho número", cuadrado)

def calcular_producto():
    num1=int(input("Ingrese el primer número: "))
    num2=int(input("Ingrese el segundo número: "))
    producto=num1*num2
    print("El producto de los números es ", producto)

# Bloque principal
calcular_cuadrado()
calcular_producto()
```

Guardamos el programa con el nombre de Ejercicio111.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio111.py =============
Ingrese un número: 3
El cuadrado de dicho número 9
Ingrese el primer número: 4
Ingrese el segundo número: 5
El producto de los números es  20
>>>
```

Capítulo 112.- Concepto de funciones: Programación estructurada – 4

Problema propuesto:

Desarrollar un programa que solicite la carga de tres valores y muestre el menor. Desde el bloque principal del programa llamar 2 veces dicha función (sin utilizar una estructura repetitiva).

Vamos a programar:

ejercicio112.py - D:/Curso completo de Python/ejercicio112.py (3.11.1)

File Edit Format Run Options Window Help

```
def valor_menor():
    num1=int(input("Ingrese el primer valor: "))
    num2=int(input("Ingrese el segundo valor: "))
    num3=int(input("Ingrese el tercer valor: "))
    if (num1<num2 and num1<num3):
        print(num1)
    else:
        if(num2<num3):
            print(num2)
        else:
            print(num3)

# Bloque principal
valor_menor()
valor_menor()
```

Guardamos el programa con el nombre de Ejercicio112.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio112.py =============
Ingrese el primer valor: 5
Ingrese el segundo valor: 6
Ingrese el tercer valor: 2
2
Ingrese el primer valor: 8
Ingrese el segundo valor: 9
Ingrese el tercer valor: 12
8
>>>
```

Capítulo 113.- Funciones: parámetros – 1

Vimos en el concepto anterior que una función resuelve una parte de nuestro algoritmo.

Tenemos por un lado la declaración de la función por medio de un nombre y el algoritmo de la función seguidamente. Luego para que se ejecute la función la llamamos desde el bloque principal de nuestro programa.

Ahora veremos que una función puede tener parámetros para recibir datos. Los parámetros nos permiten comunicarle algo a la función y la hace más flexible.

Una función con parámetros nos hace más flexible la misma para utilizarla en distintas circunstancias.

Problema:

Confeccionar una aplicación que muestre una presentación en pantalla del programa.
Solicite la carga de dos valores y nos muestre la suma.
Mostrar finalmente un mensaje de despedida del programa.

Vamos a programar:

ejercicio113.py - D:/Curso completo de Python/ejercicio113.py (3.11.1)

File Edit Format Run Options Window Help

```
def mostrar_mensaje(mensaje):
    print("**************************************************")
    print(mensaje)
    print("**************************************************")

def carga_suma():
    valor1=int(input("Ingrese el primer número: "))
    valor2=int(input("Ingrese el segundo número: "))
    suma=valor1+valor2
    print("La suma de los dos valores es", suma)

# Bloque principal
mostrar_mensaje("El programa calcula la suma de dos valores ingresados por teclado")
carga_suma()
mostrar_mensaje("Grácias por utilizar este programa")
```

Guardamos el programa con el nombre de Ejercicio113.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio113.py =============
**************************************************
El programa calcula la suma de dos valores ingresados por teclado
**************************************************
Ingrese el primer número: 7
Ingrese el segundo número: 12
La suma de los dos valores es 19
**************************************************
Grácias por utilizar este programa
**************************************************
>>>
```

También podemos enviar una variable.

```
*ejercicio113.py - D:/Curso completo de Python/ejercicio113.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
def mostrar_mensaje(mensaje):
    print("*************************************************")
    print(mensaje)
    print("*************************************************")

def carga_suma():
    valor1=int(input("Ingrese el primer número: "))
    valor2=int(input("Ingrese el segundo número: "))
    suma=valor1+valor2
    print("La suma de los dos valores es", suma)

# Bloque principal
cadena="El programa calcula la suma de dos valores ingresados por teclado"
mostrar_mensaje(cadena)
carga_suma()
cadena="Grácias por utilizar este programa"
mostrar_mensaje(cadena)
```

Si ejecutamos de nuevo observarás que el resultado es el mismo.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio113.py =============
**********************************************
El programa calcula la suma de dos valores ingresados por teclado
**********************************************
Ingrese el primer número: 7
Ingrese el segundo número: 12
La suma de los dos valores es 19
**********************************************
Grácias por utilizar este programa
**********************************************
>>>
```

Si tenemos que enviar más de un parámetro este irá separado por comas.

Capítulo 114.- Funciones: parámetros – 2

Problema:

Confeccionar una función que reciba tres enteros y nos muestre el mayor de ellos. La carga de los valores hacerlo por teclado en otra función.

Vamos a programar:

```
ejercicio114.py - D:/Curso completo de Python/ejercicio114.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
def  valor_mayor(num1, num2, num3):
    if num1>num2 and num1>num3:
        print(num1)
    else:
        if num2>num3:
            print(num2)
        else:
            print(num3)

def pedir_valores():
    nu1=int(input("Ingrese el primer valor: "))
    nu2=int(input("Ingrese el segundo valor: "))
    nu3=int(input("Ingrese el tercer valor: "))
    valor_mayor(nu1, nu2, nu3)

# Bloque principal
pedir_valores()
```

Guardamos el programa con el nombre de Ejercicio114.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio114.py =============
Ingrese el primer valor: 7
Ingrese el segundo valor: 10
Ingrese el tercer valor: 3
10
>>>
```

En este ejemplo podemos observar como de una función se llama a otra función.

La función que llama otra función, esta tiene que estar declarada.

Capítulo 115.- Funciones: parámetros – 3

Problema:

Desarrollar un programa que permita ingresar el lado de un cuadrado. Luego preguntar si quiere calcular y mostrar su perímetro o superficie.

Vamos a programar:

```
ejercicio115.py - D:/Curso completo de Python/ejercicio115.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
def calcular_superficie(l):
    superficie=l*l
    print("La superficie es", superficie)

def calcular_perimetro(l):
    perimetro=l*4
    print("El perímetro es", perimetro)

def cargar_lado():
    lado_cuadrado=int(input("Ingrese el lado de un cuadrado: "))
    pregunta=input("Para calcular  el perímetro pulsar 'p' y para superficie 's': ")
    if pregunta=="p" or pregunta=="P":
        calcular_perimetro(lado_cuadrado)
    else:
        if pregunta=="s" or pregunta=="S":
            calcular_superficie(lado_cuadrado)
        else:
            print("Respuesta incorrecta")

# Bloque principal
cargar_lado()

```

Guardar el programa con el nombre de Ejercicio115.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio115.py =============
Ingrese el lado de un cuadrado: 5
Para calcular  el perímetro pulsar 'p' y para superficie 's': p
El perímetro es 20
>>>
============= RESTART: D:/Curso completo de Python/ejercicio115.py =============
Ingrese el lado de un cuadrado: 5
Para calcular  el perímetro pulsar 'p' y para superficie 's': s
La superficie es 25
>>>
============= RESTART: D:/Curso completo de Python/ejercicio115.py =============
Ingrese el lado de un cuadrado: 5
Para calcular  el perímetro pulsar 'p' y para superficie 's': g
Respuesta incorrecta
>>>
```

Capítulo 116.- Funciones: parámetros – 4

Problema propuesto:

Desarrolla una función que reciba un string como parámetro y nos muestre la cantidad de vocales.

Llamarla desde el bloque principal del programa 3 veces con string distintos.

Vamos a programar:

ejercicio116.py - D:/Curso completo de Python/ejercicio116.py (3.11.1)

File Edit Format Run Options Window Help

```
def cuenta_vocales(cadena):
    cadena=cadena.lower()
    cantidad=0
    for x in range(len(cadena)):
        if cadena[x]=="a" or cadena[x]=="e" or cadena[x]=="i" \
           or cadena[x]=="o" or cadena[x]=="u":
            cantidad=cantidad+1
    print("La frase (", cadena , ") tiene ", cantidad, " de vocales.")

# Bloque principal
for x in range(3):
    texto=input("Ingrese un texto: ")
    cuenta_vocales(texto)
```

Utilizaremos la barra '\' cuando queremos realizar un salto de línea en nuestro código y no afecte al programa.

Guardamos el programa con el nombre de Ejercicio116.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio116.py =============
Ingrese un texto: Los dias de la semana son siete.
La frase ( los dias de la semana son siete. ) tiene  12  de vocales.
Ingrese un texto: Es un dia de pleno verano.
La frase ( es un dia de pleno verano. ) tiene  10  de vocales.
Ingrese un texto: Estoy programando con Python.
La frase ( estoy programando con python. ) tiene  8  de vocales.
>>>
```

Capítulo 117.- Funciones: parámetros – 5

Problema propuesto:

Confeccionar una función que reciba tres enteros y los muestre ordenados de menor a mayor. En otra función solicitar la carga de 3 enteros por teclado y proceder a llamar a la primera función definida.

Vamos a programar:

ejercicio117.py - D:/Curso completo de Python/ejercicio117.py (3.11.1)

File Edit Format Run Options Window Help

```
def ordenar(num1, num2, num3):
    if num1<num2 and num1<num3:
        if num2<num3:
            print(num1, num2, num3)
        else:
            print(num1, num3, num1)
    else:
        if num2<num3:
            if num1<num3:
                print(num2, num1, num3)
            else:
                print(num2, num3, num1)
        else:
            if num1<num3:
                print(num3, num1, num2)
            else:
                print(num3, num2, num1)

def cargar_numeros():
    n1=int(input("Ingrese el primer número: "))
    n2=int(input("Ingrese el segundo número: "))
    n3=int(input("Ingrese el tercer número: "))
    ordenar(n1, n2, n3)

# Bloque principal
cargar_numeros()
```

Guardamos el programa con el nombre de Ejercicio117.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio117.py =============
Ingrese el primer número: 1
Ingrese el segundo número: 2
Ingrese el tercer número: 3
1 2 3
>>>
============= RESTART: D:/Curso completo de Python/ejercicio117.py =============
Ingrese el primer número: 3
Ingrese el segundo número: 2
Ingrese el tercer número: 1
1 2 3
>>>
```

```
============= RESTART: D:/Curso completo de Python/ejercicio117.py =============
Ingrese el primer número: 2
Ingrese el segundo número: 1
Ingrese el tercer número: 3
1 2 3
>>>
```

Capítulo 118.- Funciones: retorno de datos – 1

Hemos comenzado a pensar con la metodología de programación estructurada. Buscamos dividir un problema en subproblemas y plantear algoritmos en Python que los resuelvan.

Vimos que una función la definimos mediante un nombre y que puede recibir datos por medio de sus parámetros.

Los parámetros son la forma para que una función reciba datos para ser procesados. Ahora veremos otra característica de las funciones que es la de devolver un dato a quien invocó la función (recordemos que una función la podemos llamar desde el bloque principal de nuestro programa o desde otra función que desarrollemos).

Problema:

Confeccionar una función que le enviemos como parámetro el valor del lado de un cuadrado y nos retorne la superficie.

Vamos a programar:

ejercicio118.py - D:/Curso completo de Python/ejercicio118.py (3.11.1)

File Edit Format Run Options Window Help

```
def retornar_superficie(lado):
    sup=lado*lado
    return sup

# Bloque principal
la=int(input("Ingrese el valor del lado del cuadrado: "))
superficie=retornar_superficie(la)
print("La superficie del cuadrado es", superficie)
la=int(input("Ingrese el valor del lado del cuadrado: "))
print("La superficie del cuadrado es", retornar_superficie(la))

```

Guardamos el programa con el nombre de Ejercicio118.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio118.py =============
Ingrese el valor del lado del cuadrado: 10
La superficie del cuadrado es 100
Ingrese el valor del lado del cuadrado: 5
La superficie del cuadrado es 25
>>>
```

```
la=int(input("Ingrese el valor del lado del cuadrado: "))
if retornar_superficie(la)==100:
    print("La superficie es 100")
else:
    print("La superficie no es 100")
```

Esta estructura condicional también es correcta.

Capítulo 119.- Funciones: retorno de datos – 2

Problema:

Confeccionar una función que le enviemos como parámetro dos enteros y nos retorne el mayor.

Vamos a programar:

ejercicio119.py - D:/Curso completo de Python/ejercicio119.py (3.11.1)

File Edit Format Run Options Window Help

```
def retornar_mayor(num1, num2):
    if num1>num2:
        mayor=num1
    else:
        mayor=num2
    return mayor

# Bloque principal
n1=int(input("Ingrese el primer valor: "))
n2=int(input("Ingrese el segundo valor: "))
print("El valor mayor es", retornar_mayor(n1, n2))
```

Guardamos el programa con el nombre de Ejercicio119.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio119.py =============
Ingrese el primer valor: 20
Ingrese el segundo valor: 3
El valor mayor es 20
>>>
============= RESTART: D:/Curso completo de Python/ejercicio119.py =============
Ingrese el primer valor: 3
Ingrese el segundo valor: 17
El valor mayor es 17
>>>
```

También será correcto de la siguiente forma:

ejercicio119.py - D:/Curso completo de Python/ejercicio119.py (3.11.1)

File Edit Format Run Options Window Help

```
def retornar_mayor(num1, num2):
    if num1>num2:
        return num1
    else:
        return num2

# Bloque principal
n1=int(input("Ingrese el primer valor: "))
n2=int(input("Ingrese el segundo valor: "))
print("El valor mayor es", retornar_mayor(n1, n2))
```

Cuando encuentra un return retorna el valor y finaliza la función.

Capítulo 120.- Funciones: retorno de datos – 3

Problema:

Confeccionar una función que le enviemos como parámetro un string y nos retorne la cantidad de caracteres que tiene. En el bloque principal solicitar la carga de dos nombres por teclado y llamar a la función dos veces. Imprimir en el bloque principal cual de las dos palabras tiene más caracteres.

Vamos a programar:

ejercicio120.py - D:/Curso completo de Python/ejercicio120.py (3.11.1)

File Edit Format Run Options Window Help

```
1  def num_caracteres(nombre):
2      return len(nombre)
3
4  # Bloque principal
5  nom1=input("Ingrese el primer nombre: ")
6  num_caracteres1=num_caracteres(nom1)
7  nom2=input("Ingrese el segundo nombre: ")
8  num_caracteres2=num_caracteres(nom2)
9  if num_caracteres1==num_caracteres2:
10     print("Los nombres tienen la misma cantidad de caracteres")
11 else:
12     if num_caracteres1>num_caracteres2:
13         print(nom1, "es más largo")
14     else:
15         print(nom2, "es más largo")
16
```

Guardar el programa con el nombre de Ejercicio120.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio120.py =============
Ingrese el primer nombre: Juan
Ingrese el segundo nombre: Luis
Los nombres tienen la misma cantidad de caracteres
>>>
============= RESTART: D:/Curso completo de Python/ejercicio120.py =============
Ingrese el primer nombre: Carlos
Ingrese el segundo nombre: Isidoro
Isidoro es más largo
>>>
============= RESTART: D:/Curso completo de Python/ejercicio120.py =============
Ingrese el primer nombre: Luis
Ingrese el segundo nombre: Antonio
Antonio es más largo
>>>
```

Capítulo 121.- Funciones: retorno de datos – 4

Problema propuesto:

Elaborar una función que reciba tres enteros y nos retorne el valor promedio de los mismos.

Vamos a programar:

```
ejercicio121.py - D:/Curso completo de Python/ejercicio121.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1  def promedio(num1, num2, num3):
2      pro=(num1+num2+num3)/3
3      return pro
4
5  # Bloque principal
6  n1=int(input("Ingrese el primer número: "))
7  n2=int(input("Ingrese el segundo número: "))
8  n3=int(input("Ingrese el tercer número: "))
9  print("Valor promedio de los tres números es", promedio(n1,n2,n3))
10
```

Guardamos el programa con el nombre de Ejercicio121.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
   Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
   AMD64)] on win32
   Type "help", "copyright", "credits" or "license()" for more information.
>>>
   ============= RESTART: D:/Curso completo de Python/ejercicio121.py =============
   Ingrese el primer número: 20
   Ingrese el segundo número: 45
   Ingrese el tercer número: 66
   Valor promedio de los tres números es 43.66666666666664
>>>
```

Capítulo 122.- Funciones: retorno de datos – 5

Problema propuesto:

Elaborar una función que nos retorne el perímetro de un cuadrado pasando como parámetro el valor de un lado.

Vamos a programar:

```
*ejercicio122.py - D:/Curso completo de Python/ejercicio122.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
1 def perimetro(lado):
2     peri=lado*4
3     return peri
4
5 # Bloque principal
6 l=int(input("Ingrese el lado de un cuadrado: "))
7 print("El perímetro del cuadrado es", perimetro(l))
8
```

El programa lo guardamos como Ejercicio122.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio122.py =============
    Ingrese el lado de un cuadrado: 10
    El perímetro del cuadrado es 40
>>>
```

Capítulo 123.- Funciones: retorno de datos – 6

Problema propuesto:

Confeccionar una función que calcule la superficie de un rectángulo y la retorne, la función recibe como parámetros los valores de dos de sus lados:

def retornar_superficie(lado1, lado2):

En el bloque principal del programa cargar los lados de dos rectángulos y luego mostrar cual de los dos tiene una superficie mayor:

Vamos a programar:

```
ejercicio123.py - D:/Curso completo de Python/ejercicio123.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
def retornar_superficie(lado1, lado2):
    sup=lado1*lado2
    return sup

# Bloque principal
print("Primer rectángulo")
l1=int(input("Ingrese el primer lado del rectángulo: "))
l2=int(input("Ingrese el segundo lado del rectángulo: "))
print("Segundo rectángulo")
l3=int(input("Ingrese el primer lado del rectángulo: "))
l4=int(input("Ingrese el segundo lado del rectángulo: "))
if retornar_superficie(l1,l2)==retornar_superficie(l3,l4):
    print("Los rectangulos tienen la misma superficie")
else:
    if(retornar_superficie(l1,l2)>retornar_superficie(l3,l4)):
        print("El 1° rectángulo tiene una mayor superficie con ",\
              retornar_superficie(l1,l2))
    else:
        print("El 2° rectangulo tiene una mayor superficie con ",\
              retornar_superficie(l3,l4))
```

Con la barra '\' hacemos un salto de línea en el código y no afecta al programa.

Guardamos el programa con el nombre de Ejercicio123.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio123.py =============
Primer rectángulo
Ingrese el primer lado del rectángulo: 10
Ingrese el segundo lado del rectángulo: 20
Segundo rectángulo
Ingrese el primer lado del rectángulo: 5
Ingrese el segundo lado del rectángulo: 3
El 1° rectángulo tiene una mayor superficie con  200
>>>
```

```
============= RESTART: D:/Curso completo de Python/ejercicio123.py =============
Primer rectángulo
Ingrese el primer lado del rectángulo: 12
Ingrese el segundo lado del rectángulo: 5
Segundo rectángulo
Ingrese el primer lado del rectángulo: 10
Ingrese el segundo lado del rectángulo: 10
El 2° rectangulo tiene una mayor superficie con  100
>>>
```

Capítulo 124.- Funciones: retorno de datos – 7

Problema propuesto:

Plantear una función que reciba un string en mayúsculas o minúsculas y retorne la cantidad de letras 'a' o 'A'.

Vamos a programar:

ejercicio124.py - D:/Curso completo de Python/ejercicio124.py (3.11.1)

File Edit Format Run Options Window Help

```
def cuenta_a(texto):
    cantidad=0
    for x in range(len(texto)):
        if texto[x]=="a" or texto[x]=="A":
            cantidad=cantidad+1
    return cantidad

# Bloque principal
texto=input("Ingrese una palabra: ")
print("La palabra", texto, "tiene", cuenta_a(texto), "a")
```

Guardar el programa con el nombre de Ejercicio124.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio124.py =============
Ingrese una palabra: casa
La palabra casa tiene 2 a
>>>
============= RESTART: D:/Curso completo de Python/ejercicio124.py =============
Ingrese una palabra: cAsa
La palabra cAsa tiene 2 a
>>>
```

Capítulo 125.- Funciones: parámetros de tipo lista – 1

Hemos visto que si trabajamos con programación estructurada debemos dividir nuestro problema en trocitos y plantear algoritmos que resuelvan cada uno de los trozos.

La división den Python se hace implementando funciones. La comunicación entre estas funciones se produce mediante el envío de datos a través de los parámetros y el valor que retorna la función.

Hasta ahora hemos resuelto problemas enviando datos simples como enteros, float y cadena de caracteres. En este concepto veremos que una función puede recibir tanto datos simples como estructura de datos.

La estructura de datos vista hasta el momento y que podemos enviarle a una función es la lista.

Problema:

Definir por asignación una lista de enteros en el bloque principal del programa. Elaborar tres funciones, la primera recibe la lista y retorna la suma de todos sus elementos, la segunda recibe la lista y retorna el mayor valor y la última recibe la lista y retorna el menor.

Vamos a programar:

ejercicio125.py - D:/Curso completo de Python/ejercicio125.py (3.11.1)

File Edit Format Run Options Window Help

```
def sumarlista(lista):
    suma=0
    for x in range(len(lista)):
        suma=suma+(lista[x])
    return suma

def maxlista(lista):
    max=lista[0]
    for x in range(1, len(lista)):
        if lista[x]>max:
            max=lista[x]
    return max

def minlista(lista):
    min=lista[0]
    for x in range(1, len(lista)):
        if lista[x]<min:
            min=lista[x]
    return min


# Bloque principal
listavalores=[10, 20, 30, 40, 50]
print("Lista completa")
print(listavalores)
print("La suma de todos los elementos es", sumarlista(listavalores))
print("El mayor elemento de la lista es", maxlista(listavalores))
print("El menor elemento de la lista es", minlista(listavalores))
```

Guardamos el programa con el nombre de Ejercicio125.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio125.py =============
Lista completa
[10, 20, 30, 40, 50]
La suma de todos los elementos es 150
El mayor elemento de la lista es 50
El menor elemento de la lista es 10
```

Capítulo 126.- Funciones: parámetros de tipo lista – 2

Problema:

Crear y cargar por teclado en el bloque principal del programa una lista de 5 enteros. Implementar una función que imprima el mayor y el menor valor de la lista.

Vamos a programar:

ejercicio126.py - D:/Curso completo de Python/ejercicio126.py (3.11.1)

File Edit Format Run Options Window Help

```
def max_min_lista(lista):
    max=lista[0]
    min=lista[0]
    for x in range(1, len(lista)):
        if lista[x]>max:
            max=lista[x]
        if lista[x]<min:
            min=lista[x]
    print("El valor máximo de la lista es", max)
    print("El valor mínimo de la lista es", min)

# Bloque principal
lista=[]
for x in range(5):
    valor=int(input("Ingrese un valor: "))
    lista.append(valor)
max_min_lista(lista)

```

Guardamos el programa con el nombre de Eejrcicio126.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio126.py =============
Ingrese un valor: 10
Ingrese un valor: 20
Ingrese un valor: 5
Ingrese un valor: 100
Ingrese un valor: 56
El valor máximo de la lista es 100
El valor mínimo de la lista es 5
>>>
```

Capítulo 127.- Funciones: parámetros de tipo lista – 3

Problema propuesto:

Crear una lista de enteros por asignación. Definir una función que reciba la lista de enteros y un segundo parámetro de tipo entero. Dentro de la función mostrar cada elemento de la lista multiplicado por el valor entero enviado.
lista=[3, 7, 8, 10, 2]
multiplicar(lista, 3)

Vamos a programar:

```
*ejercicio127.py - D:/Curso completo de Python/ejercicio127.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
def multiplicar(lista, num):
    for x in range(len(lista)):
        multi=lista[x]*num
        print(multi)

# Bloque principal
lista=[3,7,8,10, 2]
print("Lista original")
print(lista)
print("Lista de los elementos multiplicados por 3")
multiplicar(lista, 3)
```

Guardamos el programa con el nombre de Ejercicios127.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio127.py =============
Lista original
[3, 7, 8, 10, 2]
Lista de los elementos multiplicados por 3
9
21
24
30
6
>>>
```

Ejecutamos de nuevo, pero multiplicando por 10:

```
*ejercicio127.py - D:/Curso completo de Python/ejercicio127.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
def multiplicar(lista, num):
    for x in range(len(lista)):
        multi=lista[x]*num
        print(multi)

# Bloque principal
lista=[3,7,8,10, 2]
print("Lista original")
print(lista)
print("Lista de los elementos multiplicados por 10")
multiplicar(lista, 10)
```

Este será el resultado:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio127.py =============
Lista original
[3, 7, 8, 10, 2]
Lista de los elementos multiplicados por 10
30
70
80
100
20
>>>
```

Capítulo 128.- Funciones: parámetros de tipo lista – 4

Problema propuesto:

Desarrollar una función que reciba una lista de string y nos retorne el que tiene más caracteres. Si hay más de uno con dicha cantidad de caracteres debe retornar el que tiene un valor de componente más baja.
En el bloque principal iniciamos por asignación la lista de string:
palabras=["enero", "febrero", "marzo", "abril", "mayo", "junio"]
print("Palabra con más caracteres: ", mascaracteres(palabras))

Vamos a programar:

ejercicio128.py - D:/Curso completo de Python/ejercicio128.py (3.11.1)

File Edit Format Run Options Window Help

```
def mascaracteres(lista):
    pos=0
    for x in range(1, len(lista)):
        if len(lista[x])>len(lista[pos]):
            pos=x
    return lista[pos]

# Bloque principal
palabras=["enero", "febrero", "marzo", "abril", "mayo", "junio"]
print(palabras)
print("Palabra con más caracteres: ", mascaracteres(palabras))

```

Guardamos el programa con el nombre de Ejercicio128.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio128.py =============
['enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio']
Palabra con más caracteres:  febrero
>>>
```

Capítulo 129.- Funciones: parámetros de tipo lista – 5

Problema propuesto:

Definir una lista de enteros por asignación en el bloque principal. Llamar a una función que reciba la lista y nos retorne el producto de todos sus elementos. Mostrar dicho producto en el bloque principal de nuestro programa.

Vamos a programar:

```
ejercicio129.py - D:/Curso completo de Python/ejercicio129.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1  def producto_lista(lista):
2      producto=1
3      for x in range(len(lista)):
4          producto=producto*lista[x]
5      return producto
6
7
8  # Bloque principal
9  lista=[1, 2, 3, 4, 5]
10 print(lista)
11 print("La multiplicación de todos sus elementos", producto_lista(lista))
12
13
```

Guardamos el programa con el nombre de Ejercicio129.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio129.py =============
[1, 2, 3, 4, 5]
La multiplicación de todos sus elementos 120
>>>
```

Capítulo 130.- Funciones: retorno de una lista - 1

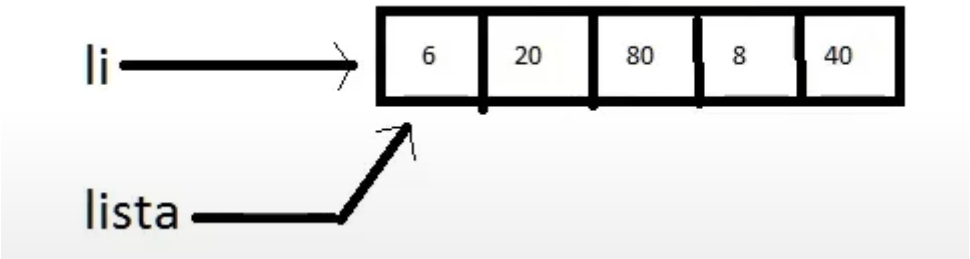
Hemos avanzado y visto que una función puede recibir como parámetros tipos de datos simples como enteros y flotantes etc. y estructura de datos tipo lista.

También hemos visto que la función mediante la palabra clave return puede retornar un tipo de dato simple desde donde se la invocó.

Lo nuevo es este concepto es que una función puede retornar una estructura de datos tipo list. Con esto estamos logrando que una función retorne varios datos ya que una lista es una colección de datos.

Problema:

Confeccionar una función que cargue por teclado una lista de 5 enteros y la retorne. Una segunda función debe recibir una lista y mostrar todos los valores mayores a 10. Desde el bloque principal del programa llamar a ambas funciones.



Vamos a programar:

ejercicio130.py - D:/Curso completo de Python/ejercicio130.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar_lista():
    lista=[]
    for x in range(5):
        num=int(input("Ingrese un valor: "))
        lista.append(num)
    return lista

def mostrar_lista(lista):
    print("Elementos de la lista mayores a 10")
    for x in range(5):
        if lista[x]>10:
            print(lista[x])

# Bloque principal
lis=cargar_lista()
print(lis)
mostrar_lista(lis)
```

Guardar el programa con el nombre de Ejercicio130.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio130.py =============
Ingrese un valor: 5
Ingrese un valor: 50
Ingrese un valor: 6
Ingrese un valor: 80
Ingrese un valor: 120
[5, 50, 6, 80, 120]
Elementos de la lista mayores a 10
50
80
120
>>>
```

Capítulo 131.- Funciones: retorno de una lista – 2

Problema:

Confeccionar una función que cargue por teclado una lista de 5 enteros y la retorne. Una segunda función debe recibir una lista y retornar el mayor y menor valor de la lista. Desde el bloque principal del programa llamar a ambas funciones e imprimir el mayor y el menor de la lista.

Vamos a programar:

ejercicio131.py - D:/Curso completo de Python/ejercicio131.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar_lista():
    lista=[]
    for x in range(5):
        valor=int(input("Ingrese un valor: "))
        lista.append(valor)
    return lista

def max_min_lista(lista):
    max=lista[0]
    min=lista[0]
    for x in range(1,5):
        if lista[x]>max:
            max=lista[x]
        if lista[x]<min:
            min=lista[x]
    return [max, min]

# Bloque principal
lis=cargar_lista()
valores=max_min_lista(lis)
print("El valor más grande es", valores[0])
print("El valor más pequeño es", valores[1])
```

Guardamos el programa con el nombre de Ejercicio131.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio131.py =============
Ingrese un valor: 20
Ingrese un valor: 1
Ingrese un valor: 100
Ingrese un valor: 50
Ingrese un valor: 33
El valor más grande es 100
El valor más pequeño es 1
>>>
```

Capítulo 132.- Funciones: retorno de una lista – 3

Problema:

Desarrollar un programa que permita cargar 5 nombres de personas y sus edades respectivas. Luego de realizar la carga por teclado de todos los datos imprimir los nombres de las personas mayores de edad (mayores o iguales a 18 años)

imprimir la edad promedio de las personas.

Vamos a programar:

ejercicio132.py - D:/Curso completo de Python/ejercicio132.py (3.11.1)

File Edit Format Run Options Window Help

```
def crear_lista():
    nombres_edades=[]
    for x in range(5):
        nom=input("Ingrese el nombre: ")
        edad=int(input("Ingrese la edad: "))
        nombres_edades.append([nom, edad])
    return nombres_edades


# Bloque principal
lista_nom=crear_lista()
suma=0
for x in range(len(lista_nom)):
    suma=suma + lista_nom[x][1]
    if lista_nom[x][1]>=18:
        print(lista_nom[x][0])
promedio=suma/(len(lista_nom))
print("Edad promedio de las personas", promedio)

```

Guardar el programa con el nombre de Ejercicio132.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio132.py =============
Ingrese el nombre: Juan
Ingrese la edad: 11
Ingrese el nombre: Ana
Ingrese la edad: 33
Ingrese el nombre: Carlos
Ingrese la edad: 66
Ingrese el nombre: Paula
Ingrese la edad: 67
Ingrese el nombre: Laura
Ingrese la edad: 2
Ana
Carlos
Paula
Edad promedio de las personas 35.8
>>>
```

Si el promedio lo queremos sin decimales en lugar de utilizar la / utilizaremos // de división entera.

Capítulo 133.- Funciones: retorno de una lista – 4

Problema propuesto:

En una empresa se almacenaron los sueldos de 10 personas.
Desarrollar la siguientes funciones y llamarlas desde el bloque principal:
1.- Carga de sueldos en una lista.
2.- Impresión de todos los sueldos.
3.- Cuantos tienen un sueldo superior a 4000.
4.- Retorna el promedio de los sueldos.
5.- Mostrar todos los sueldos que están por debajo del promedio.

Vamos a programar:

ejercicio133.py - D:/Curso completo de Python/ejercicio133.py (3.11.1)

File Edit Format Run Options Window Help

```
    sueldos=[]
    for x in range(10):
        sueldo=int(input("Ingrese el sueldo: "))
        sueldos.append(sueldo)
    return sueldos

def imprimir_lista(suel):
    for x in range(len(suel)):
        print(suel[x])

def sueldo_mayor_4000(suel):
    for x in range(len(suel)):
        if suel[x]>4000:
            print(suel[x])

def promedio_sueldo(suel):
    suma=0
    for x in range(len(suel)):
        suma=suma+suel[x]
    promedio=suma/(len(suel))
    return promedio

def sueldo_inferior_promedio(suel):
    for x in range(len(suel)):
        if suel[x]<promedio_sueldo(suel):
            print(suel[x])



# Bloque principal
lista_sueldos=cargar_lista()
imprimir_lista(lista_sueldos)
print("Los sueldos mayores de 4000")
sueldo_mayor_4000(lista_sueldos)
print("El promedio de los sueldos es", promedio_sueldo(lista_sueldos))
print("Los sueldos menores del promedio")
sueldo_inferior_promedio(lista_sueldos)

```

Guardar el programa con el nombre de Ejercicio133.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio133.py =============
Ingrese el sueldo: 1000
Ingrese el sueldo: 2000
Ingrese el sueldo: 3000
Ingrese el sueldo: 4000
Ingrese el sueldo: 5000
Ingrese el sueldo: 6000
Ingrese el sueldo: 7000
Ingrese el sueldo: 7000
Ingrese el sueldo: 7000
Ingrese el sueldo: 7000
1000
2000
3000
4000
5000
6000
7000
7000
7000
7000
Los sueldos mayores de 4000
5000
6000
7000
7000
7000
7000
El promedio de los sueldos es 4900.0
Los sueldos menores del promedio
1000
2000
3000
4000
>>>
Ln: 34  Col: 0
```

Capítulo 134.- Funciones: retorno de una lista – 5

Problema propuesto:

Desarrollar una aplicación que permita ingresar por teclado los nombres de 5 artículos y sus precios.

Definir las siguientes funcione:

1.- Cargar los nombres de artículos y sus precios.

2.- Imprimir nombres y precios.

3.- Imprimir el nombre del artículo con un precio mayor.

4.- Ingresar por teclado un importe y luego mostrar todos los artículos con un precio menor o igual al valor ingresado.

Vamos a programar:

ejercicio134.py - D:/Curso completo de Python/ejercicio134.py (3.11.1)

File Edit Format Run Options Window Help

```
def lista_articulos_precios():
    lista=[]
    for x in range(5):
        articulo=input("Ingrese el artículo: ")
        precio=float(input("Ingrese el precio: "))
        lista.append([articulo, precio])
    return lista

def imprimir_articulos_precios(lis):
    for x in range(len(lis)):
        print(lis[x][0], lis[x][1])

def articulo_mayor(lis):
    pos=0
    max=lis[0][1]
    for x in range(1, len(lis)):
        if lis[x][1]>max:
            pos=x
    return [lis[pos][0], lis[pos][1]]

def precio_inferior(lis):
    precio=float(input("Ingrese un precio: "))
    for x in range(len(lis)):
        if lis[x][1]<=precio:
            print(lis[x][0], lis[x][1])


# Bloque principal
lista1=lista_articulos_precios()
imprimir_articulos_precios(lista1)
articulo=articulo_mayor(lista1)
print("El artículo con un precio superior es", articulo[0], "a", articulo[1])
print("Artículos a un precio menor al valor a consultar: ")
precio_inferior(lista1)
```

En este ejemplo hemos creado una lista y a su vez esta contiene otras listas con el nombre del artículo y el precio, otra forma de realizarlo es creando 2 listas paralelas.

```
def cargar_datos():
    articulos=[]
    precios=[]
    for x in range(5):
        ar=input("Ingrese el nombre del articulo:")
        articulos.append(ar)
        pre=int(input("Ingrese el precio de dicho articulo:"))
        precios.append(pre)
    return [articulos, precios]
```

A continuación las demás funciones tendrán como argumentos las dos listas.

```
def imprimir(articulos,precios):
    print("Listado completo de articulos y precios")
    for x in range(len(articulos)):
        print(articulos[x], precios[x])

def precio_mayor(articulos,precios):
    may=precios[0]
    pos=0
    for x in range(1,len(precios)):
        if precios[x]>may:
            may=precios[x]
            pos=x
    print("Articulo con un precio mayor es :", articulos[pos],"su precio es",may)

def consulta_precio(articulos,precios):
    valor=int(input("Ingrese un importe para mostrar los articulos con un precio menor:"))
    for x in range(len(precios)):
        if precios[x]<=valor:
            print(articulos[x],precios[x])
```

Este será el bloque principal:

```
# bloque principal
articulos,precios=cargar_datos()
imprimir(articulos,precios)
precio_mayor(articulos,precios)
consulta_precio(articulos,precios)
```

Cuando una función retorna más de un valor se ponen las variables separadas por comas.

Guardamos el programa con el nombre de Ejercicio134.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio134.py =============
    Ingrese el artículo: Patatas
    Ingrese el precio: 40
    Ingrese el artículo: Manzanas
    Ingrese el precio: 60
    Ingrese el artículo: Mandarinas
    Ingrese el precio: 50
    Ingrese el artículo: Lechuga
    Ingrese el precio: 80
    Ingrese el artículo: Limones
    Ingrese el precio: 20
    Patatas 40.0
    Manzanas 60.0
    Mandarinas 50.0
    Lechuga 80.0
    Limones 20.0
    El artículo con un precio superior es Lechuga a 80.0
    Artículos a un precio menor al valor a consultar:
    Ingrese un precio: 50
    Patatas 40.0
    Mandarinas 50.0
    Limones 20.0
>>>
```

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio134.py =============
Ingrese el artículo: Patatas
Ingrese el precio: 40
Ingrese el artículo: Manzanas
Ingrese el precio: 60
Ingrese el artículo: Mandarinas
Ingrese el precio: 50
Ingrese el artículo: Lechuga
Ingrese el precio: 80
Ingrese el artículo: Limones
Ingrese el precio: 20
Patatas 40.0
Manzanas 60.0
Mandarinas 50.0
Lechuga 80.0
Limones 20.0
El artículo con un precio superior es Lechuga a 80.0
Artículos a un precio menor al valor a consultar:
Ingrese un precio: 50
Patatas 40.0
Mandarinas 50.0
Limones 20.0
>>>
```

Capítulo 135.- Funciones: retorno de una lista – 6

Problema propuesto:

Confeccionar un programa que permita:
1.- Cargar una lista de 10 elementos enteros.
2.- Generar dos listas a partir de la primera. En una guardar los valore positivos y en la otra los valores negativos.
3.- Imprimir las dos listas generadas.

Vamos a programar:

ejercicio135.py - D:/Curso completo de Python/ejercicio135.py (3.11.1)

File Edit Format Run Options Window Help

```
def crear_lista():
    lista=[]
    for x in range(10):
        num=int(input("Ingrese un número positivo o negativo: "))
        lista.append(num)
    return lista

def generar_dos_listas(lis):
    positivos=[]
    negativos=[]
    for x in range(len(lis)):
        if lis[x]>=0:
            positivos.append(lis[x])
        else:
            negativos.append(lis[x])
    return [positivos, negativos]

def imprimir_listas(lispos, lisneg):
    print("Valores positivos y el 0")
    for x in range(len(lispos)):
        print(lispos[x])
    print("Valores negativos")
    for x in range(len(lisneg)):
        print(lisneg[x])

# Bloque principal
numeros=crear_lista()
positivos, negativos=generar_dos_listas(numeros)
imprimir_listas(positivos, negativos)
```

Guardar el programa con el nombre de Ejercicio135.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio135.py =============
Ingrese un número positivo o negativo: -5
Ingrese un número positivo o negativo: -4
Ingrese un número positivo o negativo: -3
Ingrese un número positivo o negativo: 1
Ingrese un número positivo o negativo: 2
Ingrese un número positivo o negativo: 3
Ingrese un número positivo o negativo: 4
Ingrese un número positivo o negativo: 5
Ingrese un número positivo o negativo: 6
Ingrese un número positivo o negativo: 7
```

```
    Valores positivos y el 0
    1
    2
    3
    4
    5
    6
    7
    Valores negativos
    -5
    -4
    -3
>>>
```

Capítulo 136.- Funciones: parámetros con valor por defecto – 1

En Python se pueden definir parámetros y asignarle un dato en la misma cabecera de la función. Luego cuando llamamos a la función podemos o no enviarle un valor al parámetro.

Los parámetros por defecto nos permite crear funciones más flexibles y que se puedan emplean en distintas circunstancias.

Problema:

Confeccionar una función que reciba un string como parámetro y en forma opcional un segundo string con un carácter. La función debe mostrar el string subrayando con el carácter que indica el segundo parámetro.

Vamos a programar:

ejercicio136.py - D:/Curso completo de Python/ejercicio136.py (3.11.1)

File Edit Format Run Options Window Help

```
def titulo_subrayado(titulo, caracter="*"):
    print(titulo)
    print(caracter*len(titulo))

# Bloque principal
titulo_subrayado("Sistema de administración")
titulo_subrayado("Sistema de ventas", "_")

```

Guardamos el programa con el nombre de Ejercicio136.py y a continuación lo ejecutamos.

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio136.py =============
Sistema de administración
*************************
Sistema de ventas
_________________
>>>
```

De este modo si llamamos a la función con solo el primer parámetro que es el titulo del texto el subrayado por defecto será con el '*', en cambio si llamamos a la función agregando además un segundo carácter para el subrayado este es que se va a mostrar.

Los parámetro con valor por defecto tienen que ir en las últimas posiciones en la función.

Capítulo 137.- Funciones. parámetros con valor por defecto – 2

Problema propuesto:

Confeccionar una función que reciba entre 2 y 5 enteros. La misma nos debe retornar la suma de dichos valores. Debe tener dos parámetros por defecto:

Vamos a programar:

```
ejercicio137.py - D:/Curso completo de Python/ejercicio137.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1  def valores(v1, v2, v3=0, v4=0, v5=0):
2      suma=v1+v2+v3+v4+v5
3      return suma
4
5  # Bloque principal
6  print("La suma de 5 + 6 =", valores(5,6))
7  print("La suma de 5 + 6 + 3 =", valores(5,6,3))
8  print("La suma de 5 + 6 + 3 + 5 =", valores(5,6,3,5))
9  print("La suma de 5 + 6 + 3 + 5 + 8 =", valores(5,6,3,5,8))
10
```

Guardamos el programa con el nombre de Ejercicio137.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio137.py =============
    La suma de 5 + 6 = 11
    La suma de 5 + 6 + 3 = 14
    La suma de 5 + 6 + 3 + 5 = 19
    La suma de 5 + 6 + 3 + 5 + 8 = 27
>>>
```

De este modo los argumentos puede ser 2, 3, 4 o 5 valores.

Capítulo 138.- Funciones: llamada a la función con argumentos nombrados – 1

Esta característica de Python nos permite llamar a la función indicando en cualquier orden los parámetros de la misma, pero debemos especificar en la llamada el nombre del parámetro y el valor a enviarle.

Problema:

Confeccionar una función que llegue como primer parámetro el nombre del empleado, como segundo parámetro el coste por hora y finalmente la cantidad de horas trabajadas.

Llamar a dicha función empleando la característica de Python de argumentos nombrados.

Vamos a programar:

```
ejercicio138.py - D:/Curso completo de Python/ejercicio138.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 def calcular_sueldo(nombre, preciohora, cantidadhoras):
2     sueldo=preciohora*cantidadhoras
3     print(nombre, "trabajo", cantidadhoras, "y cobra un sueldo de", sueldo)
4
5 # Bloque principal
6 calcular_sueldo("Ana", 10, 100)
7 calcular_sueldo(cantidadhoras=50, nombre="Juan", preciohora=45)
8
```

Guardamos el programa con el nombre de Ejercicio138.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio138.py =============
    Ana trabajo 100 y cobra un sueldo de 1000
    Juan trabajo 50 y cobra un sueldo de 2250
>>>
```

Cuando ejecutamos este programa:

```
print("hola")          hola
print("mundo")         mundo
```

Esto es debido a que hay un parámetro llamada end que por defecto tiene el salto de línea \n.

```
print("hola", end="\n")   hola
print("mundo")            mundo
```

Como podrás observar el resultado es el mismo, vamos a cambiar el parámetro end.

```
print("hola", end="-")    hola-mundo
print("mundo")
```

Vamos a ver el siguiente ejemplo:

```
for x in range(10):
    print(x)
```

Este será el resultado:

```
0
1
2
3
4
5
6
7
8
9
```

```
for x in range(10):
    print(x, end="-")
```

Modificamos el parámetro end y este será el resultado:

```
0-1-2-3-4-5-6-7-8-9-
```

```
for x in range(10):
    print(x, end="-")
print("fin del programa")
```

Si añadimos una nueva línea con otro mensaje este será el resultado:

```
0-1-2-3-4-5-6-7-8-9-fin del programa
```

Para que imprima los números separados por guion y la línea fin de programa en la siguiente línea realizaremos lo siguiente:

```
for x in range(10):
    if x==9:
        print(x)
    else:
        print(x, end="-")
print("fin del programa")
```

Este será el resultado:

```
0-1-2-3-4-5-6-7-8-9
fin del programa
```

Capítulo 139.- Funciones: llamada a la función con argumentos nombrados – 2

Problema:

Cargar una lista de 10 enteros, luego mostrarlos por pantalla a cada elemento separado por una coma en la misma línea.
Vamos a programar:

ejercicio139.py - D:/Curso completo de Python/ejercicio139.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar():
    lista=[]
    for x in range(10):
        valor=int(input("Ingrese un valor: "))
        lista.append(valor)
    return lista
def imprimir(lista):
    for x in range(10):
        if x==9:
            print(lista[x])
        else:
            print(lista[x], end=",")
    print("Fin de programa")

# Bloque principal
lista=cargar()
imprimir(lista)
```

Guardamos el programa con el nombre de Ejercicio139.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio139.py =============
Ingrese un valor: 10
Ingrese un valor: 20
Ingrese un valor: 30
Ingrese un valor: 40
Ingrese un valor: 50
Ingrese un valor: 60
Ingrese un valor: 70
Ingrese un valor: 80
Ingrese un valor: 90
Ingrese un valor: 100
10,20,30,40,50,60,70,80,90,100
Fin de programa
>>>
```

Vamos a comentar otro argumento que tiene el print.

```
print("uno", "dos", "tres")            uno dos tres
```

Es lo mismo que poner:

```
print("uno", "dos", "tres", sep=" ")   uno dos tres
```

El argumento sep por defecto tiene un espacio en blanco.

```
print("uno", "dos", "tres", sep=",")   uno,dos,tres
```

Ahora lo separamos por comas.
Podemos combinar los dos argumentos sep y end.

Capítulo 140.- Funciones: llamada a la función con argumentos nombrados – 3

Problema propuesto:

Elaborar una función que muestre la tabla de multiplicar del valor que le enviemos como parámetro. Definir un segundo parámetro llamado termino que por defecto almacene el valor 10. Se deben mostrar tantos términos de la tabla de multiplicar como lo indican el segundo parámetro.
Llamar a la función desde el bloque principal de nuestro programa con argumentos nombrados.
Vamos a programar:

ejercicio140.py - D:/Curso completo de Python/ejercicio140.py (3.11.1)

File Edit Format Run Options Window Help

```
1  def tabla_multiplicar(tabla, termino=10):
2      for x in range(termino+1):
3          print(tabla, "x", x, "=", end=" ")
4          print(tabla*x)
5
6
7  # Bloque principal
8  print("Tabla de multiplicar del 5 con un termino de 12")
9  tabla_multiplicar(termino=12, tabla=5)
10 print("Tabla de multiplicar del 8 con un termino por defecto del 10")
11 tabla_multiplicar(8)
12
```

Guardar el programa con el nombre de Ejercicio140.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio140.py =============
Tabla de multiplicar del 5 con un termino de 12
5 x 0 = 0
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
5 x 11 = 55
5 x 12 = 60
Tabla de multiplicar del 8 con un termino por defecto del 10
8 x 0 = 0
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
8 x 4 = 32
8 x 5 = 40
8 x 6 = 48
8 x 7 = 56
8 x 8 = 64
8 x 9 = 72
8 x 10 = 80
```

Capítulo 141.- Funciones: con cantidad de variables de parámetro – 1

Otra variante en la declaración de una función en Python es la definición de una cantidad de variable de parámetros.

Para definir una cantidad variable de parámetros debemos anteceder el carácter (*) al último parámetro de la función.

Problema:

Confeccionar una función que reciba entre 2 y n (siendo n=(2,3,4,5,6, etc.) valores enteros, retorna la suma de dichos parámetros.

Vamos a programar:

ejercicio141.py - D:/Curso completo de Python/ejercicio141.py (3.11.1)

File Edit Format Run Options Window Help

```
def sumar(v1, v2, *lista):
    suma=v1+v2
    for x in range(len(lista)):
        suma=suma+lista[x]
    return suma

# Bloque principal
print(sumar(1,2))
print(sumar(1,2,3,4,5))
```

Guardamos el programa con el nombre de Ejercicio141.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio141.py =============
3
15
>>>
```

Capítulo 142.- Funciones: con cantidad variables de parámetros – desempaquetado de listas – 2

Puede ser que tengamos una función que recibe una cantidad fija de parámetros y necesitamos llamarla enviando valores que se encuentran en una lista. La forma más sencilla es anteceder el carácter * al nombre de la variable.

Problema:

Confeccionar una función que reciba 3 enteros y retorne la suma. Llamar a la función enviando 3 enteros que se encuentran almacenados en una lista.

Vamos a programar:

```
ejercicio142.py - D:/Curso completo de Python/ejercicio142.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 def sumar(v1, v2, v3):
2     return v1+v2+v3
3
4 # Bloque principal
5 lista=[1,2,3]
6 print(sumar(lista[0], lista[1], lista[2]))
7
```

Guardamos el programa con el nombre de Ejercicio142.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
  Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
  AMD64)] on win32
  Type "help", "copyright", "credits" or "license()" for more information.
>>>
  ============= RESTART: D:/Curso completo de Python/ejercicio142.py =============
  6
>>>
```

Otra forma de realizar los mismo es:

```
ejercicio142.py - D:/Curso completo de Python/ejercicio142.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 def sumar(v1, v2, v3):
2     return v1+v2+v3
3
4 # Bloque principal
5 lista=[1,2,3]
6 print(sumar(*lista))
7
```

En la línea 6 donde llamamos a la función, para no tener que agregar elemento por elemento agregamos un * seguido del nombre de la lista. El * significa que le estamos enviando varios parámetros.

Capítulo 143.- Funciones: con cantidad variable de parámetro – 3

Problema propuesto:

Confeccionar una función que reciba una serie de edades y me retorne la cantidad que son mayores o iguales a 18 (como mínimo se envía un entero a la función).

Vamos a programar:

```
ejercicio143.py - D:/Curso completo de Python/ejercicio143.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
def mayor_edad(edad, *lista):
    cant=0
    if edad>=18:
        cant=cant+1
    for x in range(len(lista)):
        if lista[x]>=18:
            cant=cant+1
    return cant

# Bloque principal
edad=50
lista=[5, 18, 13, 22, 7, 45, 23, 12, 19]
print("Las edades superiores o iguales a 18 años")
print(mayor_edad(edad, *lista))

```

El primer parámetro hace obligatorio que al llamar a la función como mínimo le pasemos un parámetro obligatoriamente y los demás son opcionales.

Guardamos el programa con el nombre Ejercicio143.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio143.py =============
Las edades superiores o iguales a 18 años
6
>>>
```

Capítulo 144.- Estructura de datos tipo Tupla – 1

Hemos desarrollado gran cantidad de algoritmos empleando tipos de datos simples como enteros, flotantes, cadena de caracteres y estructura de datos tipo lista.
Vamos a ver otra estructura de datos llamada Tupla.
Una tupla permite almacenar una colección de datos no necesariamente del mismo tipo. Los datos de la tupla son inmutables a diferencia de las listas que son mutables.
Una vez inicializada la tupla no podemos agregar, borrar o modificar elementos.
Utilizaremos una tupla para agrupar datos que por su naturaleza están relacionados y que no serán modificados durante la ejecución del programa.
La sintaxis para definir una tupla es indicar entre paréntesis sus valores:

tupla=(1, 2, 3)
fecha=(25, "Diciembre", 2016)
punto=(10, 2)
persona=("Rodriguez", "Pablo", 43)

Problema:

Definir varias tuplas e imprimir sus elementos.

Vamos a programar:

ejercicio144.py - D:/Curso completo de Python/ejercicio144.py (3.11.1)

File Edit Format Run Options Window Help

```
1  tupla=(1, 2, 3)
2  fecha=(25, "Diciembre", 2016)
3  punto=(10, 2)
4  persona=("Rodriguez", "Pablo", 43)
5  print(tupla)
6  print(fecha)
7  print(punto)
8  print(persona)
9  print(punto[0], punto[1])
10 punto[0]=34
11
```

Guardamos el programa con el nombre de Ejercicio144.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio144.py =============
(1, 2, 3)
(25, 'Diciembre', 2016)
(10, 2)
('Rodriguez', 'Pablo', 43)
10 2
Traceback (most recent call last):
  File "D:/Curso completo de Python/ejercicio144.py", line 10, in <module>
    punto[0]=34
TypeError: 'tuple' object does not support item assignment
>>>
```

En la línea 10 queremos modificar el valor de un elemento y no nos lo permite.

Capítulo 145.- Estructura de datos tipo Tupla – 2

Problema:

Desarrollar una función que solicite la carga del día, mes y año y almacene dichos datos en una tupla que luego debe retornar. La segunda función a implementar debe recibir una tupla con la fecha y mostrar por pantalla.

Vamos a programar:

```
ejercicio145.py - D:/Curso completo de Python/ejercicio145.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
def cargar_fecha():
    dia=int(input("Ingrese el día: "))
    mes=int(input("Ingrese el mes: "))
    año=int(input("Ingrese el año: "))
    return (dia, mes, año)

def imprimir_fecha(fecha):
    print(fecha[0], fecha[1], fecha[2], sep="/")

# Bloque principal
fecha=cargar_fecha()
imprimir_fecha(fecha)
```

Guardamos el programa con el nombre de Ejercicio145.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio145.py =============
Ingrese el día: 1
Ingrese el mes: 5
Ingrese el año: 2020
1/5/2020
>>>
```

Para convertir una tupla en lista realizaremos los siguiente:

```
*ejercicio145.py - D:/Curso completo de Python/ejercicio145.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
def cargar_fecha():
    dia=int(input("Ingrese el día: "))
    mes=int(input("Ingrese el mes: "))
    año=int(input("Ingrese el año: "))
    return (dia, mes, año)

def imprimir_fecha(fecha):
    print(fecha[0], fecha[1], fecha[2], sep="/")

# Bloque principal
fecha=cargar_fecha()
imprimir_fecha(fecha)
print("Imprimir tupla")
print(fecha)
print("Convertir tupla a lista")
lista=list(fecha)
print("Imprimir lista")
print(lista)
```

Vamos a ejecutar de nuevo:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio145.py =============
Ingrese el día: 25
Ingrese el mes: 12
Ingrese el año: 2020
25/12/2020
Imprimir tupla
(25, 12, 2020)
Convertir tupla a lista
Imprimir lista
[25, 12, 2020]
>>>
```

Tupla

Lista

Con el sentencia list podemos convertir una tupla a lista:

lista=list(tupla).

En la lista si podemos modificar los elementos:

lista[0]=2

Para convertir una lista en tupla:

tupla2=tuple(lista)

Capítulo 146.- Estructura de datos tipo Tupla – 3

Problema:

Definir una tupla con tres valores enteros. Convertir el contenido de la tupla a tipo lista. Modificar la lista y luego convertir la lista en tupla.

Crear otra tupla a partir de tres variables independiente que almacena el día, mes y año (empaquetamiento), luego desempaquetar la tupla a otras tres variables independientes.

Vamos a programar:

ejercicio146.py - D:/Curso completo de Python/ejercicio146.py (3.11.1)

File Edit Format Run Options Window Help

```
1  fecha_tupla=(1,5,2020)
2  print("Imprimimos la tupla")
3  print(fecha_tupla)
4  fecha_list=list(fecha_tupla)
5  print("Imprimimos la lista que se copio de la tupla")
6  print(fecha_list)
7  fecha_list[0]=25
8  print(fecha_list)
9  fecha_tupla2=tuple(fecha_list)
10 print("Imprimir la segunda tupla que se copio de la lista")
11 print(fecha_tupla2)
12 # Empaquetamiento
13 dia=10
14 mes=5
15 año=2020
16 fecha_tupla3=dia, mes, año
17 print(fecha_tupla3)
18 # Desempaquetar
19 dia1, mes1, año1=fecha_tupla3
20 print(dia1, mes1, año1)
21
22
```

Guardamos el programa con el nombre de Ejercicio146.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio146.py =============
Imprimimos la tupla
(1, 5, 2020)
Imprimimos la lista que se copio de la tupla
[1, 5, 2020]
[25, 5, 2020]
Imprimir la segunda tupla que se copio de la lista
(25, 5, 2020)
(10, 5, 2020)
10 5 2020
>>>
```

Capítulo 147.- Estructura de datos tipo Tupla – 4

Problema propuesto:

Confeccionar un programa con las siguientes funciones:
1.- Cargar una lista de 5 enteros.
2.- Retornar el mayor y el menor valor de la lista mediante una tupla.
Desempaquetar la tupla en el bloque principal y mostrar el mayor y menor.

Vamos a programar:

ejercicio147.py - D:/Curso completo de Python/ejercicio147.py (3.11.1)

File Edit Format Run Options Window Help

```
1  def mayor_menor():
2      lista=[1,2,3,4,5]
3      max=lista[0]
4      min=lista[0]
5      for x in range(1, len(lista)):
6          if lista[x]>max:
7              max=lista[x]
8          else:
9              if lista[x]<min:
10                 min=lista[x]
11     return (max, min)
12 # Bloque principal
13 mayor, menor=mayor_menor()
14 print("El valor máximo de la lista es", mayor)
15 print("El valor mínimo de la lista es", menor)
16
```

Empaquetar

Desempaquetar

Guardamos el programa con el nombre de Ejercicio147.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio147.py =============
El valor máximo de la lista es 5
El valor mínimo de la lista es 1
>>>
```

Capítulo 148.- Estructura de datos tipo Tupla – 5

Problema propuesto:

Confeccionar un programa con las siguientes funciones:
1.- Cargar el nombre de un empleado y su sueldo. Retornar una tupla con dichos valores.
2.- Una función que reciba como parámetro dos tuplas con los nombres y sueldos de empleados y muestre el nombre del empleado con sueldo mayor.
En el bloque principal del programa llamar dos veces a la función de carga y seguidamente llamar a la función que muestra el nombre de empleado con sueldo mayor.

Bloque principal

empleado1=cargar_empleado()
empleado2=cargar_empleado()
mayor_sueldo(empleado1, empleado2)

Vamos a programar:

ejercicio148.py - D:/Curso completo de Python/ejercicio148.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar_empleado():
    empleado=input("Ingrese el nombre del empleado: ")
    sueldo=int(input("Ingrese el sueldo del empleado: "))
    return(empleado, sueldo)

def mayor_sueldo(tupla1, tupla2):
    if tupla1[1]==tupla2[1]:
        print("Los dos empleados tienen el mismo sueldo")
    else:
        if tupla1[1]>tupla2[1]:
            print("El empleado con mayor sueldo es",tupla1[0])
        else:
            print("El empleado con mayor sueldo es",tupla2[0])

# Bloque principal
empleado1=cargar_empleado()
empleado2=cargar_empleado()
mayor_sueldo(empleado1, empleado2)
```

Guardar el programa con el nombre de Ejercicio148.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio148.py =============
Ingrese el nombre del empleado: Carlos
Ingrese el sueldo del empleado: 1500
Ingrese el nombre del empleado: Luis
Ingrese el sueldo del empleado: 1500
Los dos empleados tienen el mismo sueldo
>>>
============= RESTART: D:/Curso completo de Python/ejercicio148.py =============
Ingrese el nombre del empleado: Carlos
Ingrese el sueldo del empleado: 1500
Ingrese el nombre del empleado: Luis
Ingrese el sueldo del empleado: 1200
El empleado con mayor sueldo es Carlos
```

```
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio148.py =============
    Ingrese el nombre del empleado: Carlos
    Ingrese el sueldo del empleado: 1200
    Ingrese el nombre del empleado: Luis
    Ingrese el sueldo del empleado: 1500
    El empleado con mayor sueldo es Luis
>>>
```

Capítulo 149.- Listas y tuplas anidadas – 1

Hemos visto dos estructuras de datos fundamentales en Python que son las listas y las tuplas.

La lista es una estructura mutable (es decir podemos modificar sus elementos, agregar, borrar) en cambio un tupla es una secuencia de datos inmutable, es decir una vez definida no se puede cambiar.

En Python vino que podemos definir elementos de una lista que sean de tipo lista, es ese caso decimos que tenemos una lista anidada.

Ahora que vimos tuplas también podemos crear tuplas anidadas.

En general podemos crear y combinar tuplas con elementos de tipo lista y viceversa, es decir listas con componentes tipo tupla.

Problema:

Definir una serie de listas y tuplas anidadas.

Vamos a programar:

ejercicio149.py - D:/Curso completo de Python/ejercicio149.py (3.11.1)

File Edit Format Run Options Window Help

```
1  empleado=["Ana", 45, (1,1,2020)]
2  print(empleado)
3  empleado.append((1,5,2020))
4  print(empleado)
5
6  alumno=("Juan", [8,5])
7  print(alumno)
8
9  alumno[1].append(10)
10 print(alumno)
11
12
```

Guardamos el programa con el nombre de Ejercicio149.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio149.py =============
['Ana', 45, (1, 1, 2020)]
['Ana', 45, (1, 1, 2020), (1, 5, 2020)]
('Juan', [8, 5])
('Juan', [8, 5, 10])
>>>
```

Capítulo 150.- Listas y tuplas anidadas – 2

Problema:

Almacenar en una lista 5 elementos tuplas que guarden el nombre de un país y la cantidad de habitantes.

Definir tres funciones, en la primera cargar la lista, en la segunda imprimirla y en la tercera mostrar el nombre del país con mayor cantidad de habitantes.

Vamos a programar:

ejercicio150.py - D:/Curso completo de Python/ejercicio150.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar_lista():
    paises=[]
    for x in range(5):
        pais=input("Ingrese el pais: ")
        habitantes=int(input("Ingrese el número de habitantes en millones: "))
        paises.append((pais, habitantes))
    return paises

def imprimir_lista(lista):
    print("Listado de paises")
    for x in range(len(lista)):
        print(lista[x][0], lista[x][1])

def pais_mayor_habitantes(lista):
    mayor=lista[0][1]
    pos=0
    for x in range(1, len(lista)):
        if lista[x][1]>mayor:
            pos=x
    print("El pais con mas habitantes es:", lista[pos][0], "con", lista[pos][1])

# Bloque principal
paises_habitantes=cargar_lista()
imprimir_lista(paises_habitantes)
pais_mayor_habitantes(paises_habitantes)
```

Guardamos el programa con el nombre de Ejercicio150.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio150.py =============
Ingrese el pais: Chile
Ingrese el número de habitantes en millones: 19
Ingrese el pais: Argentina
Ingrese el número de habitantes en millones: 40
Ingrese el pais: Uruguay
Ingrese el número de habitantes en millones: 9
Ingrese el pais: Brasil
Ingrese el número de habitantes en millones: 170
Ingrese el pais: Paraguay
Ingrese el número de habitantes en millones: 3
Listado de paises
Chile 19
Argentina 40
Uruguay 9
Brasil 170
Paraguay 3
El pais con mas habitantes es: Brasil con 170
>>>
```

Capítulo 151.- Listas y tuplas anidadas – 3

Problema propuesto:

Almacenar en una lista 5 empleados, cada elemento es una lista con el nombre del empleado junto a sus últimos tres sueldos (estos tres valores en una tupla)

El programa debe tener las siguientes funciones:

1.- Cargar los nombres de los empleados y sus últimos tres sueldos.
2.- Imprimir el total cobrado por cada empleado.
3.- Imprimir los nombres de empleados que tuvieron un ingreso trimestral mayor a 10000 en los últimos tres meses.
Tener en cuenta que la estructura de datos si se carga por asignación debería ser similar a:

empleados = [["Juan", (2000,3000,4233)], ["Ana", (3444, 1000, 53333)]m etc.]

Vamos a programar:

ejercicio151.py - D:/Curso completo de Python/ejercicio151.py (3.11.1)

File Edit Format Run Options Window Help

```python
def cargar_lista():
    empleados=[]
    for x in range(5):
        nombre=input("Ingrese el nombre del empleado: ")
        sueldo1=int(input("Ingrese el primer sueldo: "))
        sueldo2=int(input("Ingrese el segundo sueldo: "))
        sueldo3=int(input("Ingrese el tercer sueldo: "))
        empleados.append([nombre, (sueldo1, sueldo2, sueldo3)])
    return empleados

def total_cobrado(lista):
    for x in range(5):
        total=lista[x][1][0]+lista[x][1][1]+lista[x][1][2]
        print("El sueldo total de", lista[x][0], "es de", total)

def ingreso_mayor_10000(lista):
    for x in range(5):
        total=lista[x][1][0]+lista[x][1][1]+lista[x][1][2]
        if total>10000:
            print("Los emplados que cobran más de 10000 trimestral es", lista[x][0])

# Bloque principal
empleados=cargar_lista()
total_cobrado(empleados)
ingreso_mayor_10000(empleados)
```

Guardar el programa con el nombre de Ejercicio151.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio151.py =============
Ingrese el nombre del empleado: Ana
Ingrese el primer sueldo: 1000
Ingrese el segundo sueldo: 1000
Ingrese el tercer sueldo: 1000
Ingrese el nombre del empleado: Juan
Ingrese el primer sueldo: 4000
Ingrese el segundo sueldo: 4000
Ingrese el tercer sueldo: 5000
```

```
Ingrese el nombre del empleado: Pedro
Ingrese el primer sueldo: 2000
Ingrese el segundo sueldo: 2999
Ingrese el tercer sueldo: 2000
Ingrese el nombre del empleado: Carlos
Ingrese el primer sueldo: 1000
Ingrese el segundo sueldo: 1000
Ingrese el tercer sueldo: 1000
Ingrese el nombre del empleado: Luis
Ingrese el primer sueldo: 5000
Ingrese el segundo sueldo: 5000
Ingrese el tercer sueldo: 5000
El sueldo total de Ana es de 3000
El sueldo total de Juan es de 13000
El sueldo total de Pedro es de 6999
El sueldo total de Carlos es de 3000
El sueldo total de Luis es de 15000
Los emplados que cobran más de 10000 trimestral es Juan
Los emplados que cobran más de 10000 trimestral es Luis
>>>
```

Capítulo 152.- Listas y tuplas anidadas – 4

Problema propuesto:

Se tiene que cargar los votos obtenidos por tres candidatos a unas elecciones.

En una lista cargar el primer componente del nombre del candidato y en la segundo componente cargar una lista con componentes de tipo tupla con el nombre de la provincia y la cantidad de votos obtenidos en dicha provincia.

Se deben cargar los datos por teclado, pero si se cargan por asignación tendrían una estructura similar a esta:

candidatos= [("Juan", [("Córdoba", 1000), ("Buenos Aires", 200)])

("Ana", [("Córdoba", 55)])

("Luis", [("Buenos Aires", 20)])]

1.- Función para cargar todos los candidatos, sus nombres y las provincias con los votos obtenidos.

2.- Imprimir el nombre del candidato y la cantidad total de votos obtenidos en todas las provincias.

Vamos a programar:

ejercicio152.py - D:/Curso completo de Python/ejercicio152.py (3.11.1)

File Edit Format Run Options Window Help

```
def candidatos():
    candidatos=[]
    cant=int(input("Cuantos candidatos se presentan: "))
    for x in range(cant):
        nombre=input("Ingrese el nombre del candidato: ")
        cantidad_ciudades=int(input("A cuantas ciudades tiene votos :"))
        lista_ciudad=[]
        for y in range(cantidad_ciudades):
            ciudad=input("Nombre de la ciudad: ")
            votos=int(input("Cuantos votos ha obtenido: "))
            lista_ciudad.append((ciudad, votos))
        candidatos.append((nombre,lista_ciudad))
    return candidatos

def contar_votos(lista):
    for x in range(len(lista)):
        suma_votos=0
        for y in range(len(lista[x][1])):
            suma_votos=suma_votos+(lista[x][1][y][1])
        print(lista[x][0], suma_votos)

# Bloque principal
votacion_candidatos=candidatos()
print(votacion_candidatos)
contar_votos(votacion_candidatos)
```

Guardamos el programa con el nombre de Ejercicio152.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio152.py =============
Cuantos candidatos se presentan: 3
Ingrese el nombre del candidato: Juan
A cuantas ciudades tiene votos :2
Nombre de la ciudad: Córdoba
Cuantos votos ha obtenido: 100
Nombre de la ciudad: Buenos Aires
Cuantos votos ha obtenido: 200
Ingrese el nombre del candidato: Ana
A cuantas ciudades tiene votos :1
Nombre de la ciudad: Córdoba
Cuantos votos ha obtenido: 55
Ingrese el nombre del candidato: Luis
A cuantas ciudades tiene votos :1
Nombre de la ciudad: Buenos Aires
Cuantos votos ha obtenido: 20
[('Juan', [('Córdoba', 100), ('Buenos Aires', 200)]), ('Ana', [('Córdoba', 55)])
, ('Luis', [('Buenos Aires', 20)])]
Juan 300
Ana 55
Luis 20
>>>
```

Capítulo 153.- Variantes de la estructura repetitiva for para recorrer tuplas y listas – 1

Hasta ahora siempre que recorremos una lista o una tupla utilizando un for y procedemos de la siguiente manera:

```
lista=[2, 3, 50, 7, 9]

for x in range(len(lista)):
    print(lista[x])
```

Esta forma de recorrer la lista es utilizada obligatoriamente cuando queremos modificar sus elementos como podría ser:

```
lista=[2, 3, 50, 7, 9]

print(lista)    # [2, 3, 50, 7, 9]

for x in range(len(lista)):
    if lista[x]<10:
        lista[x]=0

print(lista)    # [0, 0, 50, 0, 0]
```

Ahora veremos una segunda forma de acceder a los elementos de una lista con la estructura repetitiva fon sin indicar subíndices.

```
lista=[2, 3, 50, 7, 9]

for elemento in lista:
    print(elemento)
```

Como podemos ver la instrucción for requiere una variable (en este ejemplo llamada elemento), luego la palabra clave in y por último el nombre de la lista. El bloque del for se ejecutará tantas veces como elementos tenga la lista, y en vuelta del for la variable elemento almacenará un valor de la lista.

Problema:

Confeccionar un programa que permita la carga de una lista de 5 enteros por teclado. Luego en otras funciones:

1.- Imprima en forma completa.
2.- Obtener y mostrar el mayor.
3.- Mostrar la suma de todos los componentes.

Utilizar la nueva sintaxis de for vista en este concepto.

Vamos a programar:

ejercicio153.py - D:/Curso completo de Python/ejercicio153.py (3.11.1)

File Edit Format Run Options Window Help

```python
def cargar_lista():
    lista_valores=[]
    for x in range(5):
        valor=int(input("Ingrese un valor: "))
        lista_valores.append(valor)
    return lista_valores

def listado_completo(lista):
    print("Listado")
    for elemento in lista:
        print(elemento)

def valor_mayor(lista):
    max=0
    for elemento in lista:
        if elemento>max:
            max=elemento
    print("El valor máximo es", max)

def suma_valores(lista):
    suma=0
    for elemento in lista:
        suma=suma+elemento
    print("La suma de todos los valores es", suma)

# Bloque principal
lista=cargar_lista()
listado_completo(lista)
valor_mayor(lista)
suma_valores(lista)
```

Guardamos el programa con el nombre de Ejercicio153.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio153.py =============
Ingrese un valor: 1
Ingrese un valor: 2
Ingrese un valor: 3
Ingrese un valor: 4
Ingrese un valor: 5
Listado
1
2
3
4
5
El valor máximo es 5
La suma de todos los valores es 15
>>>
```

Capítulo 154.- Variantes de la estructura repetitiva for para recorrer tuplas y listas – 2

Problema:

Almacenar en una lista de 5 elementos las tuplas con el nombre de empleado y su sueldo.

Implementar las funciones:

1.- Carga de empleados.
2.- Impresión del empleados y sus sueldos.
3.- Nombre del empleado con sueldo mayor.
4.- Cantidad de empleados con sueldo menor a 1000.

Si se cargara por teclado sería algo similar a:

empleados=[("Juan", 2300), ("Pedro", 1200), etc.....]

Vamos a programar:

ejercicio154.py - D:/Curso completo de Python/ejercicio154.py (3.11.1)

File Edit Format Run Options Window Help

```python
def cargar_lista():
    empleados=[]
    for x in range(5):
        empleado=input("Ingrese el nombre del empleado: ")
        sueldo=int(input("Ingrese su sueldo: "))
        empleados.append((empleado, sueldo))
    return empleados

def imprimir(lista):
    print("Listado")
    for empleado in lista:
        print(empleado[0], empleado[1])

def sueldo_mayor(lista):
    max=0
    for elemento in lista:
        if elemento[1]>max:
            max=elemento[1]
            empleado=elemento[0]
    return empleado

def sueldo_menor_1000(lista):
    cantidad=0
    for elemento in lista:
        if elemento[1]<1000:
            cantidad=cantidad+1
    return cantidad

# Bloque principal
empleados=cargar_lista()
imprimir(empleados)
print("El empleado con mayor sueldo es", sueldo_mayor(empleados))
print("Los empleados con sueldo menor a 1000 son", sueldo_menor_1000(empleados))
```

Guardar el programa con el nombre de Ejercicio154.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio154.py =============
Ingrese el nombre del empleado: Juan
Ingrese su sueldo: 1500
Ingrese el nombre del empleado: Luis
Ingrese su sueldo: 950
Ingrese el nombre del empleado: Ana
Ingrese su sueldo: 1300
Ingrese el nombre del empleado: Carlos
Ingrese su sueldo: 850
Ingrese el nombre del empleado: Pedro
Ingrese su sueldo: 2300
Listado
Juan 1500
Luis 950
Ana 1300
Carlos 850
Pedro 2300
El empleado con mayor sueldo es Pedro
Los empleados con sueldo menor a 1000 son 2
>>>
```

Capítulo 155.- Variantes de la estructura repetitiva for para recorrer tuplas y listas – 3

Problema propuesto:

Definir una función que cargue una lista con palabras y la retorne.

Luego otra función tiene que mostrar todas las palabras de la lista que tienen más de 5 caracteres.

Vamos a programar:

ejercicio155.py - D:/Curso completo de Python/ejercicio155.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar_lista():
    palabras=[]
    cantidad=int(input("Cuantas palabras quieres ingresar: "))
    for x in range(cantidad):
        palabra=input("Ingrese una palabra: ")
        palabras.append(palabra)
    return palabras

def mostrar_mayor_5(lista):
    print("Palabras ingresadas con más de 5 caracteres")
    for elemento in lista:
        if len(elemento)>5:
            print(elemento)

# Bloque principal
palabras=cargar_lista()
mostrar_mayor_5(palabras)
```

Guardamos el programa con el nombre de Ejercicio155.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio155.py =============
Cuantas palabras quieres ingresar: 12
Ingrese una palabra: Enero
Ingrese una palabra: Febrero
Ingrese una palabra: Marzo
Ingrese una palabra: Abril
Ingrese una palabra: Mayo
Ingrese una palabra: Junio
Ingrese una palabra: Julio
Ingrese una palabra: Agosto
Ingrese una palabra: Septiembre
Ingrese una palabra: Octubre
Ingrese una palabra: Noviembre
Ingrese una palabra: Diciembre
Palabras ingresadas con más de 5 caracteres
Febrero
Agosto
Septiembre
Octubre
Noviembre
Diciembre
>>>
```

Capítulo 156.- Variantes de la estructura repetitiva for para recorrer tuplas y listas – 4

Problema propuesto:

Almacenar los nombres de 5 productos y sus precios. Utilizar una lista y cada elemento una tupla con el nombre y el precio.
Desarrollar las funciones:
1.- Cargar por teclado.
2.- Listar los productos y los precios.
3.- Imprimir los productos con precios comprendidos entre 10 y 15.
Vamos a programar:

ejercicio156.py - D:/Curso completo de Python/ejercicio156.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar_lista():
    productos=[]
    for x in range(5):
        producto=input("Ingrese el producto: ")
        precio=float(input("Ingrese su precio: "))
        productos.append((producto, precio))
    return productos

def imprimir(lista):
    for elemento in lista:
        print(elemento[0], elemento[1])


def listar_entre_10_15(lista):
    print("Mostrar los articulos que estan entre 10 y 15")
    for elemento in lista:
        if elemento[1]>=10 and elemento[1]<=15:
              print(elemento[0], elemento[1])


# Bloque principal
articulos=cargar_lista()
imprimir(articulos)
listar_entre_10_15(articulos)
```

Guardar el programa con el nombre de Ejercicio156.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio156.py =============
Ingrese el producto: Patatas
Ingrese su precio: 40
Ingrese el producto: Naranjas
Ingrese su precio: 12
Ingrese el producto: Manzanas
Ingrese su precio: 14
Ingrese el producto: Peras
Ingrese su precio: 30
Ingrese el producto: Apio
Ingrese su precio: 40
```

```
Patatas 40.0
Naranjas 12.0
Manzanas 14.0
Peras 30.0
Apio 40.0
Mostrar los articulos que estan entre 10 y 15
Naranjas 12.0
Manzanas 14.0
```

Otra forma de realizar el programa obteniendo el mismo resultado:

ejercicio156.py - D:/Curso completo de Python/ejercicio156.py (3.11.1)

File Edit Format Run Options Window Help

```
1  def cargar_lista():
2      productos=[]
3      for x in range(5):
4          producto=input("Ingrese el producto: ")
5          precio=float(input("Ingrese su precio: "))
6          productos.append((producto, precio))
7      return productos
8
9  def imprimir(lista):
10     for nombre, precio in lista:
11         print(nombre, precio)
12
13
14 def listar_entre_10_15(lista):
15     print("Mostrar los articulos que estan entre 10 y 15")
16     for nombre, precio in lista:
17         if precio>=10 and precio<=15:
18               print(nombre, precio)
19
20
21 # Bloque principal
22 articulos=cargar_lista()
23 imprimir(articulos)
24 listar_entre_10_15(articulos)
25
```

En la línea 10 observamos como aparecen dos variables una llamada nombre y otra llamada precio, ya que la lista consta de elementos de tipo tupla de dos elementos, los podemos asignar automáticamente a las variables nombre en lugar de elemento[0] y precio en lugar de elemento[1].

Si ejecutamos veremos que se obtiene el mismo resultado.

Capítulo 157.- Estructura de datos tipo diccionario – 1

Hasta ahora hemos presentado dos estructuras fundamentales de datos en Python: listas y tuplas. Ahora presentaremos y comenzaremos a utilizar la estructura de datos tipo diccionario.

La estructura de datos tipo diccionario utiliza una clave para acceder a un valor. El subíndice puede ser un entero, un float, un string una tupla, etc.

Podemos relacionarlo con conceptos que conocemos:

- Un diccionario tradicional que conocemos podemos utilizar un diccionario d ePython para representarlo. La clave sería la palabra y el valor sería la definición de dicha palabra.
- Una agenda personal también la podemos representar como un diccionario. La fecha sería la clave y las actividades de dicha fecha sería el valor.
- Un conjunto de usuarios de un sitio web podemos almacenarlo en un diccionario. El nombre de usuario sería la clave y como valor podríamos almacenar su mail, clave, fechas de login, etc.

Hay muchos problemas de la realidad que se pueden representar mediante un diccionario d Python.

Recordemos que las listas son mutables y las tuplas inmutables. Un diccionario es una estructura de datos mutable es decir podemos agregar elementos, modificar y borrar.

Definición de un diccionario por asignación.

productos={"manzanas":39, "peras":32, "lechuga":17}

print(productos) # {'manzanas':39, 'peras':32, 'lechuga':17}

print(productos["peras"]) # 32

Como vemos debemos encerrar entre llaves los elementos separados por coma. A cada elemento debemos indicar del lado izquierdo del carácter el carácter: la clave y al lado derecho el valor asignado para dicha clave. Por ejemplo para la clave "peras" tenemos un valor entero 32.

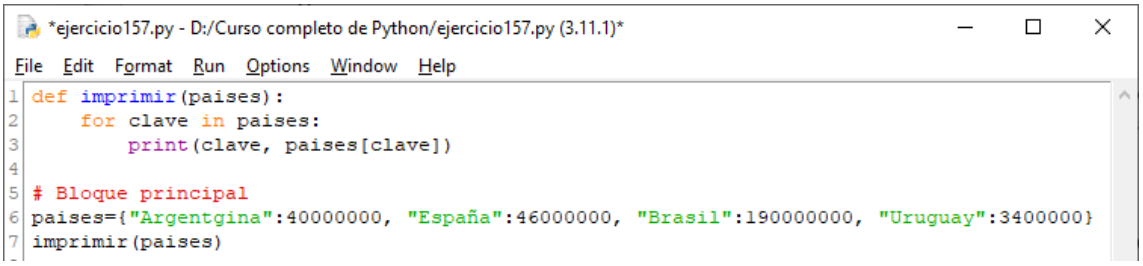
Problema:

en el bloque principal del programa definir un diccionario que almacene los nombres de los países como clave y como valor la cantidad de habitantes. Implementar una función para mostrar cada clave y valor.

Bloque principal

paises={"Argentgina":40000000, "España":46000000. "Brasil":190000000, "Uruguay":3400000}

Vamos a programar:



Guardamos el programa con el nombre de Ejercicio157.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio157.py =============
Argentgina 40000000
España 46000000
Brasil 190000000
Uruguay 3400000
>>>
```

Capítulo 158.- Estructura de datos tipo diccionario – 2

Problema:

Crear un diccionario que permita almacenar 5 artículos, utilizar como clave el nombre de productos y como valor el precio del mismo.

Desarrollar además las funciones de:

1.- Imprimir en forma completa el diccionario.

2.- Imprimir solo los artículos con precio superior a 100.

Vamos a programar:

ejercicio158.py - D:/Curso completo de Python/ejercicio158.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar():
    productos={}
    for x in range(5):
        articulo=input("Ingrese un artículo: ")
        precio=int(input("Ingrese su precio: "))
        productos[articulo]=precio
    return productos

def imprimir(diccionario):
    print("Imprimir toda la lista de artículos")
    for clave in diccionario:
        print(clave, diccionario[clave])

def imprimir_superior100(diccionario):
    print("Listar articulos superiores a 100")
    for clave in diccionario:
        if diccionario[clave]>100:
            print(clave, diccionario[clave])

# Bloque principal
articulos=cargar()
imprimir(articulos)
imprimir_superior100(articulos)
```

Guardar el programa con el nombre de Ejercicio158.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio158.py =============
Ingrese un artículo: Patatas
Ingrese su precio: 50
Ingrese un artículo: Manzanas
Ingrese su precio: 30
Ingrese un artículo: Peras
Ingrese su precio: 70
Ingrese un artículo: Zapallo
Ingrese su precio: 120
Ingrese un artículo: Mandarinas
Ingrese su precio: 110
Imprimir toda la lista de artículos
Patatas 50
Manzanas 30
Peras 70
Zapallo 120
Mandarinas 110
Listar articulos superiores a 100
Zapallo 120
Mandarinas 110
>>>
```

Capítulo 159.- Estructura de datos tipo diccionario – 3

Problema:

Desarrollar una aplicación que nos permita crear un diccionario inglés/castellano. La clave e la palabra en inglés y el valor es la palabra en castellano.

Crear las siguientes funciones:
1.- Cargar el diccionario.
2.- Listado completo del diccionario.
3.- Ingresar por teclado una palabra en inglés y si existe en el diccionario mostrar su traducción.

Vamos a programar:

ejercicio159.py - D:/Curso completo de Python/ejercicio159.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar():
    diccionario={}
    continua="s"
    while continua=="s":
        ingles=input("Ingrese la palabra en inglés: ")
        castellano=input("Ingrese la palabra en castellano: ")
        diccionario[ingles]=castellano
        continua=input("Quiere cargar otra palabra[s/n]?")
    return diccionario

def imprimir(diccionario):
    print("Listado de palabras del diccionario")
    for clave in diccionario:
        print(clave, diccionario[clave])

def traduccion(diccionario):
    pal=input("Dime que palabra quieres consultar en inglés: ");
    if pal in diccionario:
        print(pal, "en castellano es", diccionario[pal])
    else:
        print("No está la traducción de dicha palabra")

# Bloque principal
diccionario=cargar()
imprimir(diccionario)
traduccion(diccionario)
traduccion(diccionario)
```

Si preguntamos por una palabra que no está en el diccionario:

Guardamos el programa con el nombre de Ejercicio159.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio159.py =============
Ingrese la palabra en inglés: red
Ingrese la palabra en castellano: rojo
Quiere cargar otra palabra[s/n]?s
Ingrese la palabra en inglés: blue
Ingrese la palabra en castellano: azul
Quiere cargar otra palabra[s/n]?s
Ingrese la palabra en inglés: green
Ingrese la palabra en castellano: verde
Quiere cargar otra palabra[s/n]?n
Listado de palabras del diccionario
red rojo
blue azul
green verde
Dime que palabra quieres consultar en inglés: blue
blue en castellano es azul
Dime que palabra quieres consultar en inglés: yellow
No está la traducción de dicha palabra
>>>
```

Capítulo 160.- Estructura de datos tipo diccionario – 4

Problema propuesto:

Crear un diccionario en Python que defina como clave el número del documento de la persona y como valor un string con su nombre.

Desarrolle las siguientes funciones:

1.- Cargar por teclado los datos de 4 personas.

2.- Listado completo del diccionario.

3.- Consulta del nombre de una persona ingresando su número de documento.

Vamos a programar:

ejercicio160.py - D:/Curso completo de Python/ejercicio160.py (3.11.1)

File Edit Format Run Options Window Help

```python
def cargar():
    personas={}
    for x in range(4):
        dni=input("Ingrese el DNI. ")
        nombre=input("Ingrese su nombre: ")
        personas[dni]=nombre
    return personas

def imprimir(diccionario):
    print("listado de personas")
    for clave in diccionario:
        print(clave, diccionario[clave])

def consulta(diccionario):
    per=input("Dime el DNI de la persona a consutlar: ")
    if per in diccionario:
        print(per,"es de la persona", diccionario[per])
    else:
        print("Dicho DNI no está en nustra base de datos")

# Bloque principal
personas=cargar()
imprimir(personas)
consulta(personas)
consulta(personas)
```

Guardamos el programa con el nombre de Ejercicio160.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio160.py =============
Ingrese el DNI. 11111111
Ingrese su nombre: Juan
Ingrese el DNI. 22222222
Ingrese su nombre: Ana
Ingrese el DNI. 33333333
Ingrese su nombre: Luis
Ingrese el DNI. 44444444
Ingrese su nombre: Carlos
listado de personas
11111111 Juan
22222222 Ana
33333333 Luis
44444444 Carlos
Dime el DNI de la persona a consutlar: 33333333
33333333 es de la persona Luis
Dime el DNI de la persona a consutlar: 55555555
Dicho DNI no está en nustra base de datos
```

Capítulo 161.- Diccionarios con valores de tipo lista, tuplas y diccionarios – 1

Lo más poderoso que podemos encontrar en las estructuras de datos en Python es que podemos definir elementos que sean también estructuras de datos. En general se dice que podemos anidar una estructura de datos dentro de otra estructura de datos.

Ya vimos en conceptos anteriores que podemos definir elementos de una lista que sean también de tipo lista o de tupla.

Hemos dicho que en un diccionario consta de claves y valores. Desarrollaremos problemas donde los valores para esas claves sean tuplas y o listas.

Problema:

Confeccionar un programa que permita cargar un código de producto como clave en un diccionario.

Guardar para dicha clave una tupla con el nombre del producto, su precio y cantidad en stock.

Implementar las siguientes actividades:
1.- Carga de datos en el diccionario.
2.- Listado completo de productos.
3.- Consulta de un producto por su clave, mostrar el nombre, precio y stock.
4.- Listado de todos los productos que tengan un stock con valor cero.

Vamos a programar:

ejercicio161.py - D:/Curso completo de Python/ejercicio161.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar():
    productos={}
    for x in range(5):
        codigo=int(input("Ingrese el código del artículo: "))
        producto=input("Ingrese el nombre del producto: ")
        precio=float(input("Ingrese el precio del producto :"))
        stock=int(input("Ingrese el stokc del producto :"))
        productos[codigo]=(producto, precio, stock)
    return productos

def listado(productos):
    print("Listado de productos")
    for clave in productos:
        print(clave, productos[clave][0], productos[clave][1], productos[clave][2])

def consulta(productos):
    codigo=int(input("Código del articulos a consutar: "))
    if codigo in productos:
        print("Código", codigo, productos[codigo][0], productos[codigo][1], productos[codigo][2])
    else:
        print("Este código no existe")

def listado_stock_0(productos):
    print("Listado de productos sin stock")
    for clave in productos:
        if productos[clave][2]==0:
            print("Código", clave, productos[clave][0], productos[clave][1], productos[clave][2])

# Bloque principal
articulos=cargar()
listado(articulos)
consulta(articulos)
consulta(articulos)
listado_stock_0(articulos)
```

Guardamos el programa con el nombre de Ejercicio161.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio161.py =============
Ingrese el código del artículo: 1
Ingrese el nombre del producto: Baguet
Ingrese el precio del producto :0.75
Ingrese el stokc del producto :50
Ingrese el código del artículo: 2
Ingrese el nombre del producto: Madalenas
Ingrese el precio del producto :0.30
Ingrese el stokc del producto :0
Ingrese el código del artículo: 3
Ingrese el nombre del producto: Bizcocho
Ingrese el precio del producto :2.30
Ingrese el stokc del producto :7
Ingrese el código del artículo: 4
Ingrese el nombre del producto: Ensiamada
Ingrese el precio del producto :1.20
Ingrese el stokc del producto :0
Ingrese el código del artículo: 5
Ingrese el nombre del producto: Coca
Ingrese el precio del producto :2.30
Ingrese el stokc del producto :10
Listado de productos
1 Baguet 0.75 50
2 Madalenas 0.3 0
3 Bizcocho 2.3 7
4 Ensiamada 1.2 0
5 Coca 2.3 10
Código del articulos a consutar: 3
Código 3 Bizcocho 2.3 7
Código del articulos a consutar: 6
Este código no existe
Listado de productos sin stock
Código 2 Madalenas 0.3 0
Código 4 Ensiamada 1.2 0
>>>
Ln: 34  Col: 0
```

Capítulo 162.- Diccionarios con valores de tipo lista, tuplas y diccionarios – 2

Problema:

Confeccionar una agenda. Utilizar un diccionario cuya clave sea la fecha. Permitir almacenar distintas actividades para la misma fecha(se ingresa la hora y la actividad)

Implementar las siguientes funciones:
1.- Carga de datos en agenda.
2.- Listado completo de la agenda.
3.- Consulta de una fecha.

Para entender si se cargara por asignación tendríamos una estructura de datos similar a:

agenda = {'1/5/2020':[('10:00', 'correr'), ('12:00', 'almorzar'), ('22.00', 'cenar')],

'2/5/2020':[('8:00','gimnasia'),('10:30','reunión de trabajo')]}

Vamos a programar:

ejercicio162.py - D:/Curso completo de Python/ejercicio162.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar():
    agenda={}
    continuar1="s"
    while continuar1=="s":
        fecha=input("Ingrese la fecha: ")
        continuar2="s"
        eventos=[]
        while continuar2=="s":
            hora=input("Ingrese la hora: ")
            evento=input("Ingrese el evento: ")
            eventos.append((hora, evento))
            continuar2=input("Ingresar otra actividad para la misma fecha [s/n]: ")
        agenda[fecha]=eventos
        continuar1=input("Ingresa otra fecha [s/n]: ")
    return agenda

def listado_completo(agenda):
    print("Listado completo de la agenda")
    for fecha in agenda:
        print("Para la fecha", fecha)
        for hora, actividad in agenda[fecha]:
            print(hora, actividad)

def consulta_fecha(agenda):
    fecha=input("Fecha a consultar: ")
    if fecha in agenda:
        print("Fecha:", fecha)
        for hora, actividad in agenda[fecha]:
            print(hora, actividad)
    else:
        print("En dicha fecha no tiene ninguna actividad")

# Bloque principal
agenda=cargar()
listado_completo(agenda)
consulta_fecha(agenda)
consulta_fecha(agenda)
```

Ln: 39 Col: 0

Guardamos el programa con el nombre de Ejercicio162.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio162.py =============
Ingrese la fecha: 1/5/2020
Ingrese la hora: 10:00
Ingrese el evento: correr
Ingresar otra actividad para la misma fecha [s/n]: s
Ingrese la hora: 12:00
Ingrese el evento: almorzar
Ingresar otra actividad para la misma fecha [s/n]: s
Ingrese la hora: 22:00
Ingrese el evento: cenar
Ingresar otra actividad para la misma fecha [s/n]: n
Ingresa otra fecha [s/n]: s
Ingrese la fecha: 2/5/2020
Ingrese la hora: 8:00
Ingrese el evento: gimnasia
Ingresar otra actividad para la misma fecha [s/n]: s
Ingrese la hora: 10:30
Ingrese el evento: reunión de trabajo
Ingresar otra actividad para la misma fecha [s/n]: n
Ingresa otra fecha [s/n]: n
Listado completo de la agenda
Para la fecha 1/5/2020
10:00 correr
12:00 almorzar
22:00 cenar
Para la fecha 2/5/2020
8:00 gimnasia
10:30 reunión de trabajo
Fecha a consultar: 2/5/2020
Fecha: 2/5/2020
8:00 gimnasia
10:30 reunión de trabajo
Fecha a consultar: 3/5/2020
En dicha fecha no tiene ninguna actividad
>>>
Ln: 38  Col: 0
```

Capítulo 163.- Diccionarios con valores de tipo, lista, tuplas y diccionarios – 3

Problema propuesto:

Se desea almacenar los datos de 3 alumnos. Definir un diccionario cuya clave sea el número del documento del alumno. Como valor almacenar una lista con componentes de tipo tupla donde almacenamos nombres de la materia y su nota.

1.- Carga de los alumnos (de cada alumno solicitar su DNI y los nombres de las materias y sus notas)
2.- Listado de todos los alumnos con sus notas.
3.- Consulta de un alumno por su DNI, mostrar las materias que cursa y sus notas.

Si se cargara por asignación tendría una estructura similar a:

```
alumnos = {1:[('programación 1',10),('inglés 1',7)],
           2:[('programación 1',7)],
            3:[('matemáticas',4)]}
```

Vamos a programar:

ejercicio163.py - D:/Curso completo de Python/ejercicio163.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar():
    alumnos={}
    continuar1="s"
    while continuar1=="s":
        dni=input("Ingrese el DNI del alumno: ")
        continuar2="s"
        asignaturas=[]
        while continuar2=="s":
            asignatura=input("Ingrese la asignatura: ")
            nota=int(input("Ingrese la nota: "))
            asignaturas.append((asignatura, nota))
            continuar2=input("Ingrese otra asignatura para el mismos alumno [s/n]: ")
        alumnos[dni]=asignaturas
        continuar1=input("Ingrese otro DNI de otro alumno [s/n]: ")
    return alumnos

def listado_completo(alumnos):
    print("Listado completo de alumnos")
    for dni in alumnos:
        print("Para el alumno con DNI n°:", dni)
        for asignatura, nota in alumnos[dni]:
            print(asignatura, nota)

def consulta_dni(alumnos):
    dni=input("Introduzca el DNI del alumno: ")
    if dni in alumnos:
        print("Alumnos con DNI n°:", dni)
        for asignatura, nota in alumnos[dni]:
            print(asignatura, nota)
    else:
        print("Dicho alumnos con el DNI no existe:")

# Bloque principal
alumnos=cargar()
listado_completo(alumnos)
consulta_dni(alumnos)
consulta_dni(alumnos)
```

Ln: 39 Col: 0

Guardamos el programa con el nombre de Ejercicio163.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio163.py =============
Ingrese el DNI del alumno: 1
Ingrese la asignatura: Programación 1
Ingrese la nota: 10
Ingrese otra asignatura para el mismos alumno [s/n]: s
Ingrese la asignatura: Inglés
Ingrese la nota: 7
Ingrese otra asignatura para el mismos alumno [s/n]: n
Ingrese otro DNI de otro alumno [s/n]: s
Ingrese el DNI del alumno: 2
Ingrese la asignatura: Programación 1
Ingrese la nota: 7
Ingrese otra asignatura para el mismos alumno [s/n]: n
Ingrese otro DNI de otro alumno [s/n]: s
Ingrese el DNI del alumno: 3
Ingrese la asignatura: Matemáticas
Ingrese la nota: 4
Ingrese otra asignatura para el mismos alumno [s/n]: n
Ingrese otro DNI de otro alumno [s/n]: n
Listado completo de alumnos
Para el alumno con DNI n°: 1
Programación 1 10
Inglés 7
Para el alumno con DNI n°: 2
Programación 1 7
Para el alumno con DNI n°: 3
Matemáticas 4
Introduzca el DNI del alumno: 2
Alumnos con DNI n°: 2
Programación 1 7
Introduzca el DNI del alumno: 4
Dicho alumnos con el DNI no existe:
>>>
Ln: 36  Col: 0
```

Capítulo 164.- Funciones parámetros mutables e inmutables – 1

En Python tenemos tipos de datos inmutable:

enteros
float
string
tuplas

Mutables:

listas
diccionarios

Esto tiene mucha importancia cuando enviamos a una función una variable mutable, veremos como un ejemplo como podemos pasar como parámetro una lista a una función y posteriormente cambiar su contenido y esto se vea reflejado en la variable que le enviamos al llamarla.

Problema:

Confeccionar un programa que contenga la siguientes funciones:

1.- Cargar una lista y retorne al bloque principal.
2.- Fijar en cero todos los elementos de la lista que tengan un valor menor a 10.
3.- Imprimir la lista

Vamos a programar:

ejercicio164.py - D:/Curso completo de Python/ejercicio164.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar():
    lista=[]
    continuar="s"
    while continuar=="s":
        valor=int(input("Ingrese un valor: "))
        lista.append(valor)
        continuar=input("Agregar otro elemento a la lista [s/n]: ")
    return lista

def fijar_cero(lista):
    for x in range(len(lista)):
        if lista[x]<10:
            lista[x]=0


# Bloque principal
lista=cargar()
print(lista)
fijar_cero(lista)
print(lista)

```

Guardar el programa con el nombre de Ejercicio164.py y a continuación ejecutamos.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio164.py =============
Ingrese un valor: 12
Agregar otro elemento a la lista [s/n]: s
Ingrese un valor: 5
Agregar otro elemento a la lista [s/n]: s
Ingrese un valor: 15
Agregar otro elemento a la lista [s/n]: s
Ingrese un valor: 7
Agregar otro elemento a la lista [s/n]: s
Ingrese un valor: 25
Agregar otro elemento a la lista [s/n]: s
Ingrese un valor: 9
Agregar otro elemento a la lista [s/n]: s
Ingrese un valor: 14
Agregar otro elemento a la lista [s/n]: s
Ingrese un valor: 6
Agregar otro elemento a la lista [s/n]: n
[12, 5, 15, 7, 25, 9, 14, 6]
[12, 0, 15, 0, 25, 0, 14, 0]
>>>
```

Capítulo 165.- Funciones parámetros mutables e inmutables – 2

Problema:

Confeccionar un programa que contenga las siguientes funciones:

1.- Cargar una lista de 5 nombres.

2.- Ordenar alfabéticamente la lista.

3.- Imprimir la lista de nombres.

Vamos a programar:

ejercicio165.py - D:/Curso completo de Python/ejercicio165.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar():
    lista=[]
    for x in range(5):
        nombre=input("Ingrese un nombre: ")
        lista.append(nombre)
    return lista

def ordenar_lista(nombres):
    for x in range(4):
        for y in range(4):
            if nombres[y]>nombres[y+1]:
                aux=nombres[y]
                nombres[y]=nombres[y+1]
                nombres[y+1]=aux
    return nombres

def imprimir(nombres):
    for x in range(5):
        print(nombres[x])

# Bloque principal
nombres=cargar()
print("Nombres sin ordenar")
imprimir(nombres)
ordenar_lista(nombres)
print("Nombres ordenados")
imprimir(nombres)
```

Guardamos el programa con el nombre de Ejercicio165.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio165.py =============
Ingrese un nombre: Carlos
Ingrese un nombre: Luis
Ingrese un nombre: Ana
Ingrese un nombre: Pedro
Ingrese un nombre: Juan
Nombres sin ordenar
Carlos
Luis
Ana
Pedro
Juan
Nombres ordenados
Ana
Carlos
Juan
Luis
Pedro
```

Capítulo 166.- Funciones parámetros mutables e inmutables – 3

Problema:

Confeccionar un programa que almacene en un diccionario como clave el nombre de un contacto y como valor su número telefónico.

1.- Cargar de contactos y su número telefónico.
2.- Permitir modificar el número telefónico. Se ingresa el nombre del contacto para su búsqueda.
3.- Imprimir la lista completa de contactos con sus números telefónicos.

Vamos a programar:

ejercicio166.py - D:/Curso completo de Python/ejercicio166.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar_diccionario():
    agenda={}
    continuar="s"
    while continuar=="s":
        nombre=input("Ingrese un nombre: ")
        telefono=input("Ingrese su teléfono: ")
        agenda[nombre]=telefono
        continuar=input("Desea agregar un nuevo contacto [s/n]:")
    return agenda

def modificar_contacto(agenda):
    nombre=input("Ingrese el nombre del contacto a modificar: ")
    telefono=input("Ingrese el núevo teléfono: ")
    agenda[nombre]=telefono

def imprimir(agenda):
    for nombre in agenda:
        print(nombre, agenda[nombre])

# Bloque principal
agenda=cargar_diccionario()
print("Mostrar listado antes de modificar")
imprimir(agenda)
modificar_contacto(agenda)
print("Mostrar listado después de modificar")
imprimir(agenda)
```

Guardar el programa con el nombre de Ejercicio166.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio166.py =============
Ingrese un nombre: Carlos
Ingrese su teléfono: 1111111
Desea agregar un nuevo contacto [s/n]:s
Ingrese un nombre: Luis
Ingrese su teléfono: 22222222
Desea agregar un nuevo contacto [s/n]:s
Ingrese un nombre: Pedro
Ingrese su teléfono: 33333333
Desea agregar un nuevo contacto [s/n]:s
Ingrese un nombre: Antonio
Ingrese su teléfono: 44444444
Desea agregar un nuevo contacto [s/n]:s
```

```
Ingrese un nombre: Ana
Ingrese su teléfono: 55555555
Desea agregar un nuevo contacto [s/n]:n
Mostrar listado antes de modificar
Carlos 1111111
Luis 22222222
Pedro 33333333
Antonio 44444444
Ana 55555555
Ingrese el nombre del contacto a modificar: Pedro
Ingrese el núevo teléfono: 66666666
Mostrar listado después de modificar
Carlos 1111111
Luis 22222222
Pedro 66666666
Antonio 44444444
Ana 55555555
>>>
```

Ln: 34 Col: 0

Capítulo 167.- Funciones parámetros mutables e inmutables – 4

Problema propuesto:

Crear un diccionario en Python para almacenar los datos de empleados de una empresa. La clave seré su número de expediente y en su valor almacenar una lista con el nombre, profesión y sueldo.

Desarrollar las siguientes funciones:
1.- Carga de datos de empleados.
2.- Permitir modificar el sueldo de un empleado. Ingresamos su número de expediente para buscarlo.
3.- Mostrar todos los datos de empleado que tienen una profesión de "analista de sistemas".

Vamos a programar:

ejercicio167.py - D:/Curso completo de Python/ejercicio167.py (3.11.1)

File Edit Format Run Options Window Help

```python
def cargar():
    empleados={}
    continuar="s"
    while continuar=="s":
        expediente=int(input("Ingrese el expediente: "))
        datos=[]
        nombre=input("Ingrese el nombre: ")
        profesion=input("Ingrese la profesión: ")
        sueldo=int(input("Ingrese el suldo: "))
        empleados[expediente]=[nombre, profesion, sueldo]
        continuar=input("Quiere ingresar otro empleado [s/n]: ")
    return empleados
def modificar_sueldo(empleados):
    expediente=int(input("Ingrese el expediente: "))
    if expediente in empleados:
        sueldo=int(input("Ingrese el nuevo sueldo: "))
        empleados[expediente][2]=sueldo
    else:
        print("Dicho expediente no existe")

def mostrar_analista(empleados):
    print("Consultar por los datos del analista de sistemas")
    for expediente in empleados:
        if empleados[expediente][1]=="analista de sistemas":
            print(expediente, empleados[expediente][0],empleados[expediente][2])

# Bloque principal
empleados=cargar()
print(empleados)
modificar_sueldo(empleados)
modificar_sueldo(empleados)
print(empleados)
mostrar_analista(empleados)
```

Ln: 34 Col: 0

Guardamos el programa con el nombre de Ejercicio167.py y a continuación lo ejecutamos.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio167.py =============
Ingrese el expediente: 1
Ingrese el nombre: Carlos
Ingrese la profesión: Gerente
Ingrese el suldo: 2300
Quiere ingresar otro empleado [s/n]: s
Ingrese el expediente: 2
Ingrese el nombre: Luis
Ingrese la profesión: analista de sistemas
Ingrese el suldo: 1900
Quiere ingresar otro empleado [s/n]: s
Ingrese el expediente: 3
Ingrese el nombre: Juan
Ingrese la profesión: Programador
Ingrese el suldo: 1500
Quiere ingresar otro empleado [s/n]: n
{1: ['Carlos', 'Gerente', 2300], 2: ['Luis', 'analista de sistemas', 1900], 3: [
'Juan', 'Programador', 1500]}
Ingrese el expediente: 3
Ingrese el nuevo sueldo: 1650
Ingrese el expediente: 4
Dicho expediente no existe
{1: ['Carlos', 'Gerente', 2300], 2: ['Luis', 'analista de sistemas', 1900], 3: [
'Juan', 'Programador', 1650]}
Consultar por los datos del analista de sistemas
2 Luis 1900
>>>
Ln: 28  Col: 0
```

Capítulo 168.- Porciones de listas, tuplas y cadenas de caracteres – 1

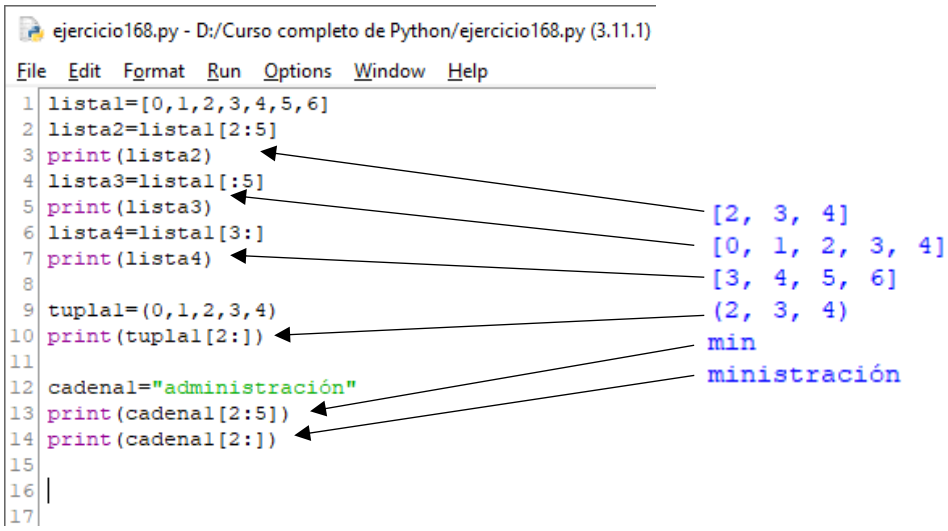
El lenguaje Python nos facilita una sintaxis muy sencilla par recuperar un trozo de una lista, tupla o cadena de caracteres.

Veremos con una serie de ejemplos como podemos rescatar uno o varios elementos de las estructura de datos mencionadas.

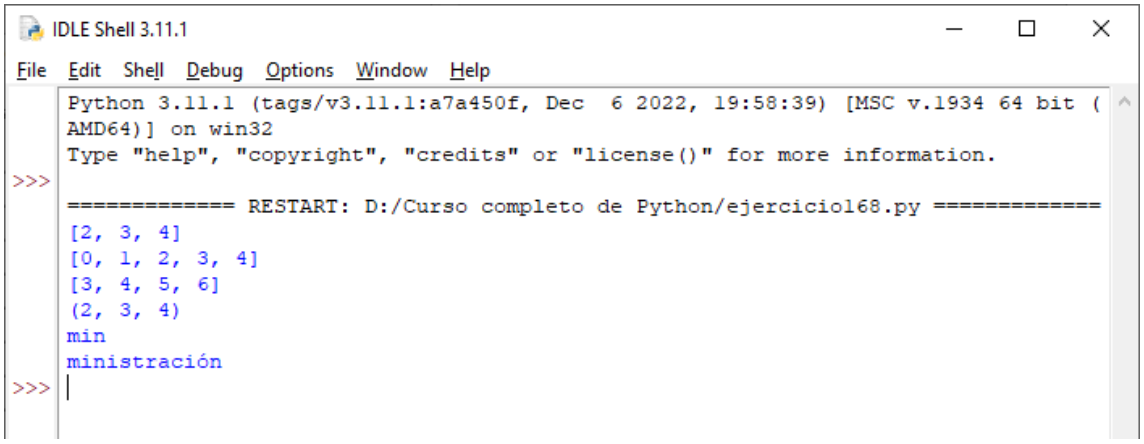
Problema:

Probar extraer porciones de una lista.

Vamos a programar:



Guardamos el programa con el nombre de Ejercicio168.py y a continuación lo ejecutamos:



Capítulo 169.- Porciones de listas, tuplas y cadenas de caracteres – 2

Problema:

Confeccionar una función que le enviemos un número de mes como parámetro y nos retorne una tupla con todos los nombres de meses que faltan hasta fin de año (utilizar el concepto de porciones)

meses=('enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio', 'julio', 'agosto', 'septiembre', 'octubre', 'noviembre', 'diciembre')

Vamos a programar:

ejercicio169.py - D:/Curso completo de Python/ejercicio169.py (3.11.1)

File Edit Format Run Options Window Help

```
def meses_pendientes(numero_mes):
    meses=('enero', 'febrero', 'marzo',
       'abril', 'mayo', 'junio',
       'julio', 'agosto', 'septiembre',
       'octubre', 'noviembre', 'diciembre')
    return meses[numero_mes:]


# Bloque principal
num=int(input("Ingrese un número de mes: "))
print("Meses que faltan para fin de año")
print(meses_pendientes(num))

```

Guardar el programa con el nombre de Ejercicio169.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio169.py =============
Ingrese un número de mes: 6
Meses que faltan para fin de año
('julio', 'agosto', 'septiembre', 'octubre', 'noviembre', 'diciembre')
>>>
```

Capítulo 170.- Porciones de listas, tuplas y cadenas de caracteres – 3

Problema:

Confeccionar una función que reciba una cadena de caracteres y nos devuelva los tres primeros. En el bloque principal el programa definir una tupla con los nombres de meses. Mostrar por pantalla los primeros tres caracteres de cada mes.

Vamos a programar:

ejercicio170.py - D:/Curso completo de Python/ejercicio170.py (3.11.1)

File Edit Format Run Options Window Help

```
def primeros_tres(cadena):
    return cadena[:3]

# Bloque principal
meses=('enero', 'febrero', 'marzo',
       'abril', 'mayo', 'junio',
       'julio', 'agosto', 'septiembre',
       'octubre', 'noviembre', 'diciembre')

for mes in meses:
    print(primeros_tres(mes))

```

Guardamos el programa con el nombre de Ejercicio170.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio170.py =============
ene
feb
mar
abr
may
jun
jul
ago
sep
oct
nov
dic
>>>
```

Capítulo 171.- Porciones de listas, tuplas y cadenas de caracteres – 4

Problema propuesto:

Realizar un programa que contenga las siguientes funciones:
1.- Carga una lista de 10 enteros.
2.- Recibir una lista y retornar otra con la primera mitad (se sabe que siempre llega una lista con una cantidad par de elementos)
3.- Imprimir una lista

Vamos a programar:

ejercicio171.py - D:/Curso completo de Python/ejercicio171.py (3.11.1)

File Edit Format Run Options Window Help

```
def cargar():
    lista=[]
    for x in range(10):
        elementos=int(input("Ingrese un valor: "))
        lista.append(elementos)
    return lista

def retornar_mitad(lista):
    lista1=lista[:(len(lista)//2)] # Dividir por entero
    return lista1

def imprimir_lista(lista):
    print("contenido de la lista")
    print(lista)

# Bloque principal
lista=cargar()
mitad_lista=retornar_mitad(lista)
imprimir_lista(lista)
imprimir_lista(mitad_lista)
```

Guardamos el programa con el nombre de Ejercicio171.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio171.py =============
Ingrese un valor: 1
Ingrese un valor: 2
Ingrese un valor: 3
Ingrese un valor: 4
Ingrese un valor: 5
Ingrese un valor: 6
Ingrese un valor: 7
Ingrese un valor: 8
Ingrese un valor: 9
Ingrese un valor: 10
contenido de la lista
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
contenido de la lista
[1, 2, 3, 4, 5]
>>>
```

Capítulo 172.- Porciones de listas, tuplas y cadenas de caracteres – 5

Problema propuesto:

Cargar una cadena por teclado luego:
1.- Imprimir los dos primeros caracteres.
2.- imprimir los dos últimos caracteres.
3.- imprimir todo menos el primero y el último carácter.

Vamos a programar:

```
ejercicio172.py - D:/Curso completo de Python/ejercicio172.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 cadena=input("Ingrese una cadena de caracteres: ")
2 print("Primeros dos caracteres")
3 print(cadena[:2])
4 print("Los dos últimos caracteres")
5 print(cadena[len(cadena)-2:])
6 print("Imprimir todos los caracteres menos el primero y el último")
7 print(cadena[1:len(cadena)-1])
8
```

Guardar el programa con el nombre de Ejercicio172.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio172.py =============
    Ingrese una cadena de caracteres: Python
    Primeros dos caracteres
    Py
    Los dos últimos caracteres
    on
    Imprimir todos los caracteres menos el primero y el último
    ytho
>>>
```

Capítulo 173.- Índices negativos en listas, tuplas y cadenas de caracteres – 1

Hemos visto que para acceder a un elemento de una lista, tupla o cadena de caracteres debemos indicar mediante su subíndice que comienza a numerarse a partir de cero.

También hemos visto el concepto anterior que podemos generar otra lista, tupla o cadena de caracteres indicando una porción con el carácter ":".

Ahora veremos que podemos utilizar un valor negativo para acceder a un elemento de la estructura de datos.

Programa: Ejercicio175.py

```
lista1=[0, 1, 2, 3, 4, 5, 6]
print(lista[-1])   # 6
print(lista[-2])   # 5
```

En Python podemos acceder fácilmente al último elemento de la secuencia indicando un subíndice -1:

```
print(lista[-1])   # 6
```

Luego el anteúltimo se accede con la sintaxis:

```
print(lista[-2])   # 5
```

Problema:

Confeccionar una función que reciba una palabra y verifique si es capicúa (es decir que se lee igual de izquierda a derecha que de derecha a izquierda)

Vamos a programar:

ejercicio173.py - D:\Curso completo de Python\ejercicio173.py (3.11.1)

File Edit Format Run Options Window Help

```
1  def capicua(cadena):
2      mitad_longitud_entera=len(cadena)//2
3      cantidad=0
4      for x in range(mitad_longitud_entera):
5          if cadena[x]==cadena[(x+1)*(-1)]:
6              cantidad=cantidad+1
7      if mitad_longitud_entera==cantidad:
8          print("La palabra", cadena, "es capicúa")
9      else:
10         print("La palabra", cadena, "no es capicúa")
11
12 # Bloque principal
13 capicua("neuquen")
14 capicua("casa")
15
```

Guardamos con el nombre de Ejercicio173.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:\Curso completo de Python\ejercicio173.py =============
La palabra neuquen es capicúa
La palabra casa no es capicúa
>>>
```

Otra forma de realizarlo y obtener el mismo resultado:

```
ejercicio173.py - D:\Curso completo de Python\ejercicio173.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1  def capicua(cadena):
2      iguales=0
3      indice=-1
4      for x in range(0, len(cadena)//2):
5          if cadena[x]==cadena[indice]:
6              iguales=iguales+1
7          indice=indice-1
8      if iguales==(len(cadena)//2):
9          print("La palabra", cadena, "es capicúa")
10     else:
11         print("La palabra", cadena, "no es capicúa")
12
13 # Bloque principal
14 capicua("neuquen")
15 capicua("casa")
16
```

Capítulo 174.- Índices negativos en listas, tuplas y cadenas de caracteres – 2

Problema propuesto:

Cargar una cadena de caracteres por teclado. Mostrar la cadena del final al principio utilizando subíndices negativos.

Vamos a programar:

```
ejercicio174.py - D:/Curso completo de Python/ejercicio174.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 cadena=input("Ingrese una palabra: ")
2 valor_negativo=len(cadena)*(-1)
3 pal=""
4 for x in range(-1, valor_negativo-1, -1):
5     pal=pal+cadena[x]
6 print(pal)
7
```

Guardamos el programa con el nombre de Ejercicio174.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio174.py =============
    Ingrese una palabra: administración
    nóicartsinimda
>>>
```

Otra forma de hacer y obtener el mismo resultado:

```
ejercicio174.py - D:/Curso completo de Python/ejercicio174.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 palabra=input("Ingrese una palabra: ")
2 indice=-1
3 for x in range(len(palabra)):
4     print(palabra[indice], end="")
5     indice=indice-1
6
```

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio174.py =============
    Ingrese una palabra: Administración
    nóicartsinimdA
>>>
```

Capítulo 175.- Índices negativos en listas, tuplas y cadenas de caracteres – 3

Problema propuesto:

Confeccionar un programa con las siguientes funciones:
1.- Cargar una lista de 5 palabras.
2.- Intercambiar la primera palabra con la última.
3.- Imprimir la lista.

Vamos a programar:

ejercicio175.py - D:/Curso completo de Python/ejercicio175.py (3.11.1)

File Edit Format Run Options Window Help

```python
def cargar():
    lista=[]
    for x in range(5):
        palabra=input("Ingrese una palabra: ")
        lista.append(palabra)
    return lista

def intercambiar(lista):
    aux=lista[0]
    lista[0]=lista[-1]
    lista[-1]=aux

def imprimir(lista):
    for elemento in lista:
        print(elemento, end=" ")
    print()
    print("---------------")

# Bloque principal
lista=cargar()
imprimir(lista)
intercambiar(lista)
imprimir(lista)
```

Guardamos el programa con el nombre de Ejercicio175.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio175.py =============
Ingrese una palabra: Azucar
Ingrese una palabra: Harina
Ingrese una palabra: Sal
Ingrese una palabra: Arroz
Ingrese una palabra: Lentejas
Azucar Harina Sal Arroz Lentejas
---------------
Lentejas Harina Sal Arroz Azucar
---------------
>>>
```

Capítulo 176.- Biblioteca estándar de Python – 1

Todos los conceptos que hemos visto hasta ahora los hemos resuelto utilizando solo la sintaxis que nos provee Python y un conjunto de funciones básicas que se incluyen automáticamente como por ejemplo son print, range, len, etc.

En Python se incluye una biblioteca extra de funciones, variables, clases, etc. que nos facilitan la resolución de problemas en una gran diversidad de áreas como matemáticas, estadísticas, comprensión de datos, internet, interfaces visuales, etc.

Veamos en este concepto como se importa un módulo de la biblioteca estándar y como se accede a su funcionalidad.

Problema:

Confeccionar un programa que simule tirar dos dados y luego muestre los valore que salieron.

Imprimir un mensaje que ganó si la suma de los mismos es igual a 7.

Para resolver este problema requerimos un algoritmo para que se genere un valor aleatorio entre 1 y 6. Como la generación de valores aleatorios es un tema muy frecuente la biblioteca estándar de Python incluye un módulo que nos resuelve la generación de valores aleatorios.

Vamos a programar:

ejercicio176.py - D:/Curso completo de Python/ejercicio176.py (3.11.1)

File Edit Format Run Options Window Help

```
1  import random
2
3  dado1=random.randint(1,6)
4  dado2=random.randint(1,6)
5  print("Primer dado:", dado1)
6  print("Segundo dado:", dado2)
7  suma=dado1+dado2
8  if suma==7:
9      print("Ganó")
10 else:
11     print("Perdio")
```

Guardamos el programa con el nombre de Ejercicio176.py y a continuación lo vamos a ejecutar varias veces:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio176.py =============
Primer dado: 5
Segundo dado: 5
Perdio
>>>
============= RESTART: D:/Curso completo de Python/ejercicio176.py =============
Primer dado: 3
Segundo dado: 4
Ganó
>>>
```

```
============= RESTART: D:/Curso completo de Python/ejercicio176.py =============
Primer dado: 5
Segundo dado: 3
Perdio
>>>
============= RESTART: D:/Curso completo de Python/ejercicio176.py =============
Primer dado: 3
Segundo dado: 6
Perdio
>>>
============= RESTART: D:/Curso completo de Python/ejercicio176.py =============
Primer dado: 4
Segundo dado: 1
Perdio
>>>
============= RESTART: D:/Curso completo de Python/ejercicio176.py =============
Primer dado: 6
Segundo dado: 5
Perdio
>>>
```

Ln: 33 Col: 0

De la 6 tiradas realizadas solo hemos ganado en una sola vez.

Para poder importar funcionalidades de la biblioteca estándar tenemos que importar el módulo.

```
import random

dado1=random.randint(1,6)
dado2=random.randint(1,6)
```

Para utilizar la función randint(valor1, valor2) que lo que hace es generar un número aleatorio comprendido entre los valores valor1 y valor2.

Una vez que hemos importado el módulo, podremos llamarlo con su función respectiva.

Capítulo 177.- Biblioteca estándar de Python – 2

Problema:

Desarrollar un programa que cargue una lista con 10 enteros.
Cargar los valores aleatorios con números enteros comprendidos entre 0 y 1000.
Mostrar la lista por pantalla.
Luego mezclar los elementos de la vista y volver a mostrarlo.

Vamos a programar:

ejercicio177.py - D:/Curso completo de Python/ejercicio177.py (3.11.1)

File Edit Format Run Options Window Help

```
import random

def cargar():
    lista=[]
    for x in range(10):
        lista.append(random.randint(0,1000))
    return lista

def imprimir(lista):
    print(lista)

def mezclar(lista):
    random.shuffle(lista)

# Bloque principal
lista=cargar()
imprimir(lista)
mezclar(lista)
imprimir(lista)
```

Guardamos el programa con el nombre de Ejercicio177.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio177.py =============
[572, 127, 176, 699, 902, 680, 442, 90, 294, 528]
[90, 902, 528, 699, 442, 176, 680, 572, 294, 127]
>>>
```

En el siguiente enlace podremos consultar las librerías estándar:

http://docs.python.org/3/library/random.html

Table of Contents

random — Generate pseudo-random numbers
- Bookkeeping functions
- Functions for bytes
- Functions for integers
- Functions for sequences
- Real-valued distributions
- Alternative Generator
- Notes on Reproducibility
- Examples
- Recipes

Previous topic

random — Generate pseudo-random numbers

Source code: Lib/random.py

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, the von Mises distribution is available.

Capítulo 178.- Biblioteca estándar de Python – 3

Problema propuesto:

Confeccionar un programa que genere un número aleatorio entre 1 y 100 y no se muestre. El operador debe tratar de adivinar el número ingresado.
Cada vez que ingrese un número mostrar un mensaje "Ganó" si es igual al generado o "El número aleatorio es mayor" o "El número aleatorio en menor".
Mostrar cuando gana el jugador cuantos intentos necesitó.

Vamos a programar:

ejercicio178.py - D:/Curso completo de Python/ejercicio178.py (3.11.1)

File Edit Format Run Options Window Help

```
import random
numero_aleatorio=random.randint(0,100)
numero=-1
intentos=0
while(numero_aleatorio!=numero):
    numero=int(input("Ingrese un número de 0 a 100: "))
    intentos=intentos+1
    if numero>numero_aleatorio:
        print("El número es menor")
    else:
        if numero<numero_aleatorio:
            print("El número es mayor")
print("Enhorabuena has acertado al intento", intentos)

```

Guardamos el programa con el nombre de Ejercicio178.py y a continuación ejecutamos.

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio178.py =============
Ingrese un número de 0 a 100: 50
El número es mayor
Ingrese un número de 0 a 100: 60
El número es mayor
Ingrese un número de 0 a 100: 70
El número es mayor
Ingrese un número de 0 a 100: 80
El número es menor
Ingrese un número de 0 a 100: 75
El número es menor
Ingrese un número de 0 a 100: 72
Enhorabuena has acertado al intento 6
>>>
```

Capítulo 179.- Biblioteca estándar de Python – 4

Problema propuesto:

Confeccionar un programa con las siguientes funciones:
1.- Generar una lista con 4 elementos enteros aleatorios comprendidos entre 1 y 3. Agregar un quinto elemento con un 1.
2.- Controlar que el primer elemento de la lista sea un 1, en caso que haya 2 o 3 mezclar la lista y volver a controlar hasta que haya un 1.
3.- Imprimir la lista.

Vamos a programar:

ejercicio179.py - D:/Curso completo de Python/ejercicio179.py (3.11.1)

File Edit Format Run Options Window Help

```
import random

def cargar():
    lista=[]
    for x in range(4):
        valor=random.randint(1,3)
        lista.append(valor)
    lista.append(1)
    return lista

def controlar_primero(lista):
    contador=0
    while lista[0]!=1:
        random.shuffle(lista)
        contador=contador+1
    print("Ha generado", contador, "listas")

def imprimir(lista):
    print(lista)


# Bloque principal
lista=cargar()
imprimir(lista)
controlar_primero(lista)
imprimir(lista)
```

Guardamos el programa con el nombre de Ejercicio179.py y a continuación lo ejecutamos varias veces:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio179.py =============
[2, 1, 3, 3, 1]
Ha generado 2 listas
[1, 3, 2, 1, 3]
>>>
============= RESTART: D:/Curso completo de Python/ejercicio179.py =============
[2, 3, 1, 2, 1]
Ha generado 1 listas
[1, 3, 2, 2, 1]
>>>
```

```
============= RESTART: D:/Curso completo de Python/ejercicio179.py =============
[3, 2, 2, 3, 1]
Ha generado 8 listas
[1, 3, 2, 2, 3]
>>>
============= RESTART: D:/Curso completo de Python/ejercicio179.py =============
[1, 2, 3, 3, 1]
Ha generado 0 listas
[1, 2, 3, 3, 1]
>>>
```

Capítulo 180.- Importar algunas funcionaliddes de un módulo de la biblioteca estándar de Python – 1

Hemos visto que para importar toda la funcionalidad de un módulo de la Biblioteca estándar de Python utilizamos la palabra clave import y seguidamente el nombre del módulo:

import random

Con esta sintaxis todas las funcionalidades del módulo "random" pueden ser accesibles desde nuestro módulo.

Ahora veremos que en Python tenemos otra sintaxis para las situaciones que queremos acceder una o pocas funcionalidades de un módulo.

Por ejemplo si queremos acceder solo a la función randint del módulo random en Python lo podemos expresar con la siguiente sintaxis:

from random import randint

Utilizamos la palabra clave from y seguidamente el nombre del módulo de donde queremos importar funcionalidades del mismo. Luego indicamos la palabra clave import y la funcionalidad que queremos importar, en nuestro ejemplo la función randint.

También cambia como utilizamos la función randint dentro de nuestro módulo:

valor=randint(1,10)
print(valor)

Como vemos no le antecedemos ningún nombre de módulo y hacemos referencia directamente a la función importada.

Si necesitamos importar más de una funcionalidad de un módulo debemos separar por comas las funcionalidades importadas:

from random import randint, shuffle

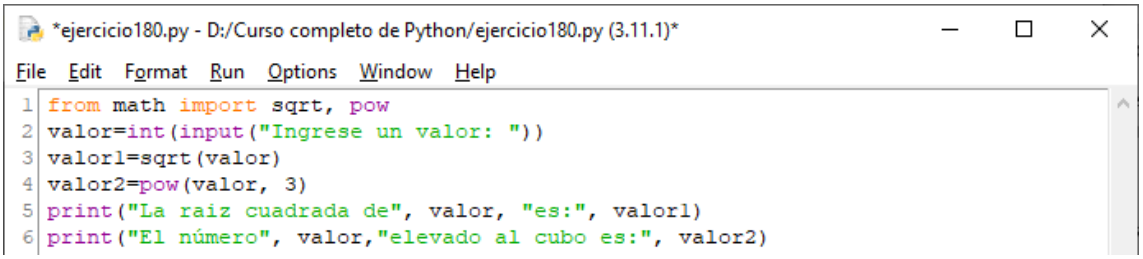
Problema:

Confeccionar un programa que solicite la carga de un valor entero por teclado y luego nos muestre la raíz cuadrada del número y el valor elevado al cubo.

Para resolver este problema utilizamos dos funcionalidades que nos provee el módulo math de la biblioteca estándar de Python:

from math import sqrt, pow

Vamos a programar:



Guardamos el programa con el nombre de Ejercicio180.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio180.py =============
Ingrese un valor: 9
La raiz cuadrada de 9 es: 3.0
El número 9 elevado al cubo es: 729.0
>>>
```

Capítulo 181.- Importar algunas funcionalidades de un módulo de la biblioteca estándar de Python – 2

Podemos definir un nombre distinto para una funcionalidad que importamos de otro módulo. Esto puede tener como objetivo que nuestro programa sea más legible o evitar que un nombre de función que importamos colisione con un nombre de función de nuestro propio módulo.

Resolvemos el mismo programa que solicite cargar un valor por teclado y luego nos muestra la raíz cuadrada del número y el valor elevado al cubo.

Vamos a programar:

```
ejercicio181.py - D:/Curso completo de Python/ejercicio181.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 from math import sqrt as raiz, pow as elevar
2 valor=int(input("Ingrese un valor: "))
3 valor1=raiz(valor)
4 valor2=elevar(valor, 3)
5 print("La raiz cuadrada de", valor, "es:", valor1)
6 print("El número", valor,"elevado al cubo es:", valor2)
7
```

Guardamos el programa con el nombre de Ejercicio181.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio181.py =============
    Ingrese un valor: 9
    La raiz cuadrada de 9 es: 3.0
    El número 9 elevado al cubo es: 729.0
>>>
```

Capítulo 182.- Importar algunas funcionalidades de un módulo de la biblioteca estándar de Python – 3

Problema propuesto:

Calcular el factorial de un número ingresado por teclado.
El factorial de un número es la cantidad que resulta de la multiplicación de determinado número natural por todos los números naturales que le anteceden excluyendo el cero. Por ejemplo el factorial de 4 es 24, que resulta de multiplicar 4*3*2*1.
No hay que implementar el algoritmo para calcular el factorial sino hay que importar dicha funcionalidad de módulo math.
El módulo math tiene una función llamada factorial que recibe como parámetro un entero del que necesitamos que nos retorne el factorial.
Solo importar la funcionalidad factorial del módulo math de la biblioteca estándar de Python.

Vamos a programar:

```
ejercicio182.py - D:/Curso completo de Python/ejercicio182.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 from math import factorial
2 valor=int(input("Ingrese un valor: "))
3 print("El factorial de", valor, "es", factorial(valor))
4
```

Guardamos el programa con el nombre Ejercicio182.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
  Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
  AMD64)] on win32
  Type "help", "copyright", "credits" or "license()" for more information.
>>>
  ============= RESTART: D:/Curso completo de Python/ejercicio182.py =============
  Ingrese un valor: 5
  El factorial de 5 es 120
>>>
```

Nos ahorraríamos este proceso:

```
ejercicio182.py - D:/Curso completo de Python/ejercicio182.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 valor=int(input("Ingrese un valor: "))
2 valor1=valor
3 factorial=1
4 while valor>0:
5     factorial=factorial*valor
6     valor=valor-1
7 print("El factorial de", valor1, "es", factorial)
8
```

El resultado será el mismo:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
  Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
  AMD64)] on win32
  Type "help", "copyright", "credits" or "license()" for more information.
>>>
  ============= RESTART: D:/Curso completo de Python/ejercicio182.py =============
  Ingrese un valor: 5
  El factorial de 5 es 120
```

Capítulo 183.- Aplicaciones propias con varios módulos – 1

A medida que una aplicación sea más grande vimos que disponer todo el algoritmo en forma secuencial es difícil de mantener y desarrollar. Para eso vimos que podemos resolver por partes utilizando funciones.

Pero siempre dispusimos todas las funciones en un único archivo.

Pensemos que nuestro problema podría ser lo sufrientemente complejo y grande que también por más que lo dividamos en funciones un único archivo sea complejo de mantener.

En Python podemos agrupar funciones en distintos archivos (módulos) y luego importarlos en otros módulos.

Así como importamos los módulos de la biblioteca estándar de Python podemos importar otros módulos que codifiquemos nosotros.

Problema:

Confeccionar una aplicación que permita cargar por teclado un lista de 5 enteros, obtener y mostrar el mayor y calcular la suma. Definir un módulo con las funciones carga, identificar el mayor y sumar. En el módulo principal del programa importar el otro módulo y llamar a sus funciones.

Para ser un poco más ordenados crearemos una carpeta llamada proyecto1 y dentro de la misma creamos los dos módulos llamados:

operacioneslista.py
principal.py

El módulo operacioneslista.py contiene todas las funciones que nos permiten cargar la lista, imprimir el mayor de la lista y sumar todos los elementos y mostrar dicho valor.

Vamos a programar:

operacioneslista.py - D:/Curso completo de Python/proyecto1/operacioneslista.py (3.11.1)

File Edit Format Run Options Window Help

```
1  def cargar():
2      lista=[]
3      for x in range(5):
4          valor=int(input("Ingrese un valor:"))
5          lista.append(valor)
6      return lista
7
8  def imprimir_mayor(lista):
9      mayor=lista[0]
10     for x in range(1,5):
11         if lista[x]>mayor:
12             mayor=lista[x]
13     print("Mayor elemento de la lista", mayor)
14
15 def imprimir_suma(lista):
16     suma=0
17     for elemento in lista:
18         suma=suma+elemento
19     print("La suma de sus elementos es", suma)
20
```

Este archivo lo tenemos que guardar en la carpeta proyecto1 que hemos creado con anterioridad con el nombre de operacioneslista.py.

principal.py - D:/Curso completo de Python/proyecto1/principal.py (3.11.1)

File Edit Format Run Options Window Help

```
import operacioneslista

lista=operacioneslista.cargar()
operacioneslista.imprimir_mayor(lista)
operacioneslista.imprimir_suma(lista)
```

Este segundo archivo lo guardaremos en la misma carpeta prouecto1 con el nombre de principal.py.

Ambos archivos tienen que estar en la misma carpeta ya que el archivo principal.py tiene que importar las funciones del archivo operacioneslista.

Ahora tenemos que ejecutar el programa principal.py que es el programa que importa las funciones del archivo operacioneslista.py y las ejecuta.

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:/Curso completo de Python/proyecto1/principal.py =========
Ingrese un valor:1
Ingrese un valor:2
Ingrese un valor:3
Ingrese un valor:4
Ingrese un valor:5
Mayor elemento de la lista 5
La suma de sus elementos es 15
>>>
```

Capítulo 184.- Aplicaciones propias con varios módulos – 2

Problema:

Confeccionar un módulo que implemente dos funciones, una que retorne el mayor de dos enteros y otra que retorne el menor de dos enteros.

En el módulo principal importar solo la función que retorne el mayor, luego cargar dos enteros y mostrar el mayor de ellos.

Crea una carpeta llamada proyecto2 y dentro de la misma crea dos módulos:

Vamos a programar:

```
mayormenor.py - D:/Curso completo de Python/proyecto2/mayormenor.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
def mayor(x1, x2):
    if x1>x2:
        return x1
    else:
        return x2

def menor(x1, x2):
    if x1<x2:
        return x1
    else:
        return x2

```

Ya hemos creado el archivo que contiene las dos funciones, lo guardamos con el nombre de mayormenor.py y lo guardamos en la carpeta proyecto2.

```
*principal.py - D:/Curso completo de Python/proyecto2/principal.py (3.11.1)*
File  Edit  Format  Run  Options  Window  Help
from mayormenor import mayor

valor1=int(input("Ingres el primer valor: "))
valor2=int(input("Ingrese el segundo valor: "))
print("El mayor de los dos valores es", mayor(valor1,valor2))

```

Guardamos este programa en la carpeta proyecto2 con el nombre principal.py este programa se encargará de importar la función mayor de las dos que tiene y ejecutar el programa:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:/Curso completo de Python/proyecto2/principal.py =========
Ingres el primer valor: 30
Ingrese el segundo valor: 60
El mayor de los dos valores es 60
>>>
```

Ahora lo vamos a modificar con un alias.

principal.py - D:/Curso completo de Python/proyecto2/principal.py (3.11.1)

File Edit Format Run Options Window Help

```
from mayormenor import mayor as mayor_entero

valor1=int(input("Ingres el primer valor: "))
valor2=int(input("Ingrese el segundo valor: "))
print("El mayor de los dos valores es", mayor_entero(valor1,valor2))

```

Al ejecutar el resultado será el mismo.

Capítulo 185.- Declaración de una clase y creación de objetos – 1

El lenguaje Python tiene la característica de permitir programar con la siguientes metodologías:

- Programación Lineal: es cuando desarrollamos todo el código sin emplear funciones. El código es una secuencia lineal de comandos.
- Programación Estructurada: Es cuando planteamos funciones que agrupan actividades a desarrollar y luego dentro del programa llamamos a dichas funciones que pueden estar dentro del mismo archivo (módulo) o en librería separada.
- Programación Orientada a Objetos: Es cuando planteamos clases y definimos objetos de las mismas (Este es el objetivo de los próximos conceptos, aprender la metodología de programación orientada a objetos y la sintaxis particular de Python para la POO).

Ahora introduciremos los conceptos de programación orientada a objetos. A partir de este concepto mostraremos en forma sencilla la metodología de Programación Orientada a Objetos.

Si irán introduciendo conceptos de objeto, clases, atributo, método, etc. y de todos estos temas irán planteando problemas.

Prácticamente todos los lenguajes desarrollados en los últimos 25 años implementan la posibilidad de trabajar con POO (Programación Orientada a Objetos)

Concepto básico de Objetos

Un objeto es una entidad independiente con sus propios datos y programación. Las ventanas, menús, carpetas de archivos pueden ser identificados como objetos; el motor de un auto también es considerado un objeto, en este caso, sus datos (atributos) describen sus características físicas y su programación (método) describen el funcionamiento interno y su interrelación con otras partes del automóvil (también objetos).

El concepto renovador de la tecnología de Orientación a Objeto es la suma de funciones a elementos de datos, a esta unión se les llama encapsulamiento.

Por ejemplo, un objeto Auto contiene ruedas, motor, velocidad, color, etc, llamados atributos. Encapsulados con estos datos se encuentra los métodos para arrancar, detenerse, doblar, frenar, etc. La responsabilidad de un objeto auto consiste en realizar las acciones apropiadas y mantener actualizados sus datos internos.

Cuando otra parte del programa (otros objetos) necesitan que el auto realice algunas de estas tareas (por ejemplo arrancar) le envía un mensaje. A estos objetos que envían mensajes no les interesa la manera en que el objeto auto lleva a cabo sus tareas ni las estructuras de datos que maneja, por ello, están ocultos.

Entonces, un objeto contiene información pública, lo que necesitan los otros objetos para interactuar con él e información privada, interna, lo que necesita el objeto para operar y que es irrelevante para los otros objetos de la aplicación. Declaración de una clase y creación de objetos.

La programación orientada a objetos se basa en la definición de clases; a diferencia de la programación estructural, que está centrada en las funciones.

Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.

Un poco más abajo se define una clase Persona y luego se crean dos objetos de dicha clase.
Una clase es una plantilla (molde), que define atributos (lo que conocemos como variables) y método (lo que conocemos como funciones).
La clase define los atributos y métodos comunes a los objetos de este tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.
Debemos declarar una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

Problema:

Implementamos una clase llamada Persona que tendrá como atributo (variable) su nombre y dos métodos (funciones), uno de dichos métodos inicializará el atributo nombre y el siguiente método mostrará en la pantalla el contenido del mismo.

Definir dos objetos de la clase Persona.

Vamos a programar:

ejercicio186.py - D:/Curso completo de Python/ejercicio186.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Persona:
2
3      def inicializar(self, nom):
4          self.nombre=nom
5
6      def imprimir(self):
7          print("Nombre", self.nombre)
8
9  # Bloque principal
10
11 personal=Persona()
12 personal.inicializar("Pedro")
13 personal.imprimir()
14
15 persona2=Persona()
16 persona2.inicializar("Ana")
17 persona2.imprimir()
18
```

En la línea 1 creamos una clase con el nombre de Persona.
En la línea 3 creamos un atributo llamado inicializar, que nos permitirá asignar un nombre a los objetos creados por esta clase.
En la línea 6 creamos un método para mostrar por pantalla el nombre de la persona que hemos instanciado.
En la línea 11 instanciamos un objeto llamado persona1 que tendrá los atributos y métodos de la clase Persona.
En la línea 15 lo repetimos con un nuevo objeto llamado persona2.
Guardamos el programa con el nombre de Ejercicio186.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio186.py =============
Nombre Pedro
Nombre Ana
>>>
```

Capítulo 186.- Declaración de una clase y creación de objetos – 2

Problema:

Implementar una clase llamada Alumno que tenga atributos su nombre y su nota. Definir los métodos para inicializar sus atributos, imprimirlos y mostrar un mensaje si está apto (nota mayor o igual a 4), de lo contrario no apto.

Definir dos objetos de la clase Alumnos.

Vamos a programar:

ejercicio187.py - D:/Curso completo de Python/ejercicio187.py (3.11.1)

File Edit Format Run Options Window Help

```
class Alumno:

    def inicializar(self, nombre, nota):
        self.nombre=nombre
        self.nota=nota

    def imprimir(self):
        print("Nombre: ", self.nombre)
        print("Nota: ", self.nota)



    def mostrar_estado(self):
        if self.nota>=4:
            print(self.nombre, "APTO")
        else:
            print(self.nombre, "NO APTO")

# Bloque principal
alumno1=Alumno()
alumno1.inicializar("Juan", 2)
alumno1.imprimir()
alumno1.mostrar_estado()

alumno2=Alumno()
alumno2.inicializar("Ana", 10)
alumno2.imprimir()
alumno2.mostrar_estado()
```

Guardamos el programa con el nombre de Ejercicio186.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio187.py =============
Nombre:  Juan
Nota:  2
Juan NO APTO
Nombre:  Ana
Nota:  10
Ana APTO
>>>
```

Capítulo 187.- Declaración de una clase y creación de objetos – 3

Problema propuesto:

Confeccionar una clase que permita cargar el nombre y la edad de una persona. Mostar los datos cargados. Imprimir un mensaje si es mayor de edad (edad>=18).

Vamos a programar:

ejercicio187.py - D:/Curso completo de Python/ejercicio187.py (3.11.1)

File Edit Format Run Options Window Help

```
class Persona:

    def inicializar(self, nombre, edad):
        self.nombre=nombre
        self.edad=edad

    def imprimir(self):
        print("Nombre: ", self.nombre)
        print("Edad: ", self.edad)



    def mayor_edad(self):
        if self.edad>=18:
            print(self.nombre, "Es mayor de edad")
        else:
            print(self.nombre, "No es mayor de edad")

# Bloque principal
personal=Persona()
personal.inicializar("Juan", 15)
personal.imprimir()
personal.mayor_edad()

persona2=Persona()
persona2.inicializar("Ana", 22)
persona2.imprimir()
persona2.mayor_edad()

```

Guardar el programa con el nombre de Ejercicio187.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio187.py =============
Nombre:  Juan
Edad:  15
Juan No es mayor de edad
Nombre:  Ana
Edad:  22
Ana Es mayor de edad
>>>
```

Capítulo 188.- Declaración de una clase y creación de objetos – 4

Problema propuesto:

Desarrollar un programa que cargue los dados de un triángulo por teclado e implemente los siguientes métodos: Inicializar los atributos, imprimir el valor del lado mayor y otro método que muestre si es equilátero o no (los tres lados miden lo mismo). El nombre de la clase llamarla Triangulo.

Vamos a programar:

ejercicio188.py - D:/Curso completo de Python/ejercicio188.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Triangulo:
2
3      def inicializar(self):
4          self.lado1=int(input("Ingrese primer lado: "))
5          self.lado2=int(input("Ingrese segundo lado: "))
6          self.lado3=int(input("Ingrese tercer lado: "))
7
8      def imprimir(self):
9          print("Lado1:", self.lado1)
10         print("Lado2:", self.lado2)
11         print("Lado3:", self.lado3)
12
13     def lado_mayor(self):
14         if self.lado1>self.lado2 and self.lado2>self.lado3:
15             print("El lado mayor es de", self.lado1)
16         else:
17             if(self.lado2>self.lado3):
18                 print("El lado mayor es de", self.lado2)
19             else:
20                 print("El lado mayor es de", self.lado3)
21
22     def es_equilatero(self):
23         if self.lado1==self.lado2 and self.lado1==self.lado3:
24             print("Este triangulo es equilátero")
25         else:
26             print("Este triangulo no es equilátero")
27
28
29 # Bloque principal
30 triangulo1=Triangulo()
31 triangulo1.inicializar()
32 triangulo1.imprimir()
33 triangulo1.lado_mayor()
34 triangulo1.es_equilatero()
35
```

Guardamos el programa con el nombre de Ejercicio188.py y ejecutamos dos veces el programa:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio188.py =============
Ingrese primer lado: 6
Ingrese segundo lado: 6
Ingrese tercer lado: 6
Lado1: 6
Lado2: 6
Lado3: 6
El lado mayor es de 6
Este triangulo es equilátero
```

```
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio188.py =============
    Ingrese primer lado: 3
    Ingrese segundo lado: 7
    Ingrese tercer lado: 5
    Lado1: 3
    Lado2: 7
    Lado3: 5
    El lado mayor es de 7
    Este triangulo no es equilátero
>>>
```

Capítulo 189.- Método __init__ de la clase – 1

El método __init__ es un método especial de una clase en Python. El objetivo fundamental del método __init__ es inicializar los atributos del objeto que creamos.

Básicamente el método __init__ remplaza al método inicializar que ha habíamos hecho en el concepto anterior.

Las ventajas de implementar el método __init__ en lugar del método inicializar son:

1. El método __init__ es el primer método que se ejecuta cuando se crea un objeto.
2. El método __init__ se llama automáticamente. Es decir es imposible de olvidarse de llamarlo ya que se llamará automáticamente.
3. Quien utiliza POO en Python (Programación Orientada a Objetos) conoce el objetivo de este método.

Otras características del método __init__ son:

- Se ejecuta inmediatamente luego de crear un objeto.
- El método __init__ no puede retornar dato.
- El método __init__ puede recibir parámetros que se utilizan normalmente para inicializar atributos.
- El método __init__ es un método opcional, de todos modos es muy común declararlo.

Veamos la sintaxis del constructor:

def __init__([parámetros])
    [algoritmo]

Problema:

Confeccionar una clase que represente un empleado. Definir como atributos su nombre y sueldo. En el método __init__ cargar los atributos por teclado y luego en otro método imprimir sus datos y por último uno que imprima un mensaje si debe pagar impuestos (si el sueldo supera a los 3000).

Vamos a programar:

ejercicio189.py - D:/Curso completo de Python/ejercicio189.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Empleados:
2      def __init__(self):
3          self.nombre=input("Ingrese su nombre:")
4          self.sueldo=int(input("Ingrese su sueldo:"))
5
6      def imprimir(self):
7          print("Nombre:", self.nombre)
8          print("Sueldo:", self.sueldo)
9
10     def pagar_impuestos(self):
11         if self.sueldo>3000:
12             print(self.nombre, "Debe pagar impuestos")
13         else:
14             print(self.nombre, "No debe pagar impuestos")
15
```

```
16 # Bloque principal
17 empleado1=Empleados()
18 empleado1.imprimir()
19 empleado1.pagar_impuestos()
20
21 empleado2=Empleados()
22 empleado2.imprimir()
23 empleado2.pagar_impuestos()
24
```

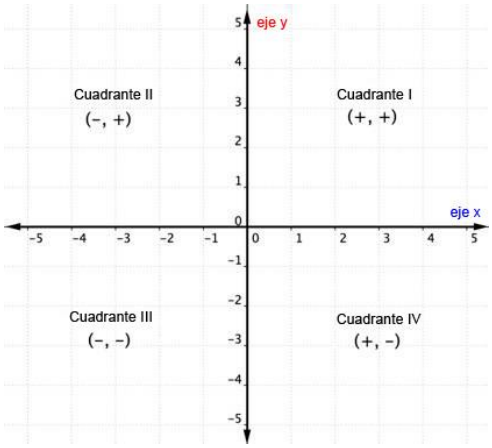
Guardamos el programa con el nombre de Ejercicio189.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio189.py =============
Ingrese su nombre:Juan
Ingrese su sueldo:3500
Nombre: Juan
Sueldo: 3500
Juan Debe pagar impuestos
Ingrese su nombre:Luis
Ingrese su sueldo:2500
Nombre: Luis
Sueldo: 2500
Luis No debe pagar impuestos
>>>
```

Capítulo 190.- Método __init__ de la clase – 2

Problema:

Desarrollar una clase que represente un punto en un plano y tenga los siguientes métodos: Inicializar los valores de x e y que llegan como parámetros, imprimir en que cuadrante se encuentra dicho punto (concepto matemático, primer cuadrante si x e y son positivas, si x<0 e y>0 segundo cuadrante, etc.)



Vamos a programar:

ejercicio190.py - D:/Curso completo de Python/ejercicio190.py (3.11.1)

File Edit Format Run Options Window Help

```
class Plano:
    def __init__(self, x, y):
        self.x=x
        self.y=y

    def imprimir(self):
        print("Posición x: ", self.x)
        print("Posición y: ", self.y)

    def pos_cuadrante(self):
        if self.x>0 and self.y>0:
            print("Pertenece al cuadrante I")
        else:
            if self.x<0 and self.y>0:
                print("Pertenece al cuadante II")
            else:
                if self.x<0 and self.y<0:
                    print("Pertenece al cuadrante III")
                else:
                    if self.x>0 and self.y<0:
                        print("Pertenece al cuadrante IV")
                    else:
                        print("No se encuentra en ningún cuadrante")
                        print("Se encuentra en un eje")

# Bloque principal
punto1=Plano(3,2)
punto1.imprimir()
punto1.pos_cuadrante()

punto2=Plano(-4,5)
punto2.imprimir()
punto2.pos_cuadrante()
```

```
35 punto3=Plano(-1,-3)
36 punto3.imprimir()
37 punto3.pos_cuadrante()
38
39 punto4=Plano(4,-2)
40 punto4.imprimir()
41 punto4.pos_cuadrante()
42
43 punto5=Plano(0,2)
44 punto5.imprimir()
45 punto5.pos_cuadrante()
```

Ln: 47 Col: 0

Guardamos el programa con el nombre de Ejercicio190.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio190.py =============
Posición x:  3
Posición y:  2
Pertenece al cuadrante I
Posición x:  -4
Posición y:  5
Pertenece al cuadante II
Posición x:  -1
Posición y:  -3
Pertenece al cuadrante III
Posición x:  4
Posición y:  -2
Pertenece al cuadrante IV
Posición x:  0
Posición y:  2
No se encuentra en ningún cuadrante
Se encuentra en un eje
>>>
```

Capítulo 191.- Método __init__ de la clase – 3

Problema propuesto:

Desarrollar una clase que represente un Cuadrado y tenga los siguientes métodos: Inicializar el valor del lado llegando como parámetro al método __init__ (definir un atributo llamado lado), imprimir su perímetro y su superficie.

Vamos a programar:

ejercicio191.py - D:/Curso completo de Python/ejercicio191.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Cuadrado:
2
3      def __init__(self, lado):
4          self.lado=lado
5
6      def mostrar_perimetro(self):
7          perimetro=self.lado*4
8          print("Su perimetro es de", perimetro)
9
10     def mostrar_superficie(self):
11         superficie=self.lado*self.lado
12         print("Su superficie es de", superficie)
13
14 # Bloque principal
15 cuadrado1=Cuadrado(2)
16 cuadrado1.mostrar_perimetro()
17 cuadrado1.mostrar_superficie()
18
19 cuadrado2=Cuadrado(4)
20 cuadrado2.mostrar_perimetro()
21 cuadrado2.mostrar_superficie()
22
23 cuadrado3=Cuadrado(8)
24 cuadrado3.mostrar_perimetro()
25 cuadrado3.mostrar_superficie()
26
```

Guardamos el programa con el nombre de Ejercicio191.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio191.py =============
Su perimetro es de 8
Su superficie es de 4
Su perimetro es de 16
Su superficie es de 16
Su perimetro es de 32
Su superficie es de 64
>>>
```

Capítulo 192.- Método __init__ de la clase – 4

Problema propuesto:

Implementar una clase Operaciones. Se debe cargar dos valores enteros por teclado en el método __init__, calcular su suma, resta, multiplicación y división, cada una en un método, imprimir dichos resultados.

Vamos a programar:

ejercicio192.py - D:/Curso completo de Python/ejercicio192.py (3.11.1)

File Edit Format Run Options Window Help

```
class Operaciones:

    def __init__(self):
        self.num1=int(input("Ingrese el primer número: "))
        self.num2=int(input("Ingrese el segundo número: "))

    def suma(self):
        sumar=self.num1+self.num2
        print("La suma de ", self.num1, "+", self.num2, "es de", sumar)

    def resta(self):
        restar=self.num1-self.num2
        print("La resta de ", self.num1, "-", self.num2, "es de", restar)

    def multiplicacion(self):
        multiplicar=self.num1*self.num2
        print("La multiplicacion de ", self.num1, "*", self.num2, "es de", multiplicar)

    def division(self):
        dividir=self.num1/self.num2
        print("La división de ", self.num1, "/", self.num2, "es de", dividir)

# Bloque Principal

operacion1=Operaciones()
operacion1.suma()
operacion1.resta()
operacion1.multiplicacion()
operacion1.division()
```

Guardamos el programa con el nombre de Ejercicio192.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio192.py =============
Ingrese el primer número: 10
Ingrese el segundo número: 5
La suma de  10 + 5 es de 15
La resta de  10 - 5 es de 5
La multiplicacion de  10 * 5 es de 50
La división de  10 / 5 es de 2.0
>>>
```

Capítulo 193.- Llamada de métodos desde otro método de la misma clase – 1

Hasta ahora todos los problemas planteados hemos llamado a los métodos desde donde definimos un objeto de dicha clase, por ejemplo.

empleado1=Empleado("Diego", 2000)
empleado1.paga_impuestos()

Utilizamos la sintaxis:

[nombre del objeto].[nombre del método]

Es decir antecedemos al nombre del método el nombre del objeto y el operador punto.

Ahora bien que pasa si queremos llamar dentro de la clase a otro método que pertenece a la misma clase, la sintaxis es la siguiente:

self.[nombre del método]

Es importante tener en cuenta que esto solo se puede hacer cuando estamos dentro de la misma clase.

Problema:

Plantear una clase Operaciones que solicite en método __init__ la carga de dos enteros e inmediatamente muestre su suma, resta, multiplicación y división. Hacer cada operación en otro método de la clase Operación y llamarlos desde el mismo método __init__.

Vamos a programar:

ejercicio193.py - D:/Curso completo de Python/ejercicio193.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Operaciones:
2
3      def __init__(self):
4          self.num1=int(input("Ingrese el primer número: "))
5          self.num2=int(input("Ingrese el segundo número: "))
6          self.suma()
7          self.resta()
8          self.multiplicacion()
9          self.division()
10
11     def suma(self):
12         sumar=self.num1+self.num2
13         print("La suma de ", self.num1, "+", self.num2, "es de", sumar)
14
15     def resta(self):
16         restar=self.num1-self.num2
17         print("La resta de ", self.num1, "-", self.num2, "es de", restar)
18
19     def multiplicacion(self):
20         multiplicar=self.num1*self.num2
21         print("La multiplicacion de ", self.num1, "*", self.num2, "es de", multiplicar)
22
23     def division(self):
24         dividir=self.num1/self.num2
25         print("La división de ", self.num1, "/", self.num2, "es de", dividir)
26
27 # Bloque Principal
28
29 operacion1=Operaciones()
```

Guardamos el programa con el nombre de Ejercicio193.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio193.py =============
Ingrese el primer número: 10
Ingrese el segundo número: 5
La suma de  10 + 5 es de 15
La resta de  10 - 5 es de 5
La multiplicacion de  10 * 5 es de 50
La división de  10 / 5 es de 2.0
>>>
```

Capítulo 194.- Llamada de métodos desde otro método de la misma clase – 2

Problema:

Plantear una clase que administre dos listas de 5 nombres de alumnos y sus notas. Mostrar un menú de opciones que permita:

1- Cargar alumnos
2- Listar alumnos
3- Mostrar alumnos con notas mayores o iguales a 7.
4- Finalizar programa.

Vamos a programar:

ejercicio194.py - D:/Curso completo de Python/ejercicio194.py (3.11.1)

File Edit Format Run Options Window Help

```
class Alumnos:

    def __init__(self):
        self.nombres=[]
        self.notas=[]

    def menu(self):
        op=0
        while op!=4:
            print("1.- Cargar alumnos.")
            print("2.- Listar alumnos.")
            print("3.- Mostrar alumnos con notas mayores o iguales a 7.")
            print("4.- Finalizar programa.")
            op=int(input("Ingrese su opción: "))
            if op==1:
                self.cargar()
            else:
                if op==2:
                    self.listar()
                else:
                    if op==3:
                        self.notas_altas()

    def cargar(self):
        for x in range(5):
            alumno=input("Ingrese el nombre del alumno: ")
            self.nombres.append(alumno)
            nota=int(input("Ingrese la nota del alumno: "))
            self.notas.append(nota)

    def listar(self):
        print("Listado completo")
        for x in range(5):
            print(self.nombres[x], self.notas[x])

    def notas_altas(self):
        print("Listado notas iguales o mayores a 7")
        for x in range(5):
            if self.notas[x]>=7:
                print(self.nombres[x], self.notas[x])

# Bloque principal
alumno1=Alumnos()
alumno1.menu()
```

Ln: 46 Col: 0

Guardamos el programa con el nombre de Ejercicio194.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio194.py =============
1.- Cargar alumnos.
2.- Listar alumnos.
3.- Mostrar alumnos con notas mayores o iguales a 7.
4.- Finalizar programa.
Ingrese su opción: 1
Ingrese el nombre del alumno: Carlos
Ingrese la nota del alumno: 8
Ingrese el nombre del alumno: Ana
Ingrese la nota del alumno: 5
Ingrese el nombre del alumno: Pedro
Ingrese la nota del alumno: 10
Ingrese el nombre del alumno: Juan
Ingrese la nota del alumno: 5
Ingrese el nombre del alumno: Luis
Ingrese la nota del alumno: 9
1.- Cargar alumnos.
2.- Listar alumnos.
3.- Mostrar alumnos con notas mayores o iguales a 7.
4.- Finalizar programa.
Ingrese su opción: 2
Listado completo
Carlos 8
Ana 5
Pedro 10
Juan 5
Luis 9
1.- Cargar alumnos.
2.- Listar alumnos.
3.- Mostrar alumnos con notas mayores o iguales a 7.
4.- Finalizar programa.
Ingrese su opción: 3
Listado notas iguales o mayores a 7
Carlos 8
Pedro 10
Luis 9
1.- Cargar alumnos.
2.- Listar alumnos.
3.- Mostrar alumnos con notas mayores o iguales a 7.
4.- Finalizar programa.
Ingrese su opción: 4
>>>
Ln: 45  Col: 0
```

Capítulo 195.- Llamada de métodos desde otro método de la misma clase – 3

Problema propuesto:

Confeccionar una clase que administre una agenda personal. Se debe almacenar el nombre de la persona, teléfono y mail. Vamos a utilizar los diccionarios.

Debe mostrar un menú con las siguientes opciones:
1.- Carga de un contacto en la agenda.
2.- Listado completo de la agenda.
3.- Consulta ingresando el nombre de la persona.
4.- Modificación de su teléfono y mail.
5.- Finalizar programa.

Vamos a programar.

```
class Agenda:

    def __init__(self):
        self.contactos={}

    def menu(self):
        op=0
        while op!=5:
            print("-------------------------------------------------------------------")
            print("| 1.- Carga de un contanto en la agenda                 |")
            print("| 2.- Listado completo de la agenda                     |")
            print("| 3.- Consulta ingresando el nombre de la persona |")
            print("| 4.- Modificación de su teléfono y mail                 |")
            print("| 5.- Finalizar programa                                 |")
            print("-------------------------------------------------------------------")
            op=int(input("Ingrese su opción: "))
            if op==1:
                self.cargar()
            else:
                if op==2:
                    self.listado()
                else:
                    if op==3:
                        self.consulta()
                    else:
                        if op==4:
                            self.modificacion()

    def cargar(self):
        print("Introducir datos")
        print("----------------")
        nombre=input("Ingrese el nombre del alumno: ")
        telefono=input("ingrese el teléfono: ")
```

```
        mail=input("Ingrese el mail: ")
        self.contactos[nombre]=[telefono, mail]
        print("_____________________________________")

    def listado(self):
        print("Listado de la agenda")
        print("--------------------")
        for nombre in self.contactos:
            print(nombre, self.contactos[nombre][0], self.contactos[nombre][1])

    def consulta(self):
        print("Consulta")
        print("--------")
        nom=input("Que nombre quires consultar: ")
        if nom in (self.contactos):
            print("Datos de", nom)
            print("Su teléfono es", self.contactos[nom][0])
            print("Su mail es", self.contactos[nom][1])
        else:
            print("Este nombre no está en la agenda")

    def modificacion(self):
        print("Modificaciones")
        print("--------------")
        nom=input("Los datos de quin quiers modificar: ")
        if nom in (self.contactos):
            tel=input("Ingrese el nuevo teléfono: ")
            mai=input("Ingrese el nuevo mail: ")
            self.contactos[nom]=[tel, mai]
        else:
            print("Este nombre no está en la agenda")

# Bloque principal

alumno1=Agenda()
alumno1.menu()
```

Lo guardamos con el nombre de Ejercicio195.py y a continuación lo ejecutamos:

```
----------------------------------------------------
| 1.- Carga de un contanto en la agenda            |
| 2.- Listado completo de la agenda                |
| 3.- Consulta ingresando el nombre de la persona  |
| 4.- Modificación de su teléfono y mail           |
| 5.- Finalizar programa                           |
----------------------------------------------------
Ingrese su opción: 1
```

Seleccionamos la opción 1.

```
Introducir datos
----------------
Ingrese el nombre del alumno: Calos
ingrese el teléfono: 666111111
Ingrese el mail: carlos@hotmail.com
```

Añadimos el siguiente alumno y lo repetimos dos veces más.

```
Introducir datos
----------------
Ingrese el nombre del alumno: Luis
ingrese el teléfono: 666222222
Ingrese el mail: luis@hotmail.com

Introducir datos
----------------
Ingrese el nombre del alumno: Juan
ingrese el teléfono: 666333333
Ingrese el mail: juan@hotmail.com
```

Vamos a consultar por un listado.

```
---------------------------------------------------
| 1.- Carga de un contanto en la agenda           |
| 2.- Listado completo de la agenda               |
| 3.- Consulta ingresando el nombre de la persona |
| 4.- Modificación de su teléfono y mail          |
| 5.- Finalizar programa                          |
---------------------------------------------------
Ingrese su opción: 2
Listado de la agenda
--------------------
Calos 666111111 carlos@hotmail.com
Luis 666222222 luis@hotmail.com
Juan 666333333 juan@hotmail.com
```

Consultamos por un alumno que no está agenda.

```
---------------------------------------------------
| 1.- Carga de un contanto en la agenda           |
| 2.- Listado completo de la agenda               |
| 3.- Consulta ingresando el nombre de la persona |
| 4.- Modificación de su teléfono y mail          |
| 5.- Finalizar programa                          |
---------------------------------------------------
Ingrese su opción: 3
Consulta
--------
Que nombre quires consultar: Ana
Este nombre no está en la agenda
```

Ahora vamos a consultar por un alumno que si está en la agenda.

```
-----------------------------------------------------
| 1.- Carga de un contanto en la agenda             |
| 2.- Listado completo de la agenda                 |
| 3.- Consulta ingresando el nombre de la persona   |
| 4.- Modificación de su teléfono y mail            |
| 5.- Finalizar programa                            |
-----------------------------------------------------
Ingrese su opción: 3
Consulta
--------
Que nombre quires consultar: Luis
Datos de Luis
Su teléfono es 666222222
Su mail es luis@hotmail.com
```

Ahora vamos a modificar el teléfono de Luis.

```
-----------------------------------------------------
| 1.- Carga de un contanto en la agenda             |
| 2.- Listado completo de la agenda                 |
| 3.- Consulta ingresando el nombre de la persona   |
| 4.- Modificación de su teléfono y mail            |
| 5.- Finalizar programa                            |
-----------------------------------------------------
Ingrese su opción: 4
Modificaciones
--------------
Los datos de quin quiers modificar: Luis
Ingrese el nuevo teléfono: 666999999
Ingrese el nuevo mail: luis@hotmail.com
```

Vamos a consultar de nuevo por los datos de Luis.

```
-----------------------------------------------------
| 1.- Carga de un contanto en la agenda             |
| 2.- Listado completo de la agenda                 |
| 3.- Consulta ingresando el nombre de la persona   |
| 4.- Modificación de su teléfono y mail            |
| 5.- Finalizar programa                            |
-----------------------------------------------------
Ingrese su opción: 3
Consulta
--------
Que nombre quires consultar: Luis
Datos de Luis
Su teléfono es 666999999
Su mail es luis@hotmail.com
```

Observamos que hemos modificado el teléfono.

Ahora vamos a finalizar con la opción 5.

```
-----------------------------------------------------
| 1.- Carga de un contanto en la agenda             |
| 2.- Listado completo de la agenda                 |
| 3.- Consulta ingresando el nombre de la persona   |
| 4.- Modificación de su teléfono y mail            |
| 5.- Finalizar programa                            |
-----------------------------------------------------
Ingrese su opción: 5

>
```

Capítulo 196.- Colaboración de clases – 1

Normalmente un problema resuelto con la metodología de programación orientada a objetos no interviene una sola clase, sino que hay muchas clases que interactúan y se comunican.

Plantearemos un problema separando las actividades en dos clases.

Problema:

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule la cantidad de dinero que hay en depositado.

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Cliente y la clase Banco.

Luego debemos definir los atributos y los métodos de cada clase:

Cliente
- atributos
  - nombre
  - importe
- métodos
  - __init__
  - depositar
  - extraer
  - retornar_importe
  - imprimir

Banco
- atributos
  - 3 Clientes (3 objetos de la clase Clientes)
- métodos
  - __init__
  - operar
  - depósitos_totales

Vamos a programar:

```
class Cliente:

  def __init__(self, nombre):
    self.nombre=nombre
    self.importe=0

  def depositar(self, importe):
    self.importe=self.importe+importe

  def extraer(self, importe):
    self.importe=self.importe-importe

  def retornar_importe(self):
    return self.importe
```

```
    def imprimir(self):
        print(self.nombre, "Tiene depositado la suma de", self.importe)

class Banco:

    def __init__(self):
        self.cliente1=Cliente("Juan")
        self.cliente2=Cliente("Ana")
        self.cliente3=Cliente("Luis")

    def operar(self):
        self.cliente1.depositar(100)
        self.cliente2.depositar(150)
        self.cliente3.depositar(200)
        self.cliente3.extraer(150)

    def depositos_totales(self):

total=self.cliente1.retornar_importe()+self.cliente2.retornar_importe()+self.cliente3.retornar_i
mporte()
        print("Total de dinero en el banco", total)
        self.cliente1.imprimir()
        self.cliente2.imprimir()
        self.cliente3.imprimir()

# Bloque principal

banco1=Banco()
banco1.operar()
banco1.depositos_totales()
```

Guardamos el proyecto con el nombre de Ejercicio196.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
    Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
    AMD64)] on win32
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============= RESTART: D:/Curso completo de Python/ejercicio196.py =============
    Total de dinero en el banco 300
    Juan Tiene depositado la suma de 100
    Ana Tiene depositado la suma de 150
    Luis Tiene depositado la suma de 50
>>>
```

En este programa podemos observar que hay dos clases y una clase colabora con la otra.

Capítulo 197.- Colaboración de clases – 2

Problema:

Plantear un programa que permita jugar a los dados. Las reglas de juego son:

Se tiran tres dados si los tres salen con el mismo valor mostrar un mensaje que "Gano", sino "Perdió".

Lo primero que hacemos es identificar las clases:

Podemos identificar la clase Dado y la clase JuegoDeDados.

Luego los atributos y los métodos de cada clase:

Dado
- atributos
  - valor
- métodos
  - tirar
  - imprimir
  - retornar_valor

JuegoDeDados
- atributo
  - 3 Dados (3 objetos de la clase Dado)
- métodos
  - __init__
  - jugar

Vamos a programar:

ejercicio197.py - D:\Curso completo de Python\ejercicio197.py (3.11.1)

File Edit Format Run Options Window Help

```python
import random

class Dado:

    def tirar(self):
        self.valor=random.randint(1,6)

    def imprimir(self):
        print("Valor del dado:", self.valor)

    def retornar_valor(self):
        return self.valor

class JuegoDeDados:

    def __init__(self):
        self.dado1=Dado()
        self.dado2=Dado()
        self.dado3=Dado()

    def jugar(self):
        self.dado1.tirar()
        self.dado1.imprimir()
        self.dado2.tirar()
        self.dado2.imprimir()
        self.dado3.tirar()
        self.dado3.imprimir()
```

```
28          if (self.dado1.retornar_valor()==self.dado2.retornar_valor() and  \
29              self.dado1.retornar_valor()==self.dado3.retornar_valor()):
30              print("Ganó")
31          else:
32              print("Perdió")
33
34
35
36  # Bloque principal
37  juego1=JuegoDeDados()
38  juego1.jugar()
39
```

Ln: 40 Col: 0

Guardamos el programa con el nombre de Ejercicio197.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio197.py =============
Valor del dado: 2
Valor del dado: 1
Valor del dado: 5
Perdió
>>>
```

Como podréis comprobar la coincidencia de los tres dados con el mismo número no es muy fácil que salga.

Vamos a realizar unas modificaciones para que el programa nos diga cuantas tiradas ha realzado para poder ganar.

```
ejercicio197.py - D:\Curso completo de Python\ejercicio197.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1   import random
2
3   class Dado:
4
5       def tirar(self):
6           self.valor=random.randint(1,6)
7
8       def imprimir(self):
9           print("Valor del dado:", self.valor)
10
11      def retornar_valor(self):
12          return self.valor
13
14  class JuegoDeDados:
15
16      def __init__(self):
17          self.dado1=Dado()
18          self.dado2=Dado()
19          self.dado3=Dado()
20
```

```
21      def jugar(self):
22          self.dado1.tirar()
23          self.dado1.imprimir()
24          self.dado2.tirar()
25          self.dado2.imprimir()
26          self.dado3.tirar()
27          self.dado3.imprimir()
28          if (self.dado1.retornar_valor()==self.dado2.retornar_valor() and  \
29              self.dado1.retornar_valor()==self.dado3.retornar_valor()):
30              print("Ganó")
31          else:
32              print("Perdió")
33
34
35      def jugar_hasta_ganar(self):
36          cant=1
37          self.jugar()
38          while not(self.dado1.retornar_valor()==self.dado2.retornar_valor() and \
39                    self.dado1.retornar_valor()==self.dado3.retornar_valor()):
40              self.jugar()
41              cant=cant+1
42
43          print("Para ganar he tenido de tirar la cantidad de", cant, "veces.")
44
45
46  # Bloque principal
47  juego1=JuegoDeDados()
48  juego1.jugar_hasta_ganar()
49
                                                                Ln: 50   Col: 0
```

Si lo ejecutamos observaremos el siguiente resultado:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:\Curso completo de Python\ejercicio197.py =============
Valor del dado: 3
Valor del dado: 1
Valor del dado: 1
Perdió
```

. . .

```
Valor del dado: 3
Valor del dado: 1
Valor del dado: 3
Perdió
Valor del dado: 6
Valor del dado: 5
Valor del dado: 4
Perdió
Valor del dado: 3
Valor del dado: 3
Valor del dado: 3
Ganó
Para ganar he tenido de tirar la cantidad de 41 veces.
>>>
                                                                Ln: 170   Col: 0
```

Capítulo 198.- Colaboración de clases – 3

Problema propuesto:

Plantear una clase Club y otra clase Socio.
La clase Socio debe tener lo siguientes atributos: nombre y la antigüedad den el club (en años).
El método __init__ de la clase Socio pedir la carga por teclado del nombre y su antigüedad.
La clase Club debe tener como atributos 3 objetos de la clase Socio.
Definir una responsabilidad para imprimir el nombre del socio con mayor antigüedad en el club.

Vamos a programar:

ejercicio198.py - D:/Curso completo de Python/ejercicio198.py (3.11.1)

File Edit Format Run Options Window Help

```python
class Socio:

    def __init__(self):
        self.nombre=input("Ingrese el nombre: ")
        self.antiguedad=int(input("Ingrese los años de antigüedad: "))

    def retorna_valor(self):
        return self.antiguedad

class Club:

    def __init__(self):
        self.socio1=Socio()
        self.socio2=Socio()
        self.socio3=Socio()

    def responsabilidad(self):
        if self.socio1.retorna_valor()>self.socio2.retorna_valor() \
           and self.socio1.retorna_valor()>self.socio3.retorna_valor():
            print("Socio responsable:", self.socio1.nombre)
        else:
            if self.socio2.retorna_valor()>self.socio3.retorna_valor():
                print("Socio responsable:", self.socio2.nombre)
            else:
                print("Socio responsable:", self.socio3.nombre)

# Bloque principal
club1=Club()
club1.responsabilidad()
```

Ln: 32 Col: 0

Guardamos el programa con el nombre de Ejercicio198.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio198.py =============
Ingrese el nombre: Juan
Ingrese los años de antigüedad: 5
Ingrese el nombre: Antonio
Ingrese los años de antigüedad: 9
Ingrese el nombre: Luis
Ingrese los años de antigüedad: 3
Socio responsable: Antonio
>>>
```

Capítulo 199.- Herencia – 1

Vimos en el concepto anterior que dos clases pueden estar relacionadas por la colaboración. Ahora veremos otro tipo de relaciones entre clases, que es la Herencia.

La herencia significa que se puede crear nuevas clases partiendo de clases existentes, que tendrán todos los atributos y los métodos de su 'superclase' o 'clase padre' y además se le podrán añadir otros atributos y métodos propios.

Clase padre

Clase de la que descienden o deriva una clase. Las clases hijas (descendientes) heredan (incorporan) automáticamente los atributos y métodos de la clase padre.

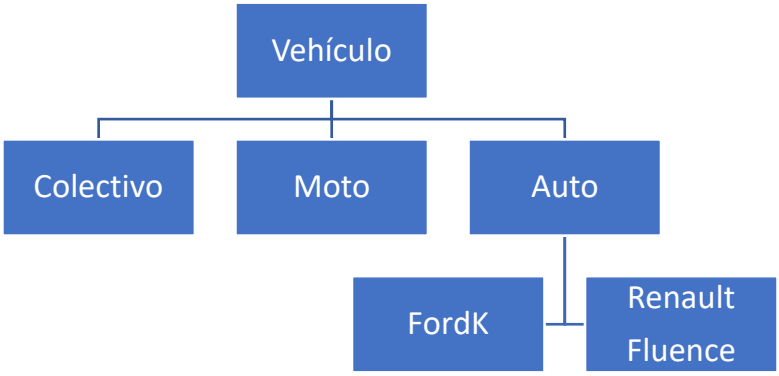
Subclase

Clase descendiente de otra. Hereda automáticamente los atributos y métodos de su superclase. Es una especialización de otra clase.

Admiten la definición de nuevos atributos y métodos para aumentar la especialización de la clase.

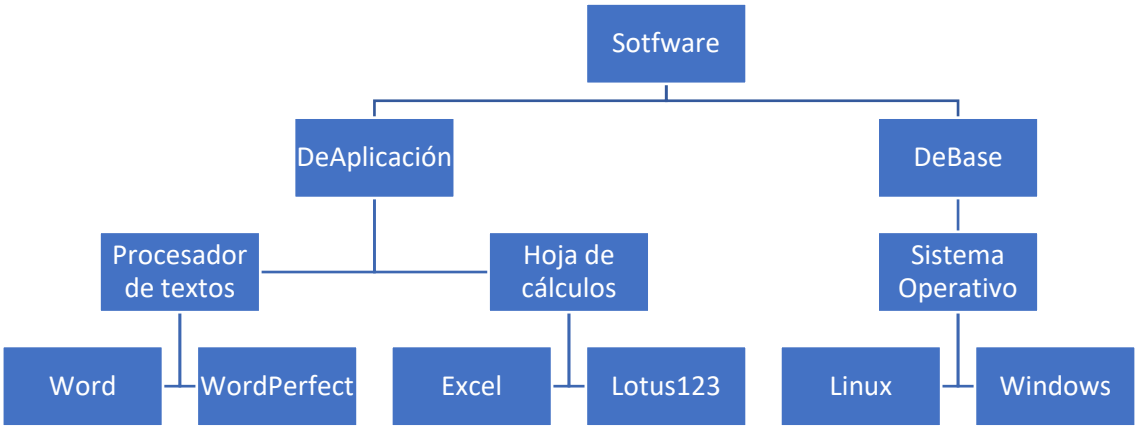
Veamos algunos ejemplos teóricos de herencia:

1) Imaginemos al clase vehículo. Qué clases podrían derivar de ella?



Siempre hacia abajo en la jerarquía hay una especialización (las subclases añaden nuevos atributos y métodos.

2) Imaginemos la clase Software. Qué clases podrían derivar de ella?



El primer tipo de relación que habíamos visto entre dos clases, es la de colaboración. Recordemos que es cuando una clase contiene objetos de otra clase como atributo.

Cuando la relación entre dos clases es del tipo "…tiene un…" o "…es parte de…", no debemos implementar herencia. Estamos frente a una relación de colaboración de clases no de herencia.

Si tenemos una ClaseA y otra ClaseB y notamos que entre ellas existe kuna relación de tipo "…tiene un…", no debe implementarse herencias sino declarar en la clase ClaseA un atributo de la clase ClaseB.

Por ejemplo: tenemos una clase Auto, una clase Rueda y una clase Volante. Vemos que la relación entre ellas es: Auto "…Tiene 4…" Ruedas, Volante "…es parte de…" Auto, pero la clase Auto no debe derivar de Rueda ni Volante del Auto porque la relación no es de tipo-subtipo sino de colaboración. Debemos declarar en la clase Auto 4 atributos de tipo Rueda y 1 de tipo Volante.

Luego si vemos que dos clases responden a la pregunta ClaseA "…es un…" ClaseB es posible que haya una relación de herencia.

Por ejemplo:

Auto "es un" Vehículo
Circulo "es una" Figura
Mouse "es un" DispositivoEntrada
Suma "es una" Operación

Problema:

Plantear kuna clase Persona que contenga dos atributos. nombre y edad. Definir como responsabilidades la carga por teclado y su impresión.

En el bloque principal del programa definir un objeto de la clase persona y llamar a sus métodos.

Declarar una segunda clase llamada Empleado que hereda de la clase Persona y agregue un atributo sueldo y muestre si debe pagar impuestos (sueldo superior a 3000).

También en el bloque principal del programa crear un objeto de la clase Empleado.

Vamos a programar:

ejercicio199.py - D:/Curso completo de Python/ejercicio199.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Persona:
2
3      def __init__(self):
4          self.nombre=input("Ingrese el nombre:")
5          self.edad=int(input("Engrese la edad:"))
6
7      def imprimir(self):
8          print("Nombre:", self.nombre)
9          print("Edad:", self.edad)
10
```

En esta parte del programa creamos la clase Persona que contiene el método __init__(self) y el método imprimir(self), hasta ahora no hay nada nuevo.

```
11 class Empleado(Persona):
12
13     def __init__(self):
14         super().__init__()
15         self.sueldo=float(input("Ingrese sueldo:"))
16
17     def imprimir(self):
18         super().imprimir()
19         print("Sueldo:", self.sueldo)
20
21     def paga_impuestos(self):
22         if self.sueldo>3000:
23             print(self.nombre,"debe pagar impuestos")
24         else:
25             print(self.nombre,"No debe pagar impuestos")
26
27 # Bloque principal
28 empleado1=Empleado()
29 empleado1.imprimir()
30 empleado1.paga_impuestos()
```

Ln: 33 Col: 0

11.- En la segunda parte del programa creamos una clase llamada Empleados que heredará todas sus atributos y métodos de la clase Persona.

13.- En el método __init__(self) le decimos que primero ejecute el método de la clase Persona (Padre) y a continuación ejecute el de su propia clase, con este ejemplo vemos como la clase Empleados además de preguntar por el nombre y edad, también pregunta por su sueldo.

17.- en el método imprimir(self) antes de ejecutar su propio método ejecuta el método de la clase padre y a continuación su propio método, por este motivo imprime el nombre y la edad del método padre además de imprimir el sueldo que se encuentra en su propio método.

18.- Con el método paga_impuestos(self) comprobamos si el sueldo que percibe es superior a 3000 o no para decirle que si supera los 3000 tendrá que pagar impuestos.

En el bloque principal creamos una instancia de la clase Empleado con el nombre de empleado1 y a continuación se ejecuta automáticamente el método __init__(self) y después llamamos a los métodos imprimir() y paga_impuestos().

Vamos a guardar el programa con el nombre de Ejercicio199.py y a continuación lo ejecutamos.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio199.py =============
Ingrese el nombre:Juan
Engrese la edad:33
Ingrese sueldo:2500
Nombre: Juan
Edad: 33
Sueldo: 2500.0
Juan No debe pagar impuestos
>>>
```

En este primero hemos ingresado un sueldo inferior a 3000, no deberá pagar impuestos.

```
============= RESTART: D:/Curso completo de Python/ejercicio199.py =============
Ingrese el nombre:Luis
Engrese la edad:28
Ingrese sueldo:3500
Nombre: Luis
Edad: 28
Sueldo: 3500.0
Luis debe pagar impuestos
>>>
```

En este segundo hemos ingresado un sueldo superior a 3000, deberá pagar impuestos.

Capítulo 200.- Herencia – 2

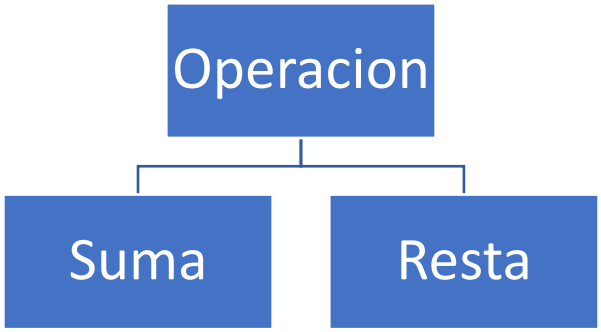
Problema:

Ahora planteamos otro problema empleando herencia. Supongamos que necesitamos implementar dos clases que llamaremos Suma y Resta. Cada clase tiene como atributo valor1, valor2 y resultado.

Los métodos a definir son cargar1(que inicializa el atributo valor1), carga2 (que inicializa el atributo valor2), operar(que en el caso de la clase "Suma" suma los dos atributos y en el caso de la clase "Resta" hace la diferencia entre valor1 y valor2), y otro método mostrar_resultado.

Si analizamos ambas clases encontramos que muchos atributos y métodos son idénticos. En estos casos es bueno definir una clase padre que agregue dichos atributos y responsabilidades comunes.

La relación de herencia que podemos disponer para este problema es:



Solamente el método operar es distinto para las clases Suma y Resta (esto hace que no lo podemos disponer en la clase Operación en principio), luego los métodos cargar1 y cargar2 y mostrar resultado son idénticos en las dos clases, esto hace que podemos disponerlos en la clase Operación. Lo mismo los atributos valor1, valor2 y resultado se definirán en la clase padre Operacion.

Vamos a programar:

ejercicio200.py - D:/Curso completo de Python/ejercicio200.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Operacion:
2
3      def __init__(self):
4          self.valor1=0
5          self.valor2=0
6          self.resultado=0
7
8      def cargar1(self):
9          self.valor1=int(input("Ingrese el primer valor: "))
10
11     def cargar2(self):
12         self.valor2=int(input("Ingrese el segundo valor: "))
13
14     def mostrar_resultado(self):
15         print(self.resultado)
16
17     def operar(self):
18         pass
19
20
```

El método operar, tiene la instrucción pass, esto quiere decir que no hace nada, las clases Suma y resta tienen el mismo método que se sobrescribirá.

```
21 class Suma(Operacion):
22
23     def operar(self):
24         self.resultado=self.valor1+self.valor2
25
26 class Resta(Operacion):
27
28     def operar(self):
29         self.resultado=self.valor1-self.valor2
30
31 # Bloque principal
32
33 suma1=Suma()
34 suma1.cargar1()
35 suma1.cargar2()
36 suma1.operar()
37 suma1.mostrar_resultado()
38
39 resta1=Resta()
40 resta1.cargar1()
41 resta1.cargar2()
42 resta1.operar()
43 resta1.mostrar_resultado()
```

Cada clase tiene su propio método operar ya que en cada clase no realiza lo mismo, en una suma y en la otra resta.

Ln: 30 Col: 0

Guardamos el programa con el nombre de Ejercicio200.py y a continuación lo ejecutamos.

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio200.py =============
Ingrese el primer valor: 10
Ingrese el segundo valor: 5
15
Ingrese el primer valor: 10
Ingrese el segundo valor: 7
3
>>>
```

Capítulo 201.- Herencia – 3

Problema propuesto:

Declarar una clase Cuenta y dos subclases CajaAhorra y PlazoFijo. Definir los atributos y métodos comunes entre una caja de ahorro y un plazo fijo y agruparlos en la clase Cuenta.

Una caja de ahorro y un plazo fijo tienen un nombre de titular y un importe. Un plazo fijo añade un plazo de imposición en días y una tasa de interés. Hacer que la caja de ahorro no genera intereses.

En el bloque principal del programa definir un objeto de la clase CajaAhorro y otro de la clase PlazoFijo.

Vamos a programar:

ejercicio201.py - D:/Curso completo de Python/ejercicio201.py (3.11.1)

File Edit Format Run Options Window Help

```python
class Cuenta:
    def __init__(self, titular, importe):
        self.titular=titular
        self.importe=importe

    def imprimir(self):
        print("Titular:", self.titular)
        print("Importe:", self.importe)

class CajaAhorro(Cuenta):

    def __init__(self, titular, importe):
        super().__init__(titular, importe)

    def imprimir(self):
        print("------------------------")
        print("Cuenta de caja de ahorro")
        print("------------------------")
        super().imprimir()

class PlazoFijo(Cuenta):

    def __init__(self, titular, importe, plazo, interes):
        super().__init__(titular, importe)
        self.plazo=plazo
        self.interes=interes

    def imprimir(self):
        print("--------------------")
        print("Cuenta de plazo fijo")
        print("--------------------")
        super().imprimir()
        print("Plazo en días:", self.plazo)
        print("Interés:", self.interes)

# Bloque principal
cajadeahorro1=CajaAhorro("Juan", 1300)
cajadeahorro1.imprimir()

plazofijo1=PlazoFijo("Ana", 1000, 30, 0.75)
plazofijo1.imprimir()
```

Ln: 45 Col: 0

Guardamos el programa con el nombre de Ejercicio201.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio201.py =============
------------------------
Cuenta de caja de ahorro
------------------------
Titular: Juan
Importe: 1300
--------------------
Cuenta de plazo fijo
--------------------
Titular: Ana
Importe: 1000
Plazo en días: 30
Interés: 0.75
>>>
```

Ahora vamos a añadir otro método para calcular el beneficio:

```
ejercicio201.py - D:/Curso completo de Python/ejercicio201.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
class Cuenta:
    def __init__(self, titular, importe):
        self.titular=titular
        self.importe=importe

    def imprimir(self):
        print("Titular:", self.titular)
        print("Importe:", self.importe)

class CajaAhorro(Cuenta):

    def __init__(self, titular, importe):
        super().__init__(titular, importe)

    def imprimir(self):
        print("------------------------")
        print("Cuenta de caja de ahorro")
        print("------------------------")
        super().imprimir()

class PlazoFijo(Cuenta):

    def __init__(self, titular, importe, plazo, interes):
        super().__init__(titular, importe)
        self.plazo=plazo
        self.interes=interes

    def imprimir(self):
        print("--------------------")
        print("Cuenta de plazo fijo")
        print("--------------------")
        super().imprimir()
        print("Plazo en días:", self.plazo)
        print("Interés:", self.interes)
```

```
35
36      def ganancia(self):
37          gan=self.importe*self.interes/100
38          print("El importe por intereses es:", gan)
39
40
41  # Bloque principal
42  cajadeahorro1=CajaAhorro("Juan", 1300)
43  cajadeahorro1.imprimir()
44
45  plazofijo1=PlazoFijo("Ana", 1000, 30, 0.75)
46  plazofijo1.imprimir()
47  plazofijo1.ganancia()
```

Ln: 49 Col: 0

Este método solo funciona a los objetos de la clase PlazoFijo.

Capítulo 202.- Variables de clase – 1

Hemos visto como definimos atributos en una clase anteponiendo la palabra clave self:

```
class Persona:

        def __init__(self, nombre):
                self.nombre=nombre
```

Los atributos son independientes por cada objeto o instancia de la clase, es decir si definimos tres objetos de la clase Persona, todas las personas tienen un atributo nombre pero cada uno tiene un valor independiente:

```
class Persona:

        def __init__(self, nombre):
                self.nombre=nombre

# Bloque principal

persona1=Persona("Juan")
persona2=Persona("Ana")
persona3=Persona("Luis")

print(persona1.nombre)   # Juan
print(persona2.nombre)   # Ana
print(persona3.nombre)   # Luis
```

En algunas situaciones necesitamos almacenar datos que sean compartidos por todos los objetos de dicha clase, en esas situaciones debemos emplear variables de clase.

Para definir una variable de clase lo hacemos dentro de la clase pero fuera de sus métodos.

```
class Persona:

        variable=20
        def __init__(self, nombre):
                self.nombre=nombre

# Bloque principal

persona1=Persona("Juan")
persona2=Persona("Ana")
persona3=Persona("Luis")

print(persona1.nombre)   # Juan
print(persona2.nombre)   # Ana
print(persona3.nombre)   # Luis

print(persona1.variable)   # 20
Persona.variable=5
print(persona2.variable)   # 5
```

Se reserva solo un espacio para la variable "variable", independientemente que se definan muchos objetos de la clase Persona. La variable "variable" es compartida por todos los objetos.

persona1, persona2 y persona3.

Para modificar la variable de clase hacemos referencia al nombre de la clase y seguidamente el nombre de la variable:

Persona.variable=5

Problema:

Definir una clase Cliente que almacene un código de cliente y un nombre.
En la clase Cliente definir una variable de clase de tipo lista que almacene todos los código de clientes que tienen suspendidas sus cuentas corrientes.
Imprimir por pantalla todos los datos de clientes y el estado que se encuentra su cuenta corriente.
Vamos a programar:

ejercicio202.py - D:/Curso completo de Python/ejercicio202.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Cliente:
2      suspendidos=[]
3
4      def __init__(self, codigo, nombre):
5          self.codigo=codigo
6          self.nombre=nombre
7
8      def imprimir(self):
9          print("Código:", self.codigo)
10         print("Nombre:", self.nombre)
11         self.esta_suspendido()
12
13     def esta_suspendido(self):
14         if self.codigo in Cliente.suspendidos:
15             print("Esta suspendido")
16         else:
17             print("No esta suspendido")
18         print("____________________________")
19
20     def suspender(self):
21         Cliente.suspendidos.append(self.codigo)
22
23 # Bloque principal
24 cliente1=Cliente(1, "Carlos")
25 cliente2=Cliente(2, "Ana")
26 cliente3=Cliente(3, "Pedro")
27 cliente4=Cliente(4, "Luis")
28
29 cliente2.suspender()
30 cliente3.suspender()
31
32 cliente1.imprimir()
33 cliente2.imprimir()
34 cliente3.imprimir()
35 cliente4.imprimir()
```

Ln: 37 Col: 0

En la línea 1 creamos una clase llamada Cliente.

En la línea 2 definimos una variable de clase tipo lista, llamada sus suspendidos.

En la línea 4 definimos el método __init__, para definir los atributos de la clase Cliente que son codigo y nombre.

En la línea 8 definimos el método imprimir para mostrar en pantalla el contenido de los campos código y nombre del objeto que lo llama.

En la línea 13 el método esta_suspendido comprueba si el código del objeto que lo llama está en la lista suspendidos en caso afirmativo mostrará el mensaje "Está suspendido" de lo contrario mostrará el mensaje "No está suspendido".

En la línea 20 agregamos a la lista llamada suspendidos el código del objeto que lo llama.

Desde el bloque principal instanciamos los objetos cliente1, cliente2, cliente3 y cliente4 a la clase Cliente.

El cliente2 y cliente3 agregamos sus códigos a la lista suspendidos.

Por último de los objetos cliente1, cliente2, cliente3 y cliente4 mostramos sus códigos, nombres y si están o no suspendidos.

Guardamos el programa con el nombre de Ejercicio202.py y a continuación ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio202.py =============
Código: 1
Nombre: Carlos
No esta suspendido
_____________________________
Código: 2
Nombre: Ana
Esta suspendido
_____________________________
Código: 3
Nombre: Pedro
Esta suspendido
_____________________________
Código: 4
Nombre: Luis
No esta suspendido
_____________________________
>>>
```

De este modo tenemos una lista que almacena todos los código de los objetos de tipo Cliente que están suspendidos, de este modo podemos tener tantos objetos de la clase Cliente que con solo una variable de tipo lista podremos almacenar los que están suspendidos.

Capítulo 203.- Variables de clase – 2

Problema propuesto:

Plantear una clase llamada Jugador.
Definir en la clase Jugador los atributos nombre y puntuación, y los métodos __init__, imprimir y pasar_tiempo (que debe reducir en uno la variable de clase).
Declarar dentro de la clase Jugador una variable de clase que indique cuantos minutos falta para el fin de juego (inicializarla con el valor 30).
Definir en el bloque principal dos objetos de la clase Jugador.
Reducir la variable de clase de uno en uno hasta llegar a cero.

Vamos a programar:

ejercicio203.py - D:/Curso completo de Python/ejercicio203.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Jugador:
2      minutos_faltan=30
3
4      def __init__(self, nombre, puntuacion):
5          self.nombre=nombre
6          self.puntuacion=puntuacion
7
8      def imprimir(self):
9          print("Nombre:", self.nombre, "Puntuación:", self.puntuacion)
10         print("Fin de jueego en", Jugador.minutos_faltan, "minutos")
11
12     def pasar_tiempo(self):
13         Jugador.minutos_faltan=Jugador.minutos_faltan-1
14
15 # Bloque principal
16 jugador1=Jugador("Juan", 10)
17 jugador2=Jugador("Luis", 8)
18 while Jugador.minutos_faltan>=0:
19     jugador1.imprimir()
20     jugador2.imprimir()
21     jugador1.pasar_tiempo()
22
```

Ln: 24 Col: 0

Guardamos el programa con el nombre de Ejercicio203.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio203.py =============
Nombre: Juan Puntuación: 10
Fin de jueego en 30 minutos
Nombre: Luis Puntuación: 8
Fin de jueego en 30 minutos
Nombre: Juan Puntuación: 10
Fin de jueego en 29 minutos
Nombre: Luis Puntuación: 8
Fin de jueego en 29 minutos
```

. . .

```
Nombre: Juan Puntuación: 10
Fin de jueego en 0 minutos
Nombre: Luis Puntuación: 8
Fin de jueego en 0 minutos
>>>
```

Ln: 129 Col: 0

Capítulo 204.- Método especial __str__ - 1

Podemos hacer que se ejecute un método definido por nosotros cuando pasamos un objeto a la función print o cuando llamamos a la función str (convertir a string).

¿Qué sucede cuando llamamos a la función print y le pasamos como parámetros un objeto?

```
class Persona:
        def __init__(self, nom, ape):
                self.nombre=nom
                self.apellido=ape

persona1=Persona("José", "Rodriguez")

print(personal)
```

Nos muestra algo parecido a esto:

```
<__main__.Persona object at 0x03E99C90>
```

Python nos permite redefinir el método que se debe ejecutar. Esto se hace definiendo en la clase el método especial __str__.

En el ejemplo anterior si queremos que se muestre el nombre y el apellido separados por coma cuando llamemos a la función print el código que debemos implementar es el siguiente:

```
class Persona:
        def __init__(self, nom, ape):
                self.nombre=nom
                self.apellido=ape

        def __str__(self):
                cadena=self.nombre+","+self.apellido
                return cadena

personal1=Persona("José", "Rodriguez")
print(persona1)
```

Como vemos debemos implementar el método __str__ y retornar un string, este luego será el que imprime la función print:

```
        def __str__(self):
                cadena=self.nombre+", "+self.apellido
                return cadena
```

Esta característica definida en Python nos permite crear programas muy legibles y flexibles.

El método __str__ también se ejecutará si llamamos a la función str y pulsamos como parámetro un objeto que tiene definido dicho método:

```
class Persona:

        def __init__(self, nom, ape):
                self.nombre=nom
                self.apellido=ape
```

```
    def __str__(self):
        cadena=self.nombre+","+self.apellido
        return cadena

# Bloque principal
persona1=Persona("José", "Rodriquez")
persona2=Persona("Ana", "Martinez")
print(str(persona)+"-"str(persona2))   # José, Rodriguez – Ana, Martinez
```

Vamos a programar:

ejercicio204.py - D:/Curso completo de Python/ejercicio204.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Persona:
2
3      def __init__(self, nom, ape):
4          self.nombre=nom
5          self.apellido=ape
6
7      def __str__(self):
8          cadena=self.nombre+","+self.apellido
9          return cadena
10
11 # Bloque principal
12 personal=Persona("José", "Rodriquez")
13 persona2=Persona("Ana", "Martinez")
14 print(str(personal)+"-"+str(persona2))    # José, Rodriguez - Ana, Martinez
15
```

Guardamos el programa con el nombre de Ejercicio204.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio204.py =============
José,Rodriquez-Ana,Martinez
>>>
```

También podemos poner lo siguiente:

```
print(personal)
print(str(personal))
```

El resultado será:

```
José,Rodriquez
José,Rodriquez
```

Resumiendo el método __str__ hace que cuando imprimamos un objeto este lo mostrará según el formato que retorne dicho método:

```
def __str__(self):
    cadena=self.nombre+","+self.apellido
    return cadena
```

Capítulo 205.- Método especial __str__ - 2

Problema:

definir una clase llamada Punto con dos atributos x e y.
Crearle el método especial __str__ para retornar un string con autoformato (x,y).

Vamos a programar:

```
ejercicio205.py - D:/Curso completo de Python/ejercicio205.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1  class Punto:
2
3      def __init__(self, x, y):
4          self.x=x
5          self.y=y
6
7      def __str__(self):
8          cadena="("+str(self.x)+","+str(self.y)+")"
9          return cadena
10
11 # Bloque principal
12 punto1=Punto(8,5)
13 print(punto1)
14
15 punto2=Punto(3,10)
16 print(punto2)
17
```

Guardamos el programa con el nombre de Ejercicio205.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio205.py =============
(8,5)
(3,10)
>>>
```

Si borramos el método __str__, este será el resultado:

```
<__main__.Punto object at 0x000001D0A2A342D0>
<__main__.Punto object at 0x000001D0A2A36790>
```

Para concatenar varios objetos:

```
print(str(punto1)+"-"+str(punto2))
```

```
(8,5)-(3,10)
```

Capítulo 206.- Método especial __str__ - 3

Problema:

Declarar una clase llamada Familia. Definir como atributos el nombre del padre, madre y una lista con los nombres de los hijos.

Definir el método especial __srt__ que retorne un string con el combre del padre, lamadre y de todos sus hijos.

Vamos a programar:

ejercicio206.py - D:/Curso completo de Python/ejercicio206.py (3.11.1)

File Edit Format Run Options Window Help

```
class Familia:

    def __init__(self, padre, madre, hijos=[]):
        self.padre=padre
        self.madre=madre
        self.hijos=hijos

    def __str__(self):
        famili=("Padre:"+self.padre+"\n")
        famili=famili+("Madre:"+self.madre+"\n")
        famili=famili+("---------------------------"+"\n")
        famili=famili+("Hijos"+"\n")
        famili=famili+("---------------------------"+"\n")
        for hijo in self.hijos:
            famili=famili+hijo+"\n"
        return famili

# Bloque principal
familia1=Familia("Pablo", "Ana", ["Carlos", "Verónica"])
familia2=Familia("Jorge", "Carla")
familia3=Familia("Luis", "María", ["Marcos"])

print(familia1)
print(familia2)
print(familia3)
```

Guardamos el programa con el nombre Ejercicio206.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio206.py =============
Padre:Pablo
Madre:Ana
---------------------------
Hijos
---------------------------
Carlos
Verónica

Padre:Jorge
Madre:Carla
---------------------------
Hijos
---------------------------
```

```
Padre:Luis
Madre:María
----------------------------
Hijos
----------------------------
Marcos
```

Capítulo 207.- Método especial __str__ - 4

Problema propuesto:

Desarrollar un programa que implemente una clase llamada Jugador.
Definir como atributos su nombre y puntuación.
Codificar el método especial __str__ que retorne el nombre y si es principiante (menos de 1000 puntos) o experto (1000 puntos o más puntos)

Vamos a programar:

ejercicio207.py - D:/Curso completo de Python/ejercicio207.py (3.11.1)

File Edit Format Run Options Window Help

```
class Jugador:

    def __init__(self, nombre, puntuacion):
        self.nombre=nombre
        self.puntuacion=puntuacion

    def __str__(self):
        cadena=("El jugador: " + self.nombre +"\n")
        if self.puntuacion < 1000:
            cadena=cadena+("se encuentra en la modalidad de principiante \n")
        else:
            cadena=cadena+("se encuentra en la modalidad de experto \n")
        cadena=cadena+"--------------------------------------------------"
        return cadena

# Bloque principal
jugador1=Jugador("Carlos", 950)
print(jugador1)

jugador2=Jugador("Luis", 1050)
print(jugador2)

```

Guardamos el programa con el nombre de Ejercicio207.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio207.py =============
El jugador: Carlos
se encuentra en la modalidad de principiante
--------------------------------------------------
El jugador: Luis
se encuentra en la modalidad de experto
--------------------------------------------------
>>>
```

Capítulo 208.- Redefinición de los operadores matemáticos con objetos – 1

Python nos permite redefinir los operadores matemáticos cuando planteamos una clase.

Los métodos especiales que debemos implementar son los siguientes:

Para el operador (+):

__add__(self, objeto2)

Para el operador(-):

__sub__(self, objeto2)

Para el operador (*):

__mul__(self, objeto2)

Para el operador (//):

__floordiv__(self, objeto2)

Para el operador (/):

__div__(self, objeto2)

Problema:

Veamos con un ejemplo la sintaxis para redefinir el operador (+).
Creamos una clase Cliente de un banco y redefinimos el operador + para que nos retorne la suma de los depósitos de los dos clientes que estamos sumando.

ejercicio208.py - D:/Curso completo de Python/ejercicio208.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Cliente:
2
3      def __init__(self, nombre, importe):
4          self.nombre=nombre
5          self.importe=importe
6
7      def __add__(self, objeto2):
8          suma=self.importe+objeto2.importe
9          return suma
10
11 # Bloque principal
12 cliente1=Cliente("Ana", 1200)
13 cliente2=Cliente("Juan", 1000)
14 print("Total depositado por los dos clientes")
15 print(cliente1+cliente2)
16
```

En la línea 15 cuando sumamos el importe de dos objetos de tipo Cliente y agregamos el operador + el programa busca el método __add__ con self hacemos referencia en este elemplo a cliente1 y con objeto2 hacemos referencia a cliente2.

Lo guardaremos con el nombre de Ejercicio208.py y a continuación lo ejecutamos:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio208.py =============
Total depositado por los dos clientes
2200
>>>
```

Acotaciones:

Es muy importante tener en cuenta que debemos redefinir un operador matemático siempre y cuando haga nuestro programa más legible.
Debemos redefinir los operadores +,-,*,/, etc. solo para las clases que con solo leer su nombre el programador intuya que operación implementaría dicho operador.
Vamos a programar:

Un ejemplo con la resta:

```
ejercicio208.py - D:/Curso completo de Python/ejercicio208.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1  class Cliente:
2
3      def __init__(self, nombre, importe):
4          self.nombre=nombre
5          self.importe=importe
6
7      def __sub__(self, objeto2):
8          suma=self.importe-objeto2.importe
9          return suma
10
11 # Bloque principal
12 cliente1=Cliente("Ana", 1200)
13 cliente2=Cliente("Juan", 1000)
14 print("La diferencia de depositos está en")
15 print(cliente1-cliente2)
16
```

Este será el resultado:

```
IDLE Shell 3.11.1
File  Edit  Shell  Debug  Options  Window  Help
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio208.py =============
La diferencia de depositos está en
200
>>>
```

Capítulo 209.- Redefinición de los operadores matemáticos con objetos – 2

Problema una clase llamada Lista, que permita pasar el método __init__ una lista de valores enteros.

Redefinir los operadores +,-,* y // con respecto a un valor entero.

Vamos a programar:

ejercicio209.py - D:/Curso completo de Python/ejercicio209.py (3.11.1)

File Edit Format Run Options Window Help

```
class Lista:

    def __init__(self, lista):
        self.lista=lista

    def imprimir(self):
        print(self.lista)

    def __add__(self, entero):
        nueva=[]
        for x in range(len(self.lista)):
            nueva.append(self.lista[x]+entero)
        return nueva

    def __sub__(self, entero):
        nueva=[]
        for x in range(len(self.lista)):
            nueva.append(self.lista[x]-entero)
        return nueva

    def __mul__(self, entero):
        nueva=[]
        for x in range(len(self.lista)):
            nueva.append(self.lista[x]*entero)
        return nueva

    def __floordiv__(self, entero):
        nueva=[]
        for x in range(len(self.lista)):
            nueva.append(self.lista[x]//entero)
        return nueva

# Bloque principal
lista1=Lista([20,33,40])
lista1.imprimir()
print(lista1+10)
print(lista1-10)
print(lista1*10)
print(lista1//2)
```

Ln: 41 Col: 0

Guardamos el programa con el nombre de Ejercicio209.py y a continuación lo ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: D:/Curso completo de Python/ejercicio209.py =============
[20, 33, 40]
[30, 43, 50]
[10, 23, 30]
[200, 330, 400]
[10, 16, 20]
>>>
```

Capítulo 210.- Redefinición de los operadores relacionales con objetos – 1

Python también nos permite redefinir los operadores relacionales cuando planteamos una clase.

Los métodos especiales que podemos implementar son los siguientes:

Para el operador (==):
__eq__(self, objeto2)

Para el operador (!=):
__ne__(self, objeto2)

Para el operador (>):
__gt__(self, objeto2)

Para el operador (>=):
__ge__(self, objeto2)

Para el operador (<):
__lt__(self, objeto2)

Para el operador (<=):
__le__(self, objeto)

Es importante recordar que una redefinición de un operador tiene sentido si se ayuda y hace más claro nuestro algoritmo.

Problema:

Crear una clase Persona que tenga como atributo el nombre y la edad.
El operador == retornará verdadero si las dos personas tienen la misma edad, el operador > retornará True si la edad del objeto de la izquierda tiene una edad mayor a la edad del objeto de la derecha del operador >, y así sucesivamente.

Vamos a programar:

ejercicio210_bis.py - D:/Curso completo de Python/ejercicio210_bis.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Persona:
2
3      def __init__(self, nombre, edad):
4          self.nombre=nombre
5          self.edad=edad
6
7      def __eq__(self, objeto2):
8          if self.edad==objeto2.edad:
9              return True
10         else:
11             return False
12
13     def __ne__(self, objeto2):
14         if self.edad!=objeto2.edad:
15             return True
16         else:
17             return False
```

```
18
19 # Bloque principal
20
21 personal=Persona("Juan", 18)
22 persona2=Persona("Luis", 18)
23
24 if (personal==persona2):
25     print("Las dos personas tienen la misma edad")
26 else:
27     print("Las dos personas no tienen la misma edad")
28
29
30 if (personal!=persona2):
31     print("Las dos personas no tienen la misma edad")
32 else:
33     print("Las dos personas tienen la misma edad")
34
```

Ln: 34 Col: 0

Otra forma de hacerlo y obtenemos el mismos resultado:

ejercicio210.py - D:/Curso completo de Python/ejercicio210.py (3.11.1)

File Edit Format Run Options Window Help

```
1 class Persona:
2
3     def __init__(self, nombre, edad):
4         self.nombre=nombre
5         self.edad=edad
6
7     def __eq__(self, objeto2):
8         return self.edad==objeto2.edad
9
10    def __ne__(self, objeto2):
11        return self.edad!=objeto2.edad
12
13 # Bloque principal
14
15 personal=Persona("Juan", 18)
16 persona2=Persona("Luis", 18)
17
18 if (personal==persona2):
19     print("Las dos personas tienen la misma edad")
20 else:
21     print("Las dos personas no tienen la misma edad")
22
23
24 if (personal!=persona2):
25     print("Las dos personas no tienen la misma edad")
26 else:
27     print("Las dos personas tienen la misma edad")
28
```

Ln: 35 Col: 0

Lo Guardamos con el nombre de Ejercicio210.py y a continuación ejecutamos:

IDLE Shell 3.11.1

File Edit Shell Debug Options Window Help

```
Python 3.11.1 (tags/v3.11.1:a7a450f, Dec  6 2022, 19:58:39) [MSC v.1934 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=========== RESTART: D:/Curso completo de Python/ejercicio210_bis.py ===========
Las dos personas tienen la misma edad
Las dos personas tienen la misma edad
>>>
```

Ahora lo vamos a completar con todos los operadores:

ejercicio210.py - D:/Curso completo de Python/ejercicio210.py (3.11.1)

File Edit Format Run Options Window Help

```
class Persona:

    def __init__(self, nombre, edad):
        self.nombre=nombre
        self.edad=edad

    def __eq__(self, objeto2):
        return self.edad==objeto2.edad

    def __ne__(self, objeto2):
        return self.edad!=objeto2.edad

    def __gt__(self, objeto2):
        return self.edad>objeto2.edad

    def __ge__(self, objeto2):
        return self.edad>=objeto2.edad

    def __lt__(self, objeto2):
        return self.edad<objeto2.edad

    def __le__(self, objeto2):
        return self.edad<=objeto2.edad

# Bloque principal

personal=Persona("Juan", 18)
persona2=Persona("Luis", 18)

if (personal==persona2):
    print("Juan y Luis tienen la misma edad")
else:
    if(personal!=persona2):
        print("Juan y Luis no tienen la misma edad")
        if(personal>persona2):
            print("Juan es mayor que Luis")
        else:
            print("Luis es mayor que Juan")
```

Ln: 40 Col: 2

Ves cambiando las edades de Juan y luis y lo ejecutas:

```
personal=Persona("Juan", 18)
persona2=Persona("Luis", 18)

Juan y Luis tienen la misma edad

personal=Persona("Juan", 19)
persona2=Persona("Luis", 18)

Juan y Luis no tienen la misma edad
Juan es mayor que Luis

personal=Persona("Juan", 19)
persona2=Persona("Luis", 22)

Juan y Luis no tienen la misma edad
Luis es mayor que Juan
```

Capítulo 211.- Redefinición de los operadores relacionales con objetos – 2

Problema:

Plantear una clase Rectangulo. Definir dos atributos (ladomenor y ladomayor). Redefinir el operador == de tal forma que tengan en cuenta la superficie del rectángulo. Lo mismo hacer con todos los otros operadores relacionales.

Vamos a programar:

ejercicio211.py - D:/Curso completo de Python/ejercicio211.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Rectangulo:
2
3      def __init__(self, ladomenor, ladomayor):
4          self.ladomenor=ladomenor
5          self.ladomayor=ladomayor
6
7      def __eq__(self, objeto2):
8          superficie1=self.ladomenor*self.ladomayor
9          superficie2=objeto2.ladomenor*objeto2.ladomayor
10         return superficie1==superficie2
11
12     def __ne__(self, objeto2):
13         superficie1=self.ladomenor*self.ladomayor
14         superficie2=objeto2.ladomenor*objeto2.ladomayor
15         return superficie1!=superficie2
16
17     def __gt__(self, objeto2):
18         superficie1=self.ladomenor*self.ladomayor
19         superficie2=objeto2.ladomenor*objeto2.ladomayor
20         return superficie1>superficie2
21
22     def __ge__(self, objeto2):
23         superficie1=self.ladomenor*self.ladomayor
24         superficie2=objeto2.ladomenor*objeto2.ladomayor
25         return superficie1>=superficie2
26
27     def __lt__(self, objeto2):
28         superficie1=self.ladomenor*self.ladomayor
29         superficie2=objeto2.ladomenor*objeto2.ladomayor
30         return superficie1<superficie2
31
32     def __le__(self, objeto2):
33         superficie1=self.ladomenor*self.ladomayor
34         superficie2=objeto2.ladomenor*objeto2.ladomayor
35         return superficie1<=superficie2
36
37 # Bloque principal
38 rectangulo1=Rectangulo(10,20)
39 rectangulo2=Rectangulo(10,10)
40
41 if rectangulo1==rectangulo2:
42     print("El area del primer rectángulo y el sugundo son iguales")
43 else:
44     if rectangulo1!=rectangulo2:
45         print("El area del primer rectángulo y el segundo no son iguales")
46         if rectangulo1>rectangulo2:
47             print("El area del primer rectángulo es mayor al segundo rectángulo")
48         else:
49             print("El area del segundo rectángulo es mayor al primer rectángulo")
50
```

Ln: 51 Col: 0

Guardamos el programa con el nombre de Ejercicio211.py y a continuación lo vamos a ejecutar con distintos valores:

```
rectangulo1=Rectangulo(10,5)
rectangulo2=Rectangulo(5,10)
```

```
El area del primer rectángulo y el sugundo son iguales

rectangulo1=Rectangulo(10,7)
rectangulo2=Rectangulo(5,10)

El area del primer rectángulo y el segundo no son iguales
El area del primer rectángulo es mayor al segundo rectángulo

rectangulo1=Rectangulo(10,5)
rectangulo2=Rectangulo(7,10)

El area del primer rectángulo y el segundo no son iguales
El area del segundo rectángulo es mayor al primer rectángulo
```

Otra forma de hacerlo obteniendo los mismos resultados:

ejercicio211.py - D:/Curso completo de Python/ejercicio211.py (3.11.1)

File Edit Format Run Options Window Help

```
1  class Rectangulo:
2
3      def __init__(self, ladomenor, ladomayor):
4          self.ladomenor=ladomenor
5          self.ladomayor=ladomayor
6
7      def superficie(self):
8          superficie=self.ladomenor*self.ladomayor
9          return superficie
10
11     def __eq__(self, objeto2):
12         return self.superficie()==objeto2.superficie()
13
14     def __ne__(self, objeto2):
15         return self.superficie()!=objeto2.superficie()
16
17     def __gt__(self, objeto2):
18         return self.superficie()>objeto2.superficie()
19
20     def __ge__(self, objeto2):
21         return self.superficie()>=objeto2.superficie()
22
23     def __lt__(self, objeto2):
24         return self.superficie()<objeto2.superficie()
25
26     def __le__(self, objeto2):
27         return self.superficie()<=objeto2.superficie()
28
29 # Bloque principal
30 rectangulo1=Rectangulo(10,5)
31 rectangulo2=Rectangulo(5,11)
32
33 if rectangulo1==rectangulo2:
34     print("El area del primer rectángulo y el sugundo son iguales")
35 else:
36     if rectangulo1!=rectangulo2:
37         print("El area del primer rectángulo y el segundo no son iguales")
38         if rectangulo1>rectangulo2:
39             print("El area del primer rectángulo es mayor al segundo rectángulo")
40         else:
41             print("El area del segundo rectángulo es mayor al primer rectángulo")
```

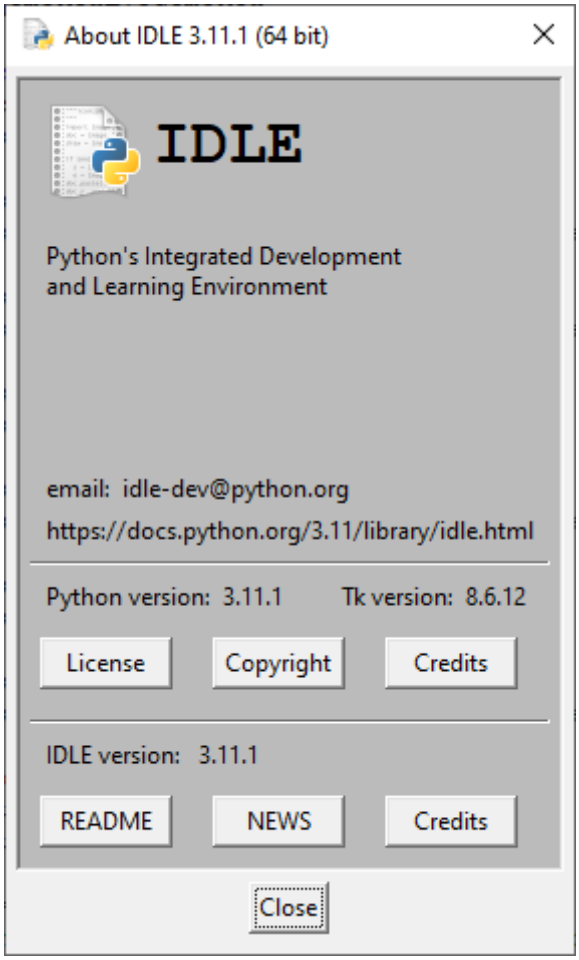
Ln: 46 Col: 4

Capítulo 212.- Instalación de editor Visual Studio Code (VSCode) y de la extensión para Python

Editores para programar Python

Hasta ahora hemos aprendido los conceptos fundamentales de Python y hemos utilizado el entorno de desarrollo IDLE que se instala por defecto con Python.

Cuando necesitamos una herramienta más completa para trabajar en forma más intensiva con Python el entorno IDLE nos puede quedar limitado.



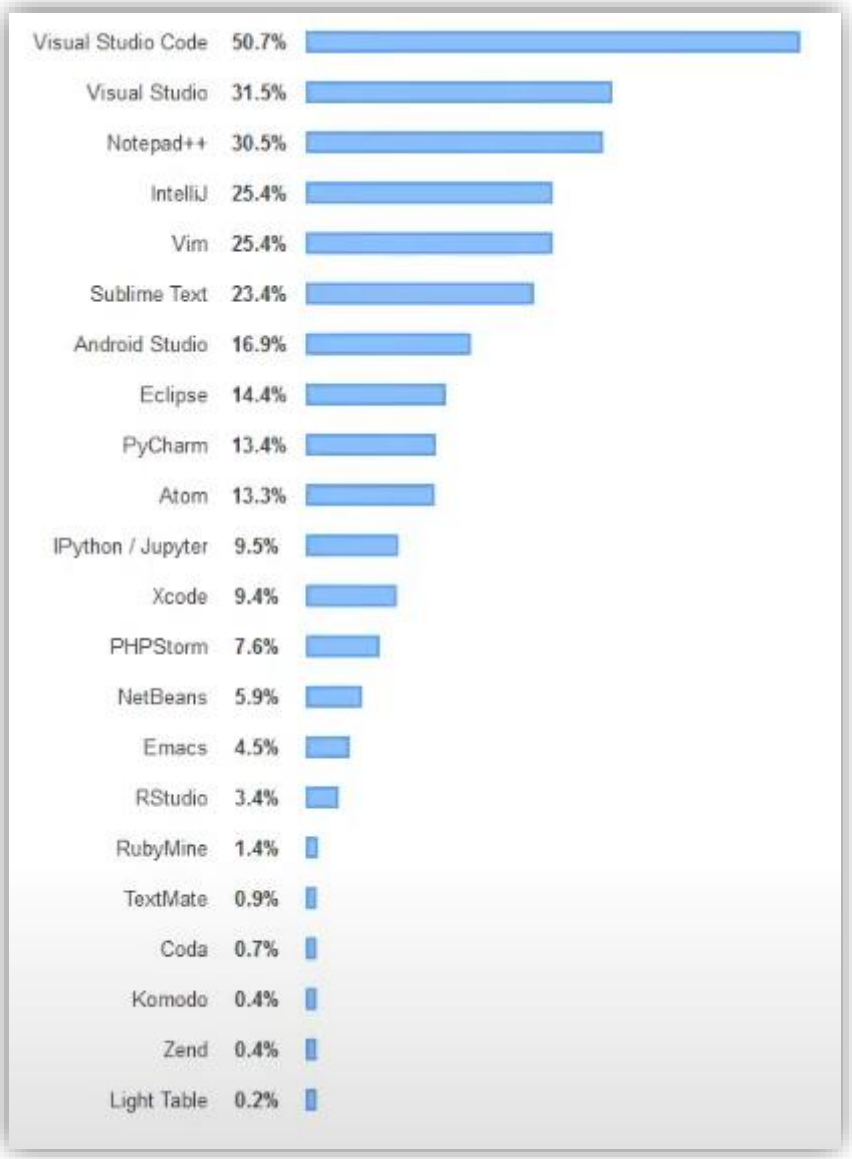
Veremos ahora otros editores que facilitan la programación en Python y elegiremos uno de ellos para continuar estudiando Python:

1. **PyCharm**: Es un entorno de desarrollo creado por la empresa Jet Brains. Viene en dos versiones, la básica que es open source y la podemos descargar del sitio JetBrains . En la misma página tenemos el enlace para registrarnos y comprar la versión profesional.
2. **PyDev IDE**: Este entorno de desarrollo es una adaptación del Editor Eclipse. Lo podemos descargar del sitio de: Pydev. Es open source al igual que el editor Eclipse.
3. **Wing**: Es un entorno de trabajo con Python con numerosas herramientas para programar y depurar programas. Hay una versión gratuita, pero la que tiene todas las herramientas es comercial. Podemos descargar este entorno del sitio: wingware.

4. **VS Code**: Es un editor de texto que agregando extensiones podemos trabajar con Python con herramientas para programar y depurar programas. Es creado y mantenido por la empresa Microsoft. es open source y multiplataforma (Windows, Linux y Mac).

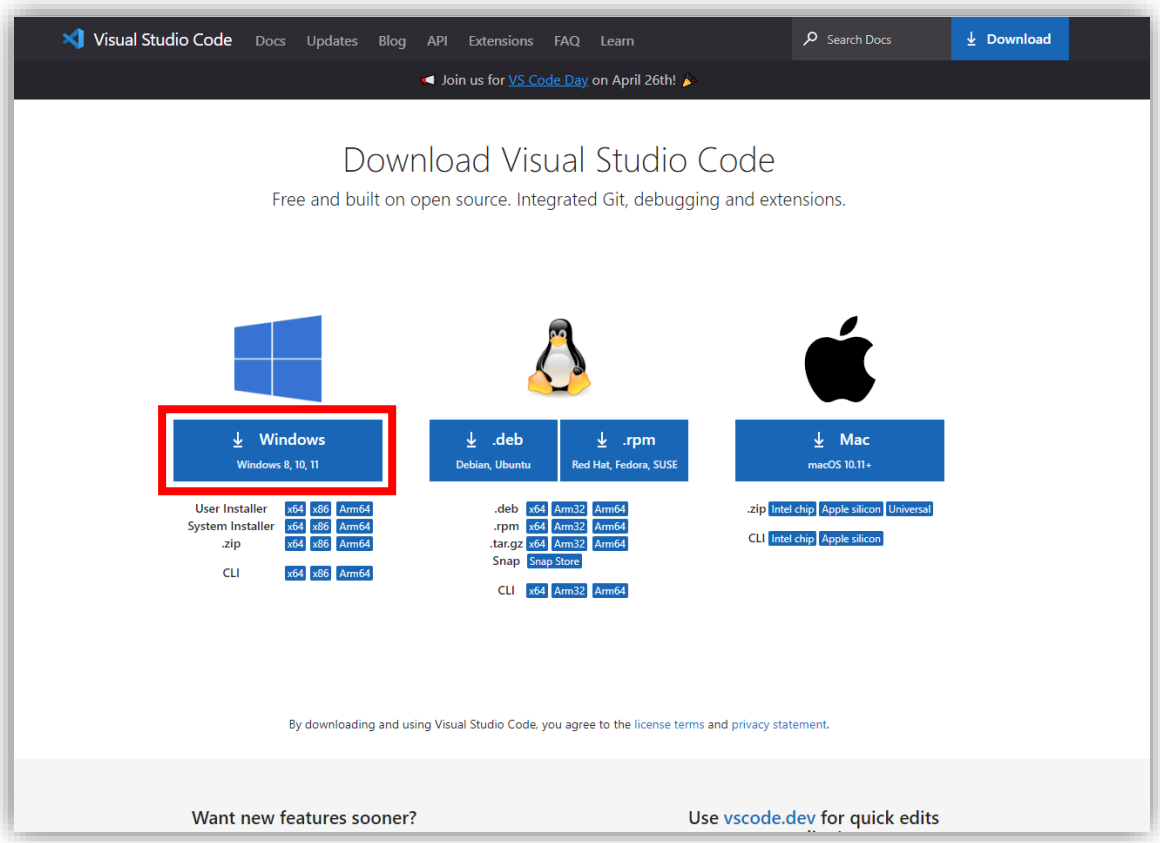
   En el último año se está convirtiendo en el editor de texto más utilizado por los programadores profesionales de todo el mundo.

   Podemos comprobar esta situación si visitamos la encuesta de desarrolladores 2019 del sitio stckovertflow.

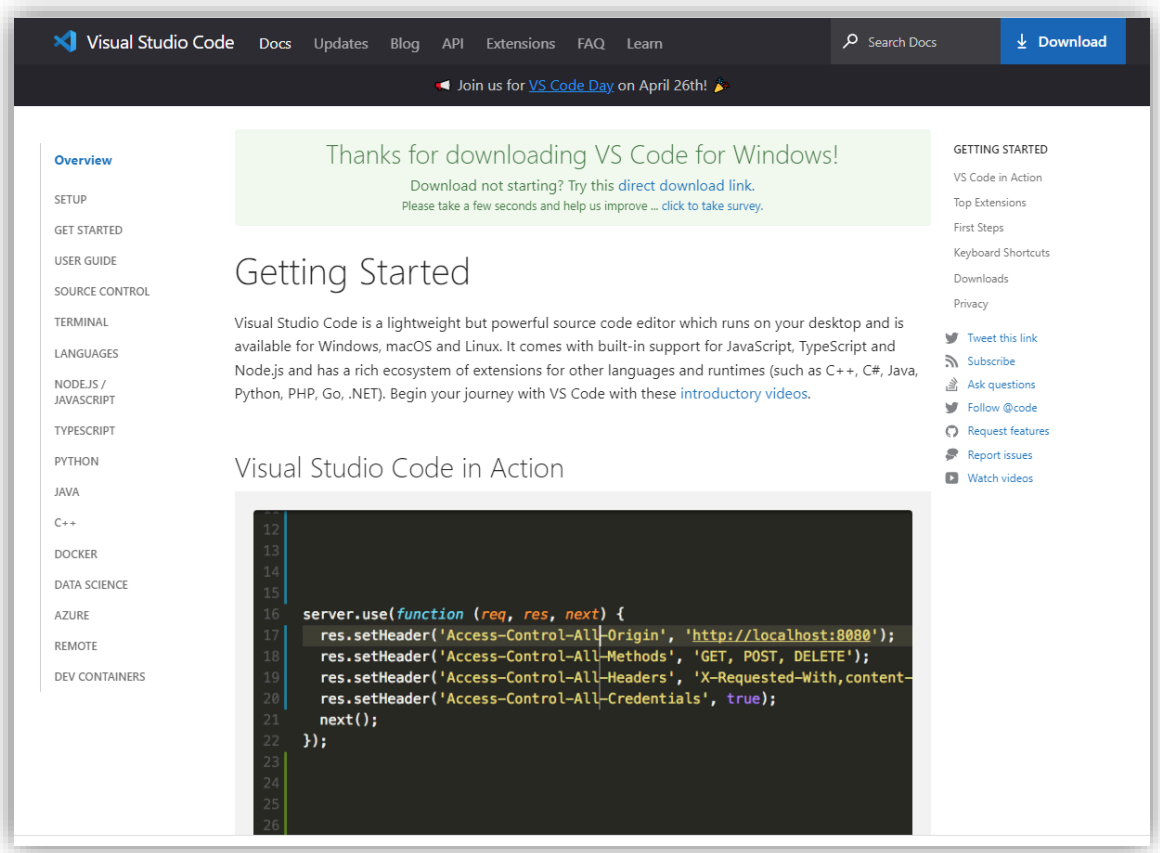


Para instalar dicho editor vamos a ir a la siguiente página web:

https://code.visualstudio.com/download

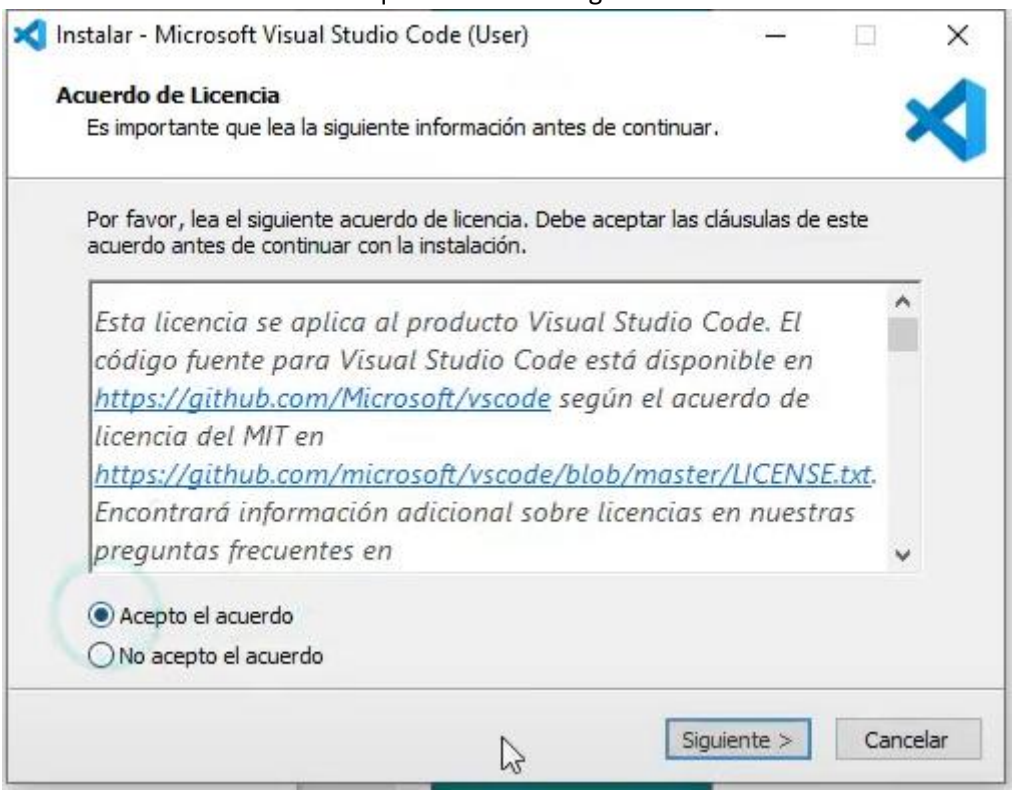


Procedemos a su descarga:

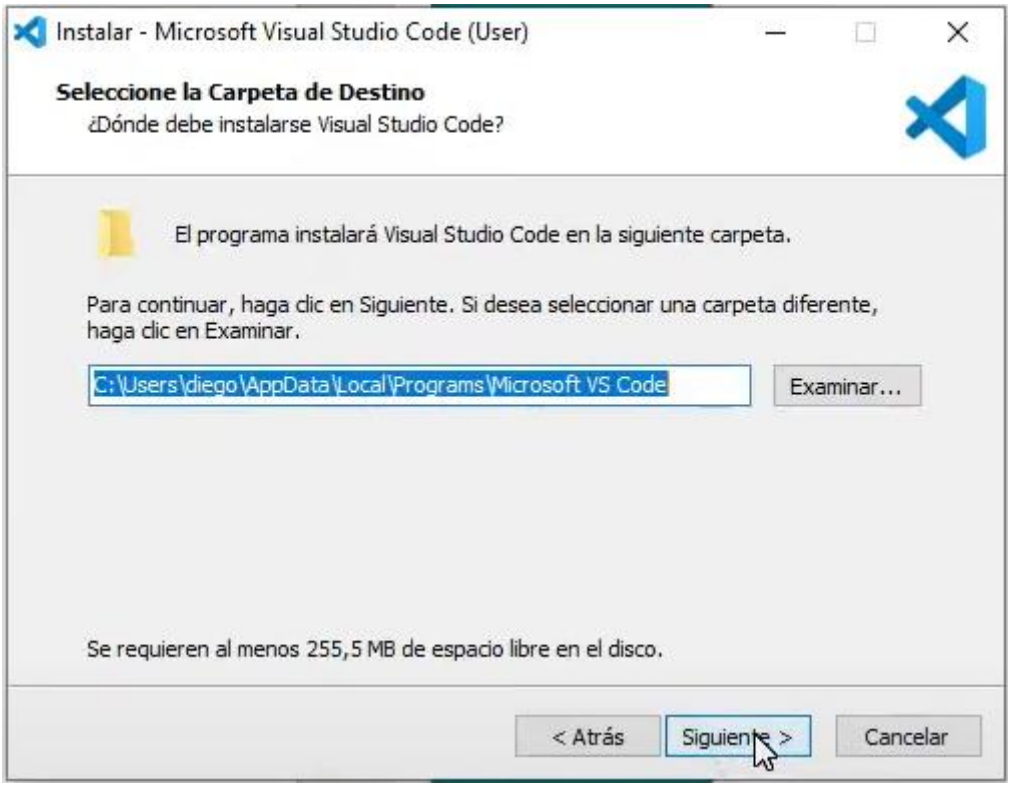


VSCodeUserSetup-x64-1.77.3.exe

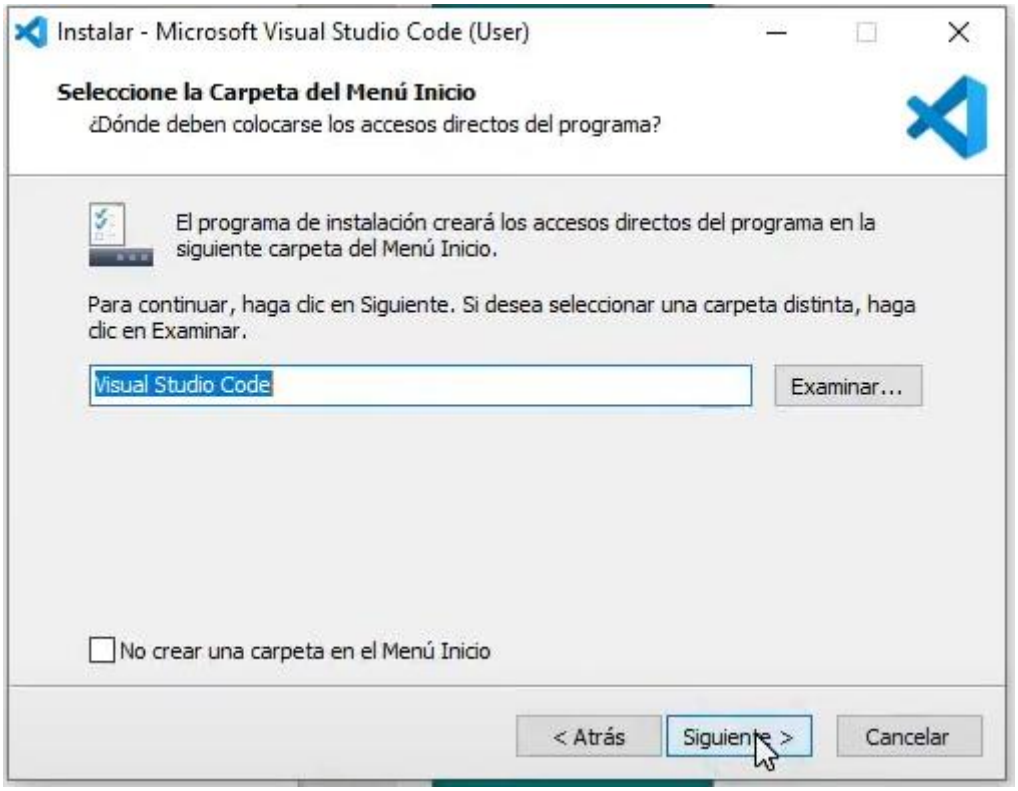
Hacemos doble clic sobre el archivo que hemos descargado.



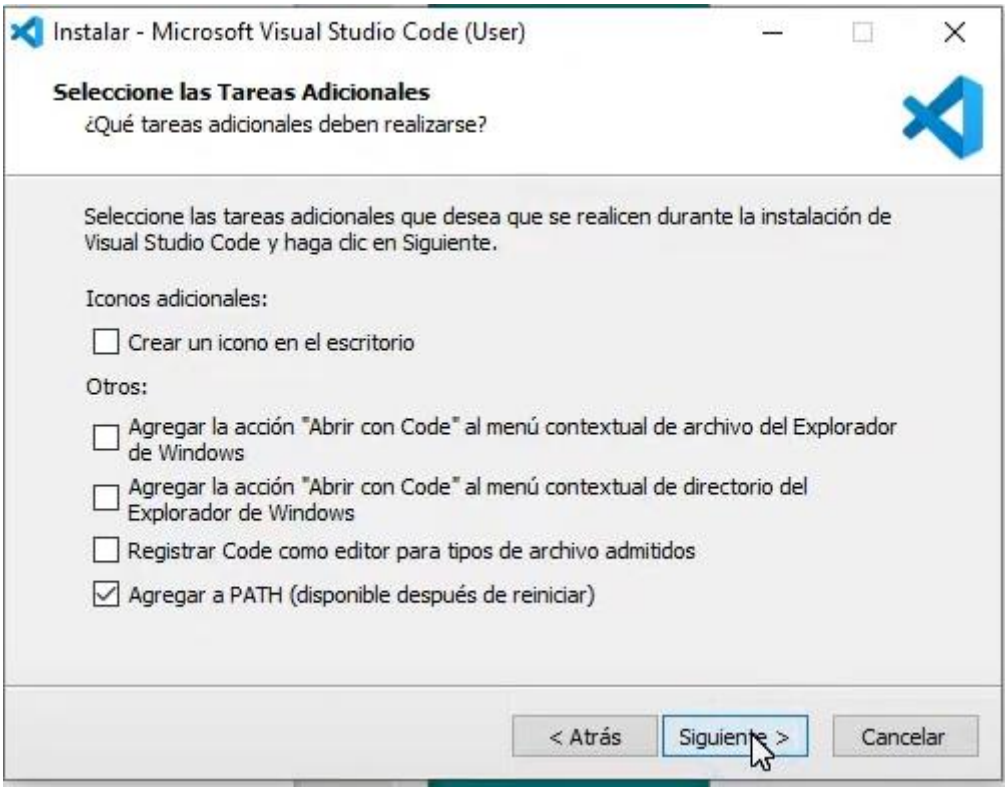
Aceptamos el acuerdo seguido del botón Siguiente.



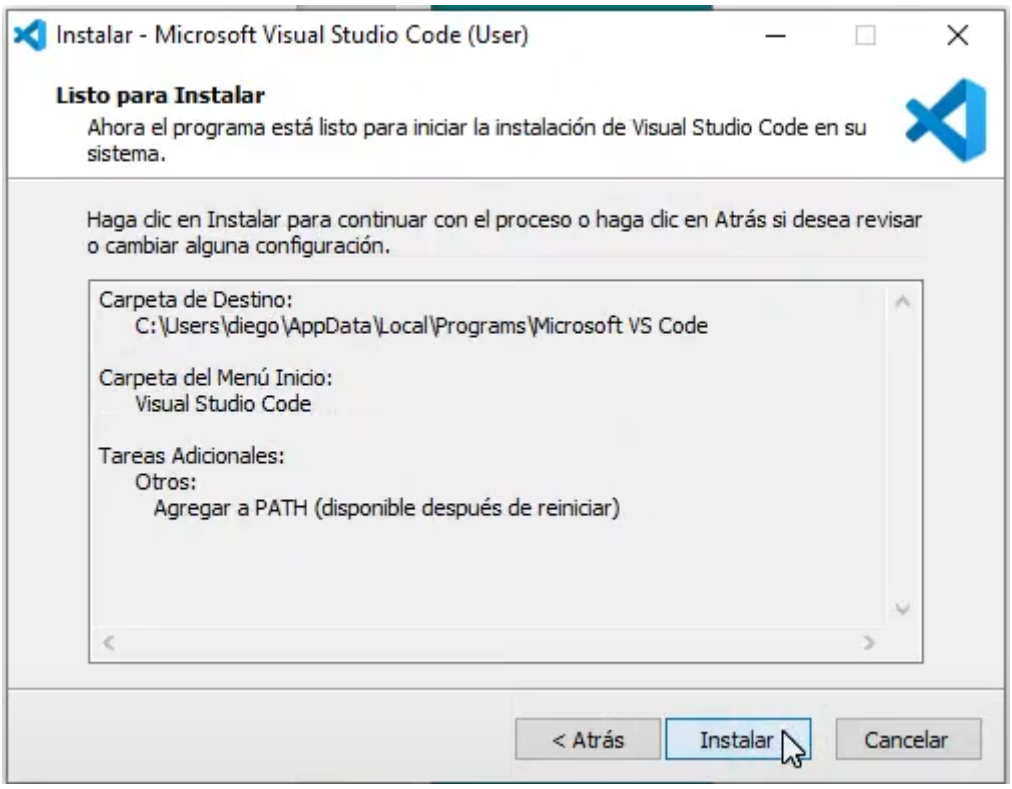
Dejamos que se instale en la ubicación que viene por defecto, seguido del botón Siguiente.



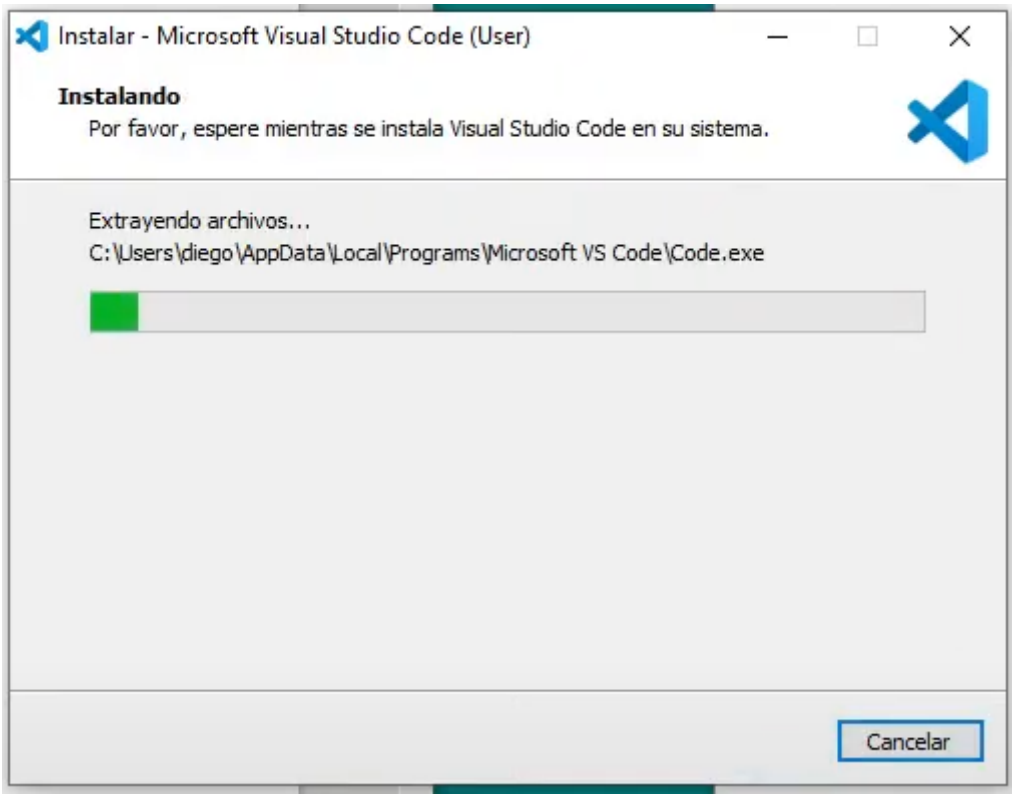
Le damos de nuevo a Siguiente.



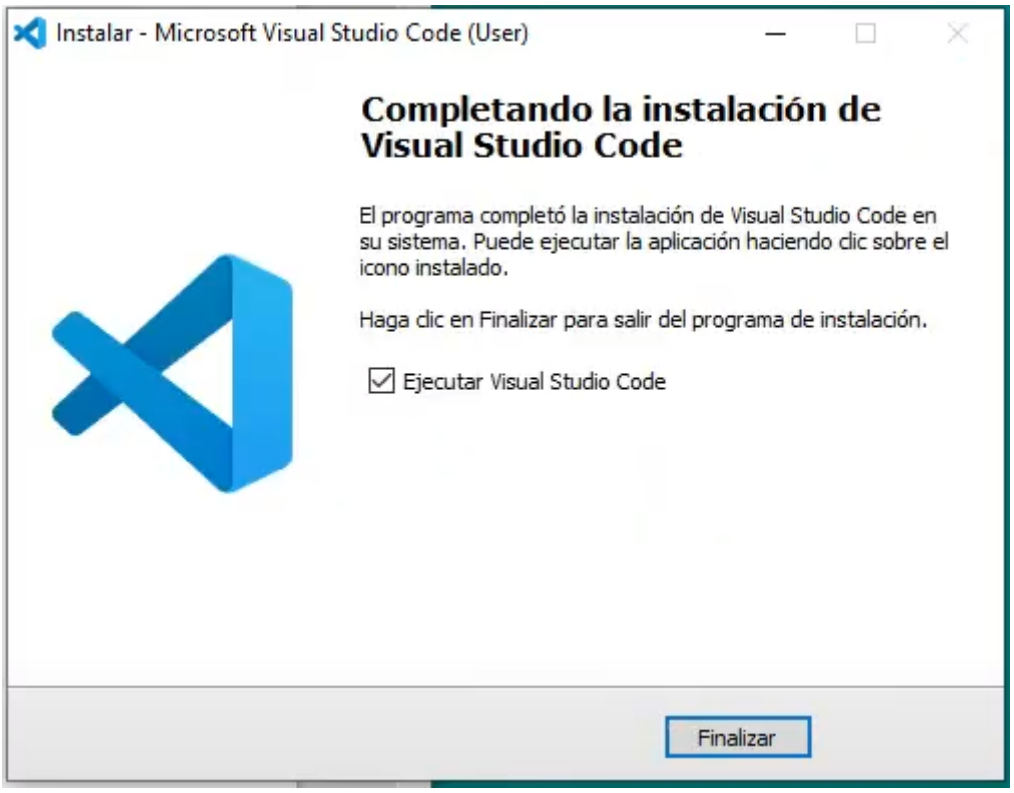
Le damos a Siguiente.



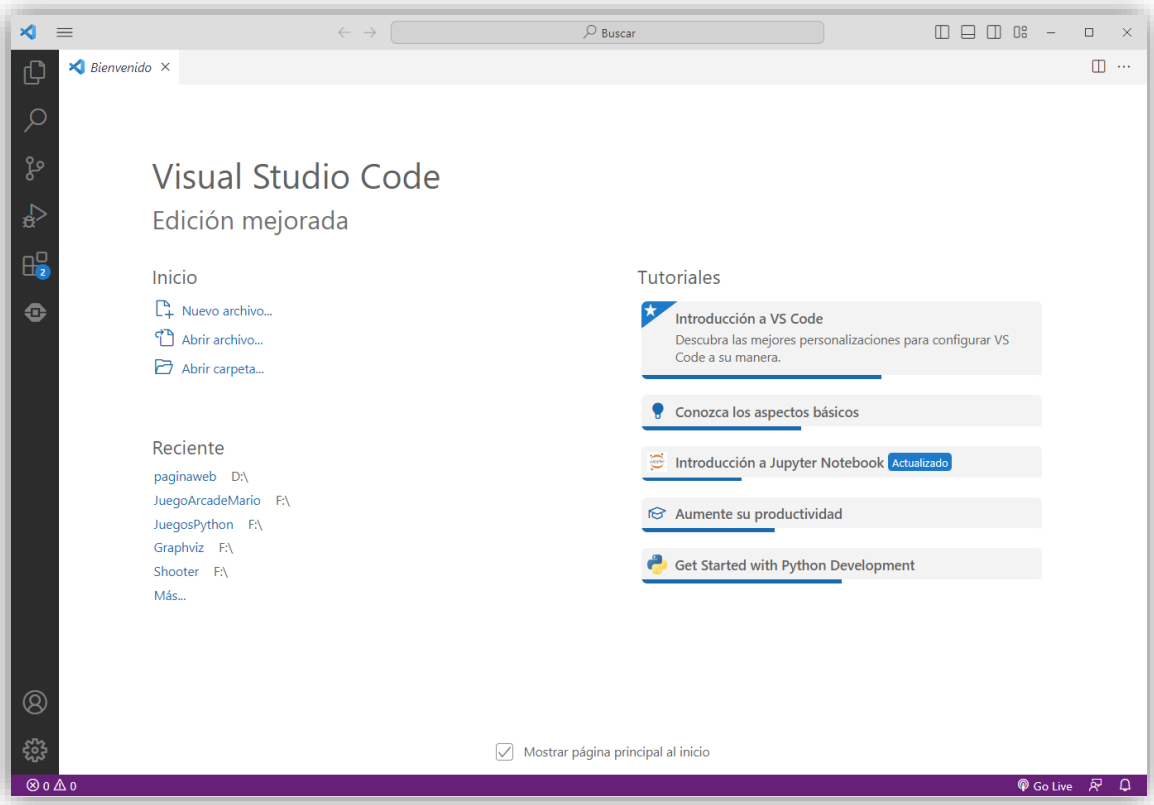
Le damos a Instalar.



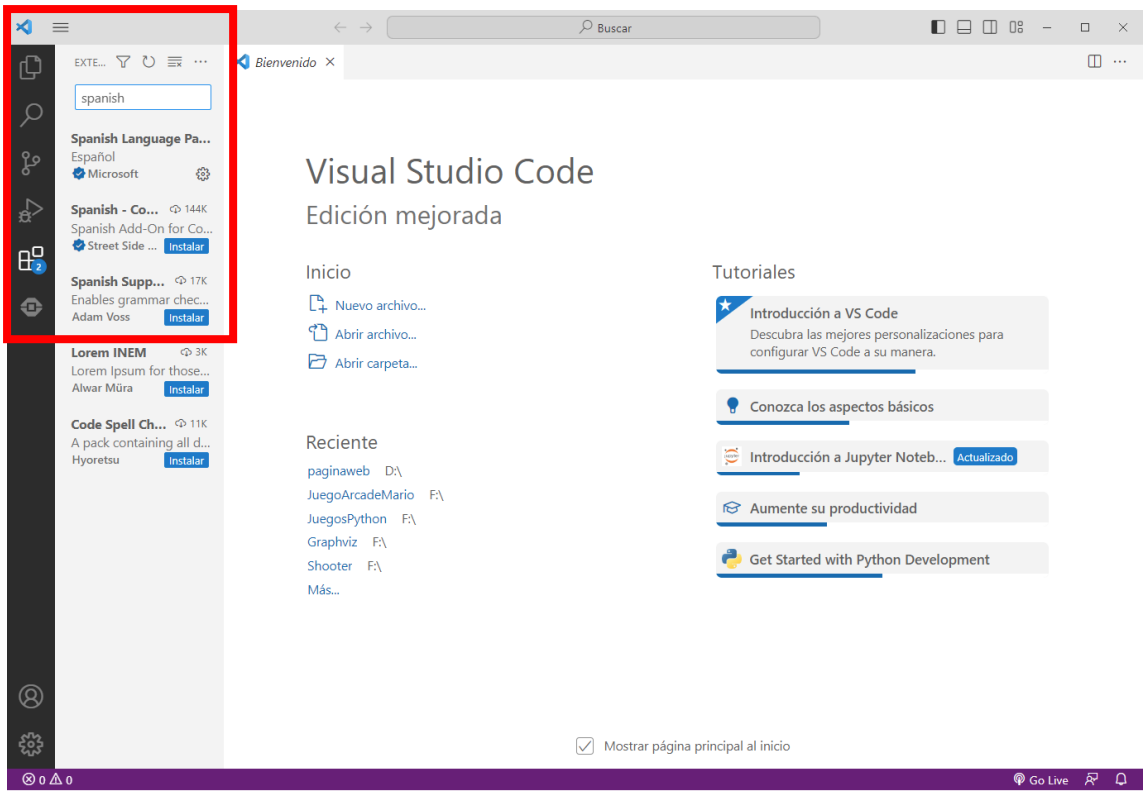
Empieza la instalación.

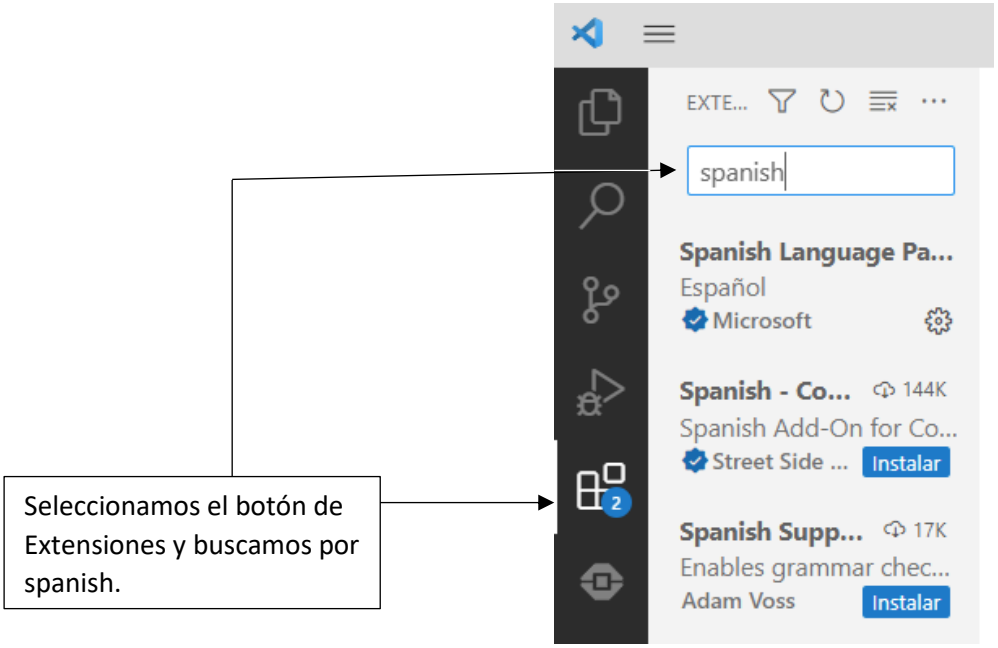


Le damos a Finalizar y el programa se ejecutará:



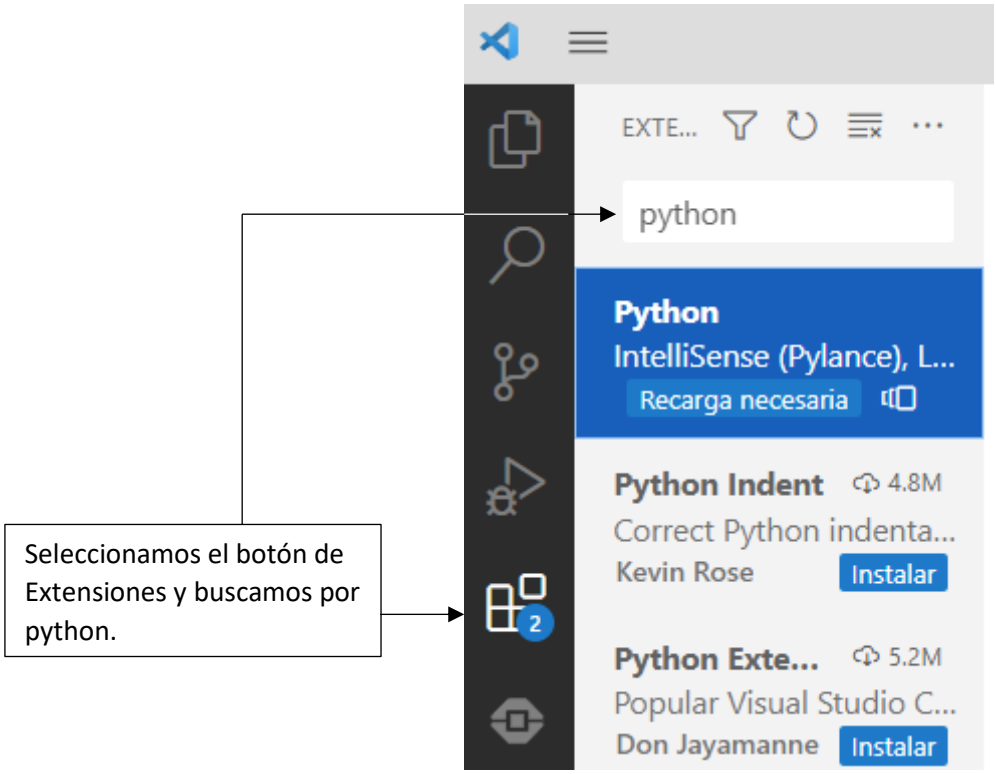
Por defecto sale en inglés vamos a instalar la extensión para que sea en castellano.

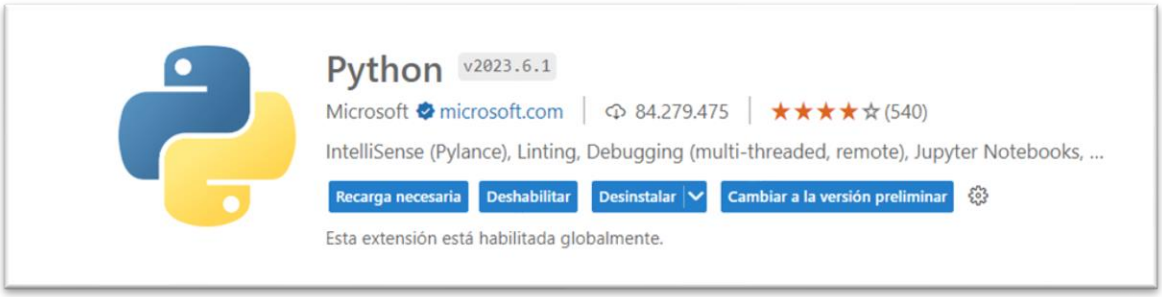

Buscar
EXTE...
spanish
Spanish Language Pa...
Español
Microsoft
Spanish - Co... 144K
Spanish Add-On for Co...
Street Side ... Instalar
Spanish Supp... 17K
Enables grammar chec...
Adam Voss Instalar
Lorem INEM 3K
Lorem Ipsum for those...
Alwar Müra Instalar
Code Spell Ch... 11K
A pack containing all d...
Hyoretsu Instalar
Bienvenido
Visual Studio Code
Edición mejorada
Inicio
Nuevo archivo...
Abrir archivo...
Abrir carpeta...
Reciente
paginaweb D:\
JuegoArcadeMario F:\
JuegosPython F:\
Graphviz F:\
Shooter F:\
Más...
Tutoriales
Introducción a VS Code
Descubra las mejores personalizaciones para configurar VS Code a su manera.
Conozca los aspectos básicos
Introducción a Jupyter Noteb... Actualizado
Aumente su productividad
Get Started with Python Development
Mostrar página principal al inicio
Go Live


EXTE...
spanish
Spanish Language Pa...
Español
Microsoft
Spanish - Co... 144K
Spanish Add-On for Co...
Street Side ... Instalar
Spanish Supp... 17K
Enables grammar chec...
Adam Voss Instalar
Seleccionamos el botón de Extensiones y buscamos por spanish.

Extensión: Spanish Language Pack for Visual Studio Code
Spanish Language Pack for Visual Studio Code
Microsoft microsoft.com 4.479.257 (14)
Language pack extension for Spanish
Desinstalar
Esta extensión está habilitada globalmente.

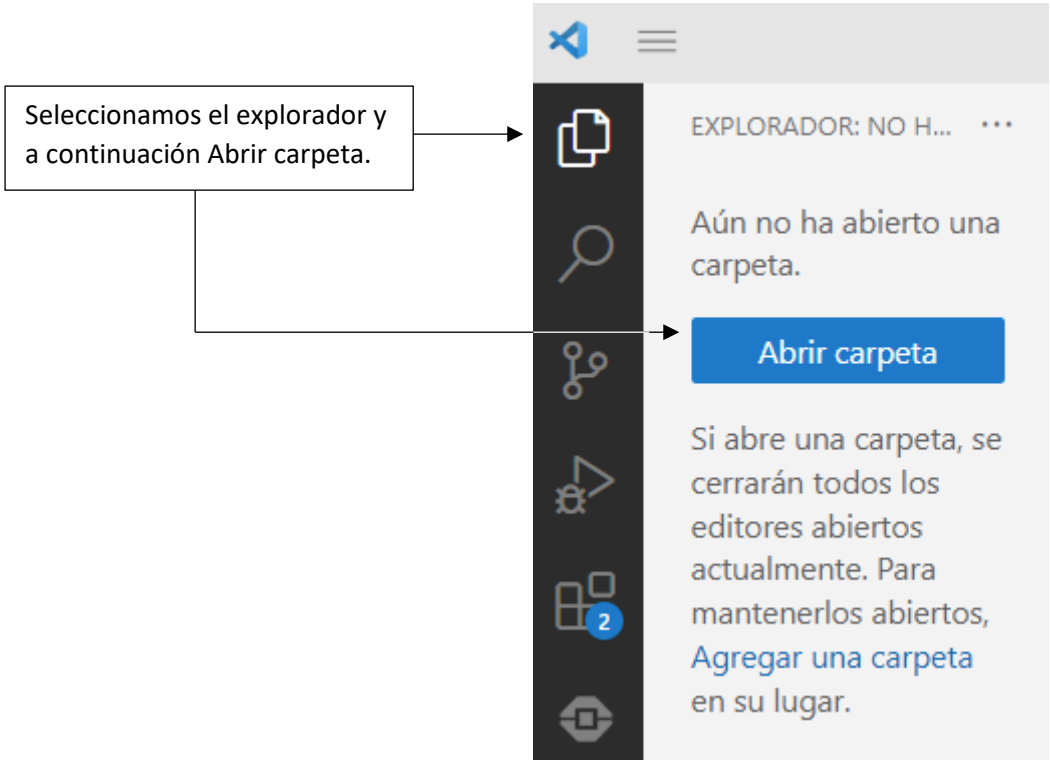
En mi caso ya lo tengo instalado, en tu caso tienen que seleccionar instalar.
Una vez terminada la instalación reiniciamos el programa.
El siguiente paso será descargar la extensión para trabajar con Python.



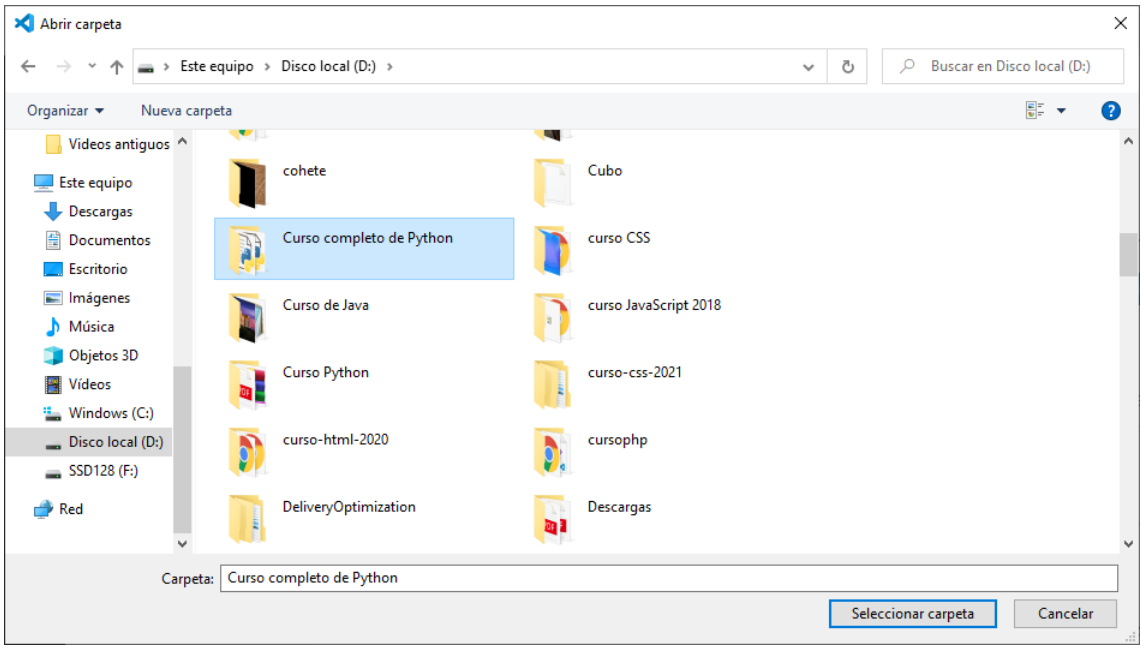


En mi caso ya lo tengo instalado tu lo tendrás que instalar.
Ahora que ya hemos instalado la extensión vamos

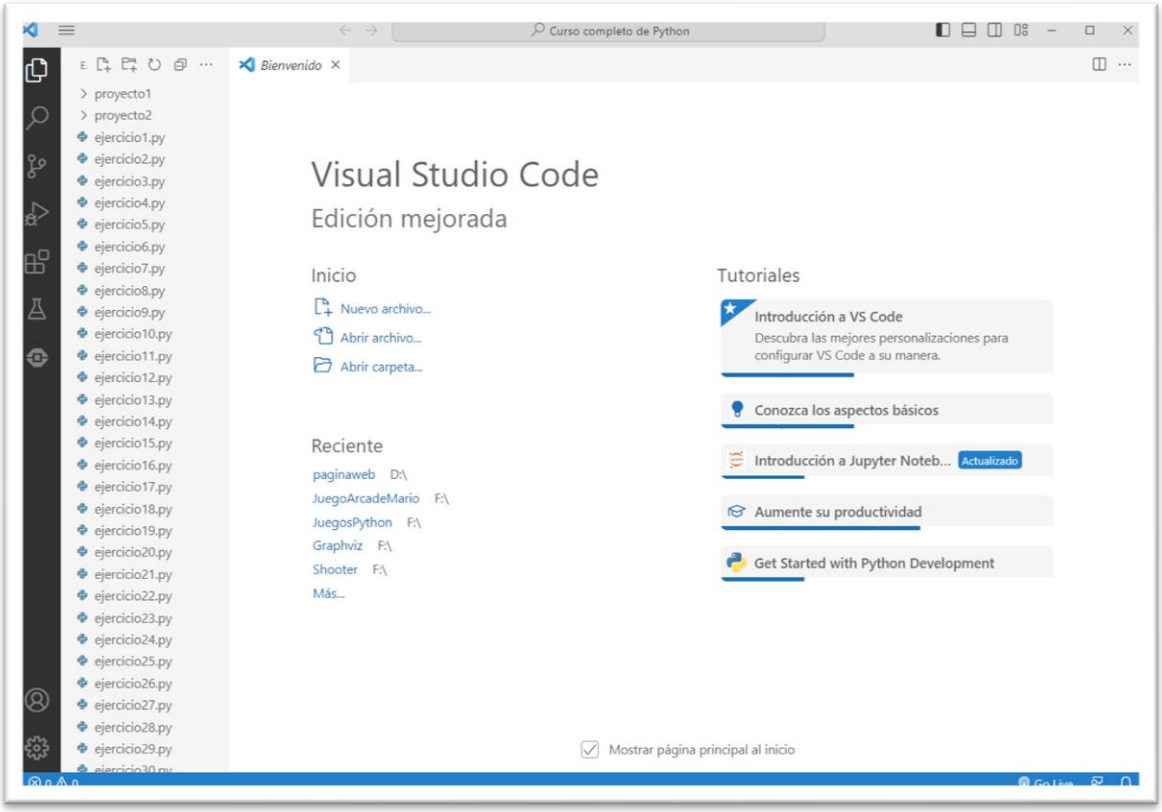
la apartado de explorador.



Seleccionamos la carpeta donde hasta ahora hemos guardado todos los proyectos.

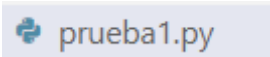


Seguido de seleccionar carpeta.



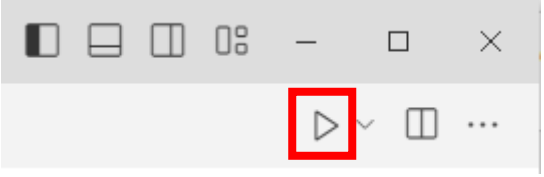
Desde el editor Visual Studio Code ya tenemos acceso a todos nuestros programas.
Vamos a realizar tres pequeños programas de ejemplo una de programación lineal, programación estructural y orientado a objetos.

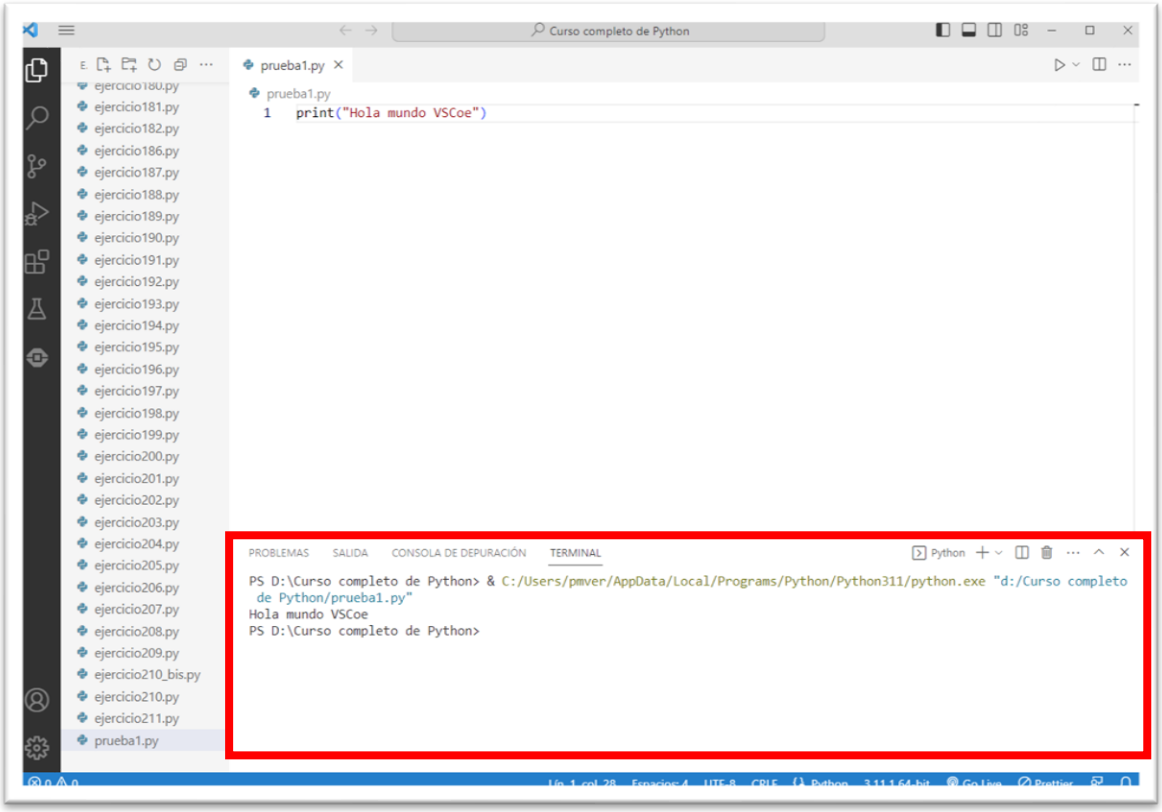
Seleccionamos el botón de nuevo proyecto.



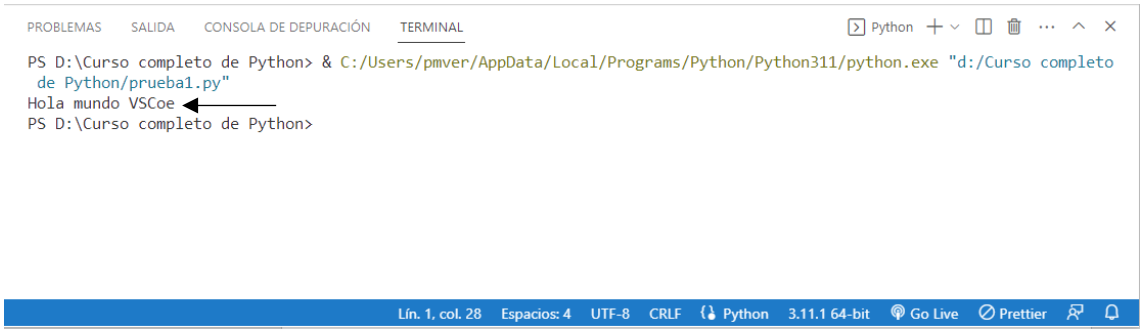
Lo vamos a llamar prueba1.py.

Ahora para ejecutarlo en la parte superior derecha encontraremos el botón para ejecutar:



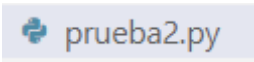


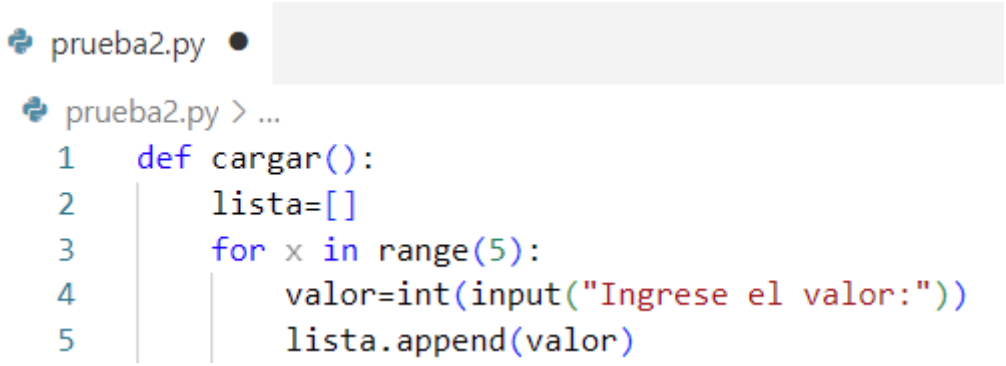
En la parte inferior aparece el terminal con el resultado del programa:



Desde la terminar obtendremos el resultado cuando ejecutemos el programa.

Vamos a crear nuestra segunda aplicación con el nombre de prueba2.py.





```
6        return lista
7
8    def imprimir(lista):
9        for elementos in lista:
10           print(elementos)
11
12
13   # Bloque principal
14   lista=cargar()
15   imprimir(lista)
```

Lo ejecutamos y este será el resultado:

```
PROBLEMAS    SALIDA    CONSOLA DE DEPURACIÓN    TERMINAL

 de Python/prueba2.py"
Ingrese el valor:1
Ingrese el valor:2
Ingrese el valor:3
Ingrese el valor:4
Ingrese el valor:5
1
2
3
4
5
PS D:\Curso completo de Python>
```

Vamos a realizar el tercer ejemplo con el nombre prueba3.py.

```
prueba3.py ×
prueba3.py > ...
1    class Persona:
2
3        def __init__(self,nombre,edad):
4            self.nombre=nombre
5            self.edad=edad
6
7        def imprimir(self):
8            print("Nombre:", self.nombre)
9            print("Edad:", self.edad)
10
```

```
11      def es_mayor(self):
12          if self.edad>=18:
13              print(self.nombre, "es mayor de edad")
14          else:
15              print(self.nombre,"No es mayor de edad")
16
17  # Bloque principal
18  persona1=Persona("Juan", 44)
19  persona1.imprimir()
20  persona1.es_mayor()
```

Si ejecutamos este será el resultado en la terminal:

```
PROBLEMAS    SALIDA    CONSOLA DE DEPURACIÓN    TERMINAL                    Python

PS D:\Curso completo de Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe
 "d:/Curso completo de Python/prueba3.py"
Nombre: Juan
Edad: 44
Juan es mayor de edad
PS D:\Curso completo de Python>
```

Podemos observar que tiene muchas más opciones para depurar los programas.

Podemos hacer que el programa se detenga en un determinado paso o hacer que se ejecute línea por línea.

Ya veremos en próximos capítulos las opciones que tiene.

Capítulo 213.- pip: instalador de paquetes

pip es un programa que viene con Python y nos permite instalar paquetes de terceros con el objetivo de valernos de librerías y código desarrollado por la comunidad de Python.

El sitio que podemos visitar para conocer que paquetes están disponibles se llama pipy.org. En este momento hay más de 232.000 paquetes disponibles sobre temas tan diversos como:

Problema 1:

Instalar el paquete colorise y probar alguna de sus funcionalidades.

Desde la línea de comando de la terminal, si esta no se encuentra visible desde el menú seleccionaremos Ver y de este Terminal.

```
[notice] A new release of pip is available: 23.0 -> 23.1.2
[notice] To update, run: python.exe -m pip install --upgrade pip
PS D:\Curso completo de Python>
```

Desde la terminal escribimos pip install colorise, cuando le demos a enter se realizará la instalación.

Vamos a realizar nuestro programa llamado Ejercicio213.py.

Ejercicio213.py

```
1   import colorise
2
3   colorise.cprint("Hola Mundo", fg="#ff0000", bg="#0000ff")
4   colorise.cprint("Línea 2", fg="#ff0000", bg="#ffffff")
```

Ahora lo vamos a ejecutar:

```
Hola Mundo
Línea 2
```

Recuerda que para utilizarlo tendremos que importar el paquete.

Capítulo 214.- Interfaz gráfica de usuario: tkinter

La interfaz gráfica de usuario, conocida también como GUI (graphical user interface) son fundamentales cuando queremos implementar programas para usuarios finales que faciliten la entrada y salida de datos.

Python por defecto instala el módulo 'tkinter' para la implementación de interfaces visuales. El editor IDLE que viene con Python no está desarrollado utilizando este módulo.

Veremos en una serie de conceptos como implementar interfaces visuales con ventanas, botones, checkbox, label, etc. haciendo uno del módulo 'tkinter'.

Lo primero que hay que tener en cuenta es que el módulo 'tkinter' (**tkinter**face) viene instalado por defecto con Python.

La librería tkinter está codificada con la metodología de programación orientada a objetos.

Problema:

Mostrar una ventana y que en su título aparezca el mensaje 'Hola Mundo'.

Creamos un nuevo proyecto con el nombre de Ejercicio214.py.

Ejercicio215.py > ...

```
1   import tkinter as tk
2
3   ventana1=tk.Tk()
4   ventana1.title("Hola Mundo")
5   ventana1.mainloop()
```

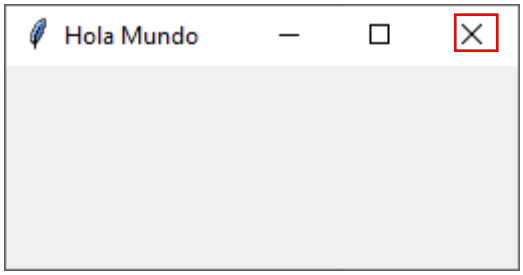
En la línea 1 importamos la librería tkinter y le agregamos el alias as para cuando la tengamos que llamar el nombre sea más corto.

En la línea 3 creamos un objeto llamado ventana1 que le instanciamos la librería tkinter con el método Tk() para que este objeto sea la ventana.

En la línea 4 a la ventana le agregamos un título con el método title.

En la línea 5 con el método mainloop() se va generando un bucle para que vaya recibiendo los eventos de dicha pantalla como moverla, cambiarla de tamaño, etc.

Lo vamos a ejecutar:



Si le damos a la x cerramos la aplicación.

Aplicación orientada a objetos

Todos los ejemplos que haremos de ahora en más independientemente de su complejidad utilizaremos la metodología de POO (Programación Orientada a Objetos).

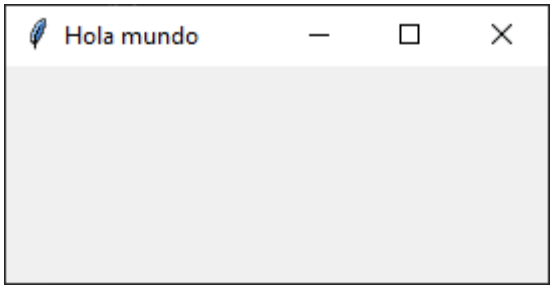
Rehacer el programa a que muestra una ventana y que en su título aparece el mensaje 'Hola Mundo'.

Vamos a ver el mismo ejemplo pero con POO.

Ejercicio215.py > ...

```
1   import tkinter as tk
2
3   class Aplicacion:
4       def __init__(self):
5           self.ventana1=tk.Tk()
6           self.ventana1.title("Hola mundo")
7           self.ventana1.mainloop()
8
9   # Bloque principal
10
11  aplicacion1=Aplicacion()
```

Si ejecutamos este será el resultado:



Capítulo 215.- tkinter: Controles Button y Label – 1

En las interfaces visuales las ventanas son controles visuales contenedores de otros controles como los botones y las etiquetas de texto.

Veremos los pasos que debemos dar si queremos que se muestren objetos de la clase Button y Label en nuestra ventana.

Problema:

Mostrar una ventana y en su interior dos botones y un label. El label muestra inicialmente el valor 1.

Cada uno de los botones permiten incrementar o decrementar en uno el contenido del label.

Vamos a programar:

ejercicio215.py > ...

```
1   import tkinter as tk
2
3   class Aplicacion:
4
5       def __init__(self):
6           self.valor=1
7           self.ventana1=tk.Tk()
8           self.ventana1.title("Controles Button y Label")
9
10          self.label1=tk.Label(self.ventana1, text=self.valor)
11          self.label1.grid(column=0,row=0)
12          self.label1.configure(background="red")
13
14          self.boton1=tk.Button(self.ventana1, text="Incrementar", command=self.incrementar)
15          self.boton1.grid(column=0, row=1)
16
17          self.boton2=tk.Button(self.ventana1, text="Decrementar", command=self.decrementar)
18          self.boton2.grid(column=0, row=2)
19
20          self.ventana1.mainloop()
21
22      def incrementar(self):
23          self.valor=self.valor+1
24          self.label1.config(text=self.valor)
25
26      def decrementar(self):
27          self.valor=self.valor-1
28          self.label1.config(text=self.valor)
29
30  # Bloque principal
31  aplicacion1=Aplicacion()
```

Vamos a ejecutar:

Ahora podemos aumentar el valor o disminuirlo de uno en uno.

Capítulo 216.- tkinter: Controles Button y Label – 2

Problema:

Mostrar dos label, en una se muestra el nombre del programa y en la segunda el año de creación.
Disponer de un botón para finalizar el programa.
No permitir al usuario redimensionar la ventana.

Vamos a programar:

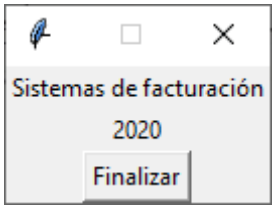
ejercicio216.py > ...

```python
import tkinter as tk
import sys

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Ejercicio 216")
        self.label1=tk.Label(self.ventana1, text="Sistemas de facturación")
        self.label1.grid(column=0, row=0)
        self.label2=tk.Label(self.ventana1, text="2020")
        self.label2.grid(column=0, row=1)
        self.boton1=tk.Button(self.ventana1, text="Finalizar", command=self.finalizar)
        self.boton1.grid(column=0, row=2)
        self.ventana1.resizable(False,False)
        self.ventana1.mainloop()

    def finalizar(self):
        sys.exit(0)

# Bloque principal
aplicacion1=Aplicacion()
```

No podemos modificar la ventana ni a lo ancho ni a lo alto.

Para utilizar el sys.exit(0) necesitamos importar la librería sys.

Vamos a ejecutar:



Capítulo 217.- tkinter: Controles Button y Label – 3

Problema propuesto:

Disponer dos objetos de la clase Button con las etiquetas: "varón" y "mujer", al presionarse mostrar en la barra de títulos de la ventana la etiqueta del botón presionado.

Vamos a programar:

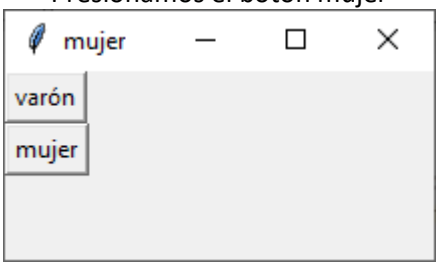
ejercicio217.py > ...

```python
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()

        self.boton1=tk.Button(self.ventana1, text="varón", command=self.mensaje_varon)
        self.boton1.grid(column=0, row=0)

        self.boton2=tk.Button(self.ventana1, text="mujer", command=self.mensaje_mujer)
        self.boton2.grid(column=0, row=1)

        self.ventana1.mainloop()

    def mensaje_varon(self):
        self.ventana1.title("varón")

    def mensaje_mujer(self):
        self.ventana1.title("mujer")

# Bloque principal

aplicacion1=Aplicacion()
```

Vamos a ejecutar:

Presionamos el botón varón

Presionamos el botón mujer



Capítulo 218.- tkinter: Controles Button y Label – 4

Problema propuesto:

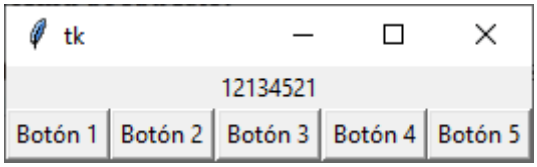
Mostrar los botones del 1 al 5. Cuando se presiona mostrar en un Label todos los botones presionados hasta el momento.

Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):

        self.valor=""

        self.ventana1=tk.Tk()
        self.label1=tk.Label(self.ventana1, text=self.valor)
        self.label1.grid(column=2, row=0)

        self.boton1=tk.Button(self.ventana1, text="Botón 1",
command=self.boton1)
        self.boton1.grid(column=0, row=1)

        self.boton2=tk.Button(self.ventana1, text="Botón 2",
command=self.boton2)
        self.boton2.grid(column=1, row=1)

        self.boton3=tk.Button(self.ventana1, text="Botón 3",
command=self.boton3)
        self.boton3.grid(column=2, row=1)

        self.boton4=tk.Button(self.ventana1, text="Botón 4",
command=self.boton4)
        self.boton4.grid(column=3, row=1)

        self.boton5=tk.Button(self.ventana1, text="Botón 5",
command=self.boton5)
        self.boton5.grid(column=4, row=1)

        self.ventana1.mainloop()

    def boton1(self):
        self.valor=self.valor+"1"
        self.label1.config(text=self.valor)

    def boton2(self):
        self.valor=self.valor+"2"
        self.label1.config(text=self.valor)
```

```
    def boton3(self):
        self.valor=self.valor+"3"
        self.label1.config(text=self.valor)

    def boton4(self):
        self.valor=self.valor+"4"
        self.label1.config(text=self.valor)

    def boton5(self):
        self.valor=self.valor+"5"
        self.label1.config(text=self.valor)

# Bloque principal
aplicacion1=Aplicacion()
```

Lo guardamos con el nombre de Ejercicio218.py y a continuación lo ejecutamos:



Capítulo 219.- tkinter: Control Entry – 1

Hemos visto hasta ahora los controles Button y Label, ahora veremos otro control visual indispensable para hacer la entrada de datos por teclado.

En tkinter el control de entrada de datos por teclado se llama Entry. Con este control aparece el típico recuadro que cuando se le da el foco aparece el cursor en forma intermitente esperando que el operador escriba algo por teclado.

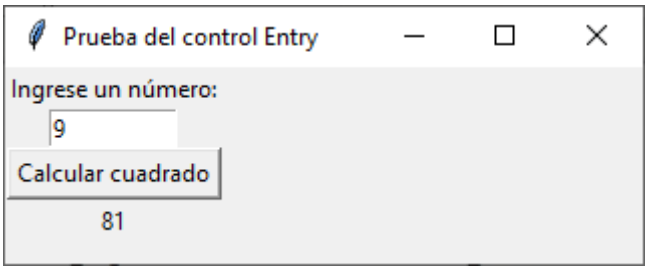
Problema:

Confeccionar una aplicación que permita ingresar un entero por teclado y al presionar un botón muestre dicho valor elevado al cuadrado en un Label.

Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Prueba del control Entry")
        self.label1=tk.Label(self.ventana1, text="Ingrese un número:")
        self.label1.grid(column=0, row=0)
        self.dato=tk.StringVar()
        self.entry1=tk.Entry(self.ventana1, width=10,
textvariable=self.dato)
        self.entry1.grid(column=0, row=1)
        self.boton1=tk.Button(self.ventana1, text="Calcular cuadrado",
command=self.calcularcuadrado)
        self.boton1.grid(column=0, row=2)
        self.label2=tk.Label(self.ventana1, text="resultado")
        self.label2.grid(column=0, row=3)
        self.ventana1.mainloop()

    def calcularcuadrado(self):
        valor=int(self.dato.get())
        cuadrado=valor*valor
        self.label2.configure(text=cuadrado)

# Bloque principal
aplicacion1=Aplicacion()
```

Lo vamos a ejecutar:



Capítulo 220.- tkinter: Control Entry – 2

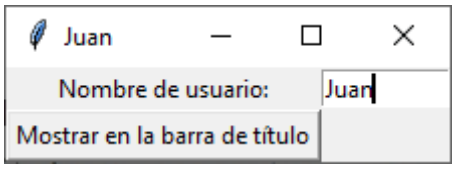
Problema:

Confeccionar un programa que permita ingresar el nombre de usuario en un control Entry y cuando presione un botón mostrar el valor ingresado en la barra de títulos de la ventana.

Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Prueba del control Entry")
        self.label1=tk.Label(self.ventana1, text="Nombre de usuario:")
        self.label1.grid(column=0, row=0)
        self.dato=tk.StringVar()
        self.entry1=tk.Entry(self.ventana1, width=10,
textvariable=self.dato)
        self.entry1.grid(column=1, row=0)
        self.boton1=tk.Button(self.ventana1, text="Mostrar en la barra de
título", command=self.mostrartitulo)
        self.boton1.grid(column=0, row=1)
        self.ventana1.mainloop()

    def mostrartitulo(self):
        self.ventana1.title(self.dato.get())

# Bloque principal
aplicacion1=Aplicacion()
```

Vamos a ejecutar:



Capítulo 221.- tkinter: Control Entry – 3

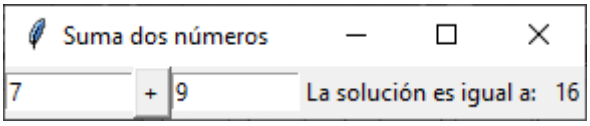
Problema propuesto:

Confeccionar un programa que permita ingresar dos números en controles de tipo Entry, luego sumar los dos valores ingresados y mostrar la suma en un Label al presionar un botón.

Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Suma dos números")
        self.dato1=tk.StringVar()
        self.entry1=tk.Entry(self.ventana1, width=10,
textvariable=self.dato1)
        self.entry1.grid(column=0, row=0)
        self.boton1=tk.Button(self.ventana1, text="+",
command=self.sumar)
        self.boton1.grid(column=1, row=0)
        self.dato2=tk.StringVar()
        self.entry2=tk.Entry(self.ventana1, width=10,
textvariable=self.dato2)
        self.entry2.grid(column=2, row=0)
        self.label1=tk.Label(self.ventana1, text="La solución es igual a:
")
        self.label1.grid(column=3, row=0)
        self.label2=tk.Label(self.ventana1, text="0")
        self.label2.grid(column=4, row=0)
        self.ventana1.mainloop()

    def sumar(self):
        num1=int(self.dato1.get())
        num2=int(self.dato2.get())
        suma=num1+num2
        self.label2.configure(text=suma)

# Bloque principal
aplicacion1=Aplicacion()
```

Si ejecutamos este será el resultado:



Capítulo 222.- tkinter: Control Entry – 4

Problema propuesto:

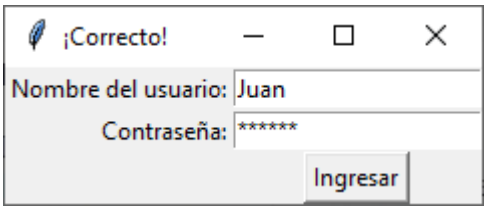
Ingresar el nombre de usuario y clave en controles de tipo Entry. Si se ingresa la cadenas (usuario: Juan, clave="abc123") luego mostrar en el título de la ventana el mensaje "Correcto" en caso contrario mostrar el mensaje "Incorrecto".

Para mostrar '*' cuando se ingresa la clave debemos pasar en el parámetro 'show' el carácter a mostrar:

self.entry=tk.Entry(self.ventana1, whidth=30, textvariable=self.datos2, show="*")

Vamos a programar:

```
mport tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Usuario y contraseña")
        self.label1=tk.Label(self.ventana1, text="Nombre del usuario:")
        self.label1.grid(column=0, row=0)
        self.usuario=tk.StringVar()
        self.entry1=tk.Entry(self.ventana1, width=20,
textvariable=self.usuario)
        self.entry1.grid(column=1, row=0)
        self.label2=tk.Label(self.ventana1,
text="                Contraseña:")
        self.label2.grid(column=0, row=1)
        self.contrasena=tk.StringVar()
        self.entry2=tk.Entry(self.ventana1, width=20,
textvariable=self.contrasena, show="*")
        self.entry2.grid(column=1, row=1)
        self.boton1=tk.Button(self.ventana1, text="Ingresar",
command=self.ingresar)
        self.boton1.grid(column=1, row=2)
        self.ventana1.mainloop()

    def ingresar(self):
        if self.usuario.get()=="Juan" and
self.contrasena.get()=="abc123":
            self.ventana1.title("¡Correcto!")
        else:
            self.ventana1.title("¡Incorrecto!")

# Bloque principal
aplicacion1=Aplicacion()
```



Vamos a ejecutar:

Capítulo 223.- tkinter: Control Radiobutton – 1

Otro control visual muy común es el Radiobutton que normalmente se muestran un conjunto de Radiobutton y permite la selección de solo uno de ellos.

Se los debre agrupar para que actúe en conjunto, es decir cuando se selecciona uno automáticamente se deben deseleccionar los otros.

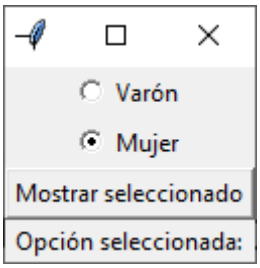
Problema:

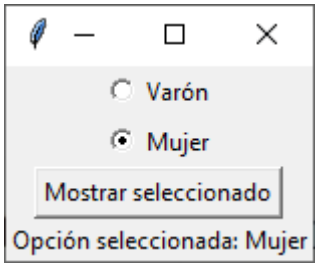
Mostrar dos controles de tipo Radiubutton con las etiquetas "Varón" y "Mujer", cuando se presione un botón actuaiza un Lebel con el Radiobutton seleccionado.

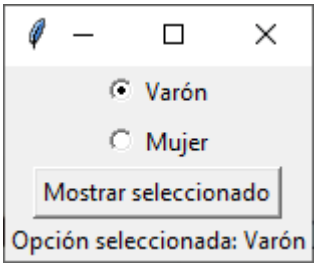
Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.seleccion=tk.IntVar()
        self.seleccion.set(2)
        self.radio1=tk.Radiobutton(self.ventana1, text="Varón",
variable=self.seleccion, value=1)
        self.radio1.grid(column=0, row=0)
        self.radio2=tk.Radiobutton(self.ventana1, text="Mujer",
variable=self.seleccion, value=2)
        self.radio2.grid(column=0, row=1)
        self.boton1=tk.Button(self.ventana1, text="Mostrar seleccionado",
command=self.mostrarseleccionado)
        self.boton1.grid(column=0, row=2)
        self.label1=tk.Label(self.ventana1, text="Opción seleccionada:")
        self.label1.grid(column=0, row=3)
        self.ventana1.mainloop()

    def mostrarseleccionado(self):
        if self.seleccion.get()==1:
            self.label1.configure(text="Opción seleccionada: Varón")
        if self.seleccion.get()==2:
            self.label1.configure(text="Opción seleccionada: Mujer")

# Bloque principal
aplicacion1=Aplicacion()
```

Vamos a ejecutar:


Varón
Mujer
Mostrar seleccionado
Opción seleccionada:


Varón
Mujer
Mostrar seleccionado
Opción seleccionada: Mujer


Varón
Mujer
Mostrar seleccionado
Opción seleccionada: Varón

Capítulo 224.- tkinter: Control Radiobutton – 2

Problema:

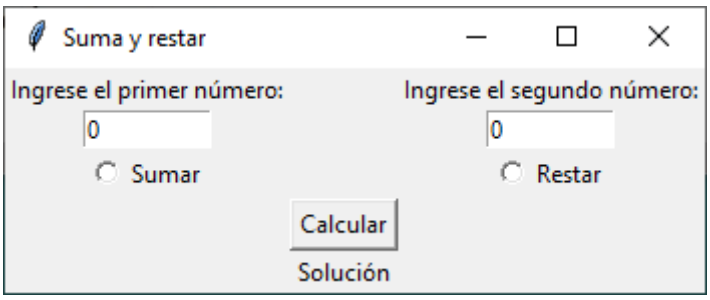
Disponer dos controles de tipo Entry para el ingreso de enteros. Mediante dos controles Radiobutton permitir seleccionar si queremos sumarlos o restarlos. Al presionar un botón mostrar el resultado de la operación seleccionada.

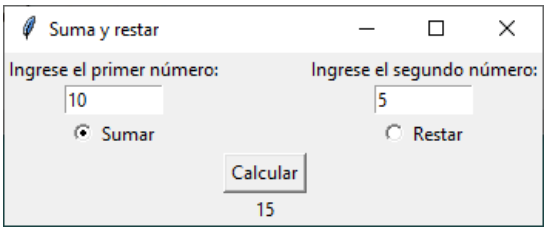
Vamos a programar:

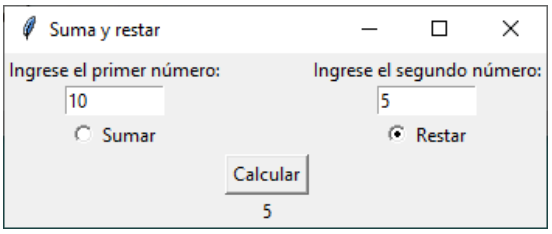
```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Suma y restar")
        label1=tk.Label(self.ventana1, text="Ingrese el primer número:")
        label1.grid(column=0, row=0)
        label2=tk.Label(self.ventana1, text="Ingrese el segundo número:")
        label2.grid(column=2, row=0)
        self.num1=tk.IntVar()
        self.entry1=tk.Entry(self.ventana1, width=10,
textvariable=self.num1)
        self.entry1.grid(column=0, row=1)
        self.num2=tk.IntVar()
        self.entry2=tk.Entry(self.ventana1, width=10,
textvariable=self.num2)
        self.entry2.grid(column=2, row=1)
        self.opcion=tk.IntVar()
        self.radio1=tk.Radiobutton(self.ventana1, text="Sumar",
variable=self.opcion, value=1)
        self.radio1.grid(column=0, row=2)
        self.radio2=tk.Radiobutton(self.ventana1, text="Restar",
variable=self.opcion, value=2)
        self.radio2.grid(column=2, row=2)
        self.boton1=tk.Button(self.ventana1, text="Calcular",
command=self.calcular)
        self.boton1.grid(column=1, row=3)
        self.label3=tk.Label(self.ventana1, text="Solución")
        self.label3.grid(column=1, row=4)
        self.ventana1.mainloop()

    def calcular(self):
        if self.opcion.get()==1:
            suma=self.num1.get()+self.num2.get()
            self.label3.configure(text=suma)

        if self.opcion.get()==2:
            resta=self.num1.get()-self.num2.get()
```

```
            self.label3.configure(text=resta)

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:







Capítulo 225.- tkinter: Control Radiobutton – 3

Problema propuesto:

Disponer tres controles de tipo Radionutton con las etiquetas 'Rojo', Verde' y 'Azul'. Cuando se presione un botón cambiar el color del fondo del formulario.

Si consideramos que la variable ventana1 es un objeto de la clase Tk. Luego si queremos que el fondo sea de color rojo debemos llamar al método configure y en el parámetro bg indicar un string con el color a activar ("red", "green" o "blue"):

self.ventana1.congigure(bg="red")

Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Colores")
        self.opcion=tk.IntVar()
        self.radio1=tk.Radiobutton(self.ventana1, text="Rojo ",
variable=self.opcion, value=1)
        self.radio1.grid(column=0, row=0)
        self.radio2=tk.Radiobutton(self.ventana1, text="Verde",
variable=self.opcion, value=2)
        self.radio2.grid(column=0, row=1)
        self.radio3=tk.Radiobutton(self.ventana1, text="Azul ",
variable=self.opcion, value=3)
        self.radio3.grid(column=0, row=2)
        self.boton1=tk.Button(self.ventana1, text="Cambiar color",
command=self.cambiarcolor)
        self.boton1.grid(column=0, row=3)
        self.ventana1.mainloop()

    def cambiarcolor(self):
        if self.opcion.get()==1:
            self.ventana1.configure(bg="red")
        if self.opcion.get()==2:
            self.ventana1.configure(bg="green")
        if self.opcion.get()==3:
            self.ventana1.configure(bg="blue")

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:

Rojo
Verde
Azul
Cambiar color
Rojo
Verde
Azul
Cambiar color
Rojo
Verde
Azul
Cambiar color
Rojo
Verde
Azul
Cambiar color

Capítulo 226.- tkinter: Control Checkbutton – 1

El control visual Checkbutton permite implementar un botón de dos estados, más conocidos como un cuadro de selección.

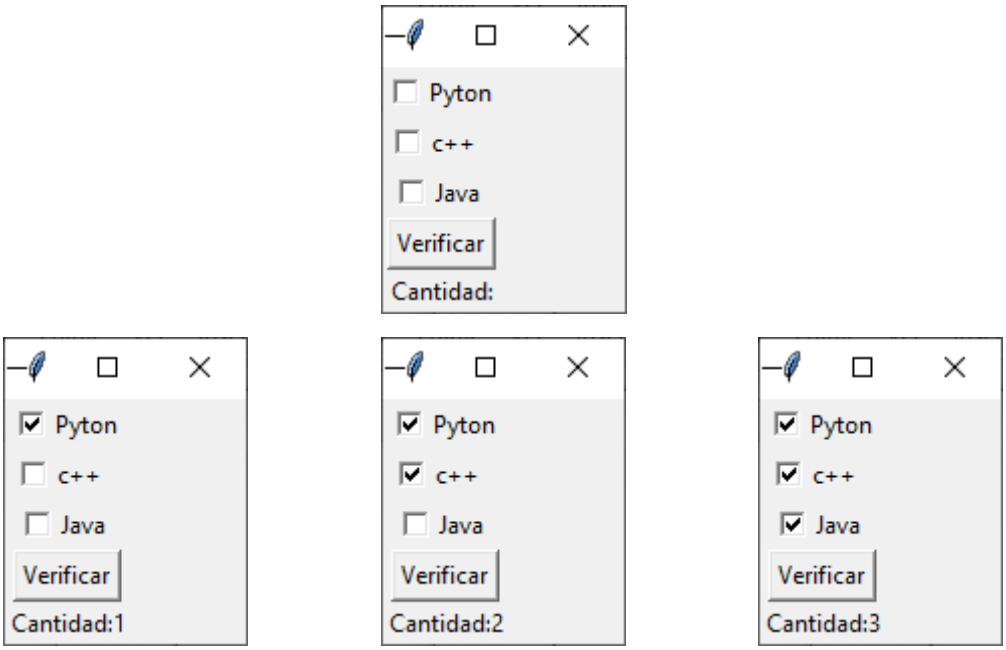
Problema:

Mostrar una ventana y en su interior tres controles de tipo Checkbutton cuyas etiquetas correspondan a distintos lenguajes de programación. Cuando se presione un botón mostrar en un Label la cantidad de Checkbutton que se encuentran chequeados.

Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.seleccion1=tk.IntVar()
        self.check1=tk.Checkbutton(self.ventana1, text="Pyton",
variable=self.seleccion1)
        self.check1.grid(column=0, row=0)
        self.seleccion2=tk.IntVar()
        self.check2=tk.Checkbutton(self.ventana1, text="c++  ",
variable=self.seleccion2)
        self.check2.grid(column=0, row=1)
        self.seleccion3=tk.IntVar()
        self.check3=tk.Checkbutton(self.ventana1, text="Java ",
variable=self.seleccion3)
        self.check3.grid(column=0, row=2)
        self.boton1=tk.Button(self.ventana1, text="Verificar",
command=self.verificar)
        self.boton1.grid(column=0, row=3)
        self.label1=tk.Label(self.ventana1, text="Cantidad:")
        self.label1.grid(column=0, row=4)
        self.ventana1.mainloop()

    def verificar(self):
        cant=0
        if self.seleccion1.get()==1:
            cant=cant+1
        if self.seleccion2.get()==1:
            cant=cant+1
        if self.seleccion3.get()==1:
            cant=cant+1
        self.label1.configure(text="Cantidad:"+str(cant))

# Bloque principal
aplicacion1=Aplicacion()
```

Vamos a ejecutar:



Capítulo 227.- tkinter: Control Checkbutton – 2

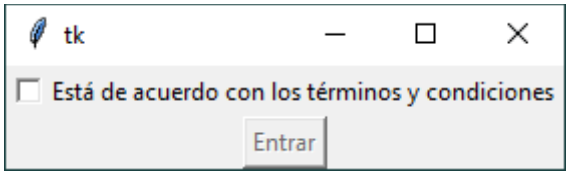
Problema:

Disponer un control Checkbutton que muestre el siguiente mensaje: ¿Está de acuerdo con los términos y condiciones?, además agregar un Botón desactivo. Cuando se tilde el Checkbutton inmediatamente activar el botón.

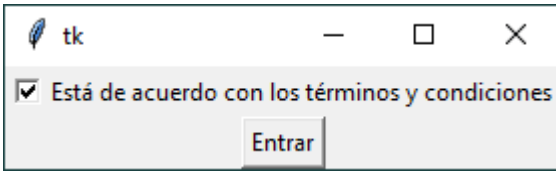
Vamos a programar:

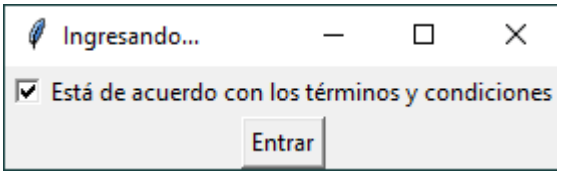
```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.seleccion=tk.IntVar()
        self.check1=tk.Checkbutton(self.ventana1, text="Está de acuerdo
con los términos y condiciones", variable=self.seleccion,
command=self.cambiarestado)
        self.check1.grid(column=0, row=0)
        self.boton1=tk.Button(self.ventana1, text="Entrar",
state="disabled", command=self.ingresar)
        self.boton1.grid(column=0, row=1)
        self.ventana1.mainloop()

    def cambiarestado(self):
        if self.seleccion.get()==1:
            self.boton1.configure(state="normal")
        if self.seleccion.get()==0:
            self.boton1.configure(state="disabled")

    def ingresar(self):
        self.ventana1.title("Ingresando...")

# Bloque principal
aplicacion1=Aplicacion()
```

Vamos a ejecutar:



Activamos el checkbutton de este modo estará activo el botón.





Seleccionamos el botón y muestra el mensaje "Ingresando..." en el título.

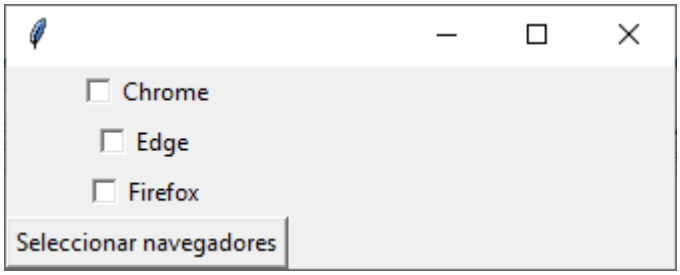
Capítulo 228.- tkinter: Control Checkbutton – 3

Problema propuesto:

Disponer varios objetos de la clase Checbutton con nombres de navegadores web. En el título de la ventana mostrar todos los nombres de navegadores seleccionados.
Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.opcion1=tk.IntVar()
        self.check1=tk.Checkbutton(self.ventana1, text="Chrome",
variable=self.opcion1)
        self.check1.grid(column=0, row=0)
        self.opcion2=tk.IntVar()
        self.check2=tk.Checkbutton(self.ventana1, text="Edge ",
variable=self.opcion2)
        self.check2.grid(column=0, row=1)
        self.opcion3=tk.IntVar()
        self.check3=tk.Checkbutton(self.ventana1, text="Firefox",
variable=self.opcion3)
        self.check3.grid(column=0, row=2)
        self.boton1=tk.Button(self.ventana1, text="Seleccionar
navegadores", command=self.navegadores)
        self.boton1.grid(column=0, row=3)
        self.ventana1.mainloop()

    def navegadores(self):
        mensaje=""
        if self.opcion1.get()==1:
            mensaje=mensaje+"Chrome "
        if self.opcion2.get()==1:
            mensaje=mensaje+"Edge "
        if self.opcion3.get()==1:
            mensaje=mensaje+"Firefox"
        self.ventana1.title(mensaje)


# Bloque principal
aplicacion1=Aplicacion()
```

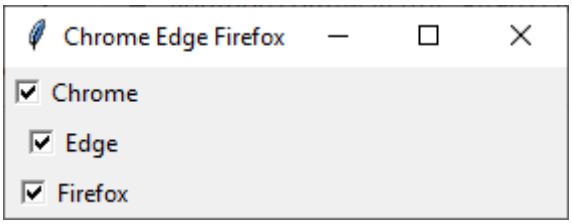
Este será el resultado:



También lo podemos hacer sin el botón:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.opcion1=tk.IntVar()
        self.check1=tk.Checkbutton(self.ventana1, text="Chrome",
variable=self.opcion1, command=self.navegadores)
        self.check1.grid(column=0, row=0)
        self.opcion2=tk.IntVar()
        self.check2=tk.Checkbutton(self.ventana1, text="Edge ",
variable=self.opcion2, command=self.navegadores)
        self.check2.grid(column=0, row=1)
        self.opcion3=tk.IntVar()
        self.check3=tk.Checkbutton(self.ventana1, text="Firefox",
variable=self.opcion3, command=self.navegadores)
        self.check3.grid(column=0, row=2)
        self.ventana1.mainloop()

    def navegadores(self):
        mensaje=""
        if self.opcion1.get()==1:
            mensaje=mensaje+"Chrome "
        if self.opcion2.get()==1:
            mensaje=mensaje+"Edge "
        if self.opcion3.get()==1:
            mensaje=mensaje+"Firefox"
        self.ventana1.title(mensaje)


# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



Capítulo 229.- tkinter: Control Listbox – 1

El control visual Listbox se emplea para mostrar una lista de elementos.

El usuario puede seleccionar uno a más elementos de la lista según como se lo configure al crearlo.

Problema:

Disponer un Listbox con una serie de nombres de frutas.

Permitir la selección solo de uno de ellos. Cuando se presione un botón recuperar la fruta seleccionada y mostrarla en un Label.

Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.listbox1=tk.Listbox(self.ventana1)
        self.listbox1.grid(column=0, row=0)
        self.listbox1.insert(0,"Patatas")
        self.listbox1.insert(1,"Manzanas")
        self.listbox1.insert(2,"Melón")
        self.listbox1.insert(3,"Naranjas")
        self.listbox1.insert(4,"Uvas")
        self.listbox1.insert(5,"Cerezas")
        self.boton1=tk.Button(self.ventana1, text="Recuperar",
command=self.recuperar)
        self.boton1.grid(column=0, row=1)
        self.label1=tk.Label(self.ventana1, text="Seleccionado:")
        self.label1.grid(column=0, row=2)
        self.ventana1.mainloop()

    def recuperar(self):
        if (len(self.listbox1.curselection()))!=0:
            self.label1.configure(text=self.listbox1.get(self.listbox1.cu
rselection()[0]))

# Bloque principal

aplicacion1=Aplicacion()
```

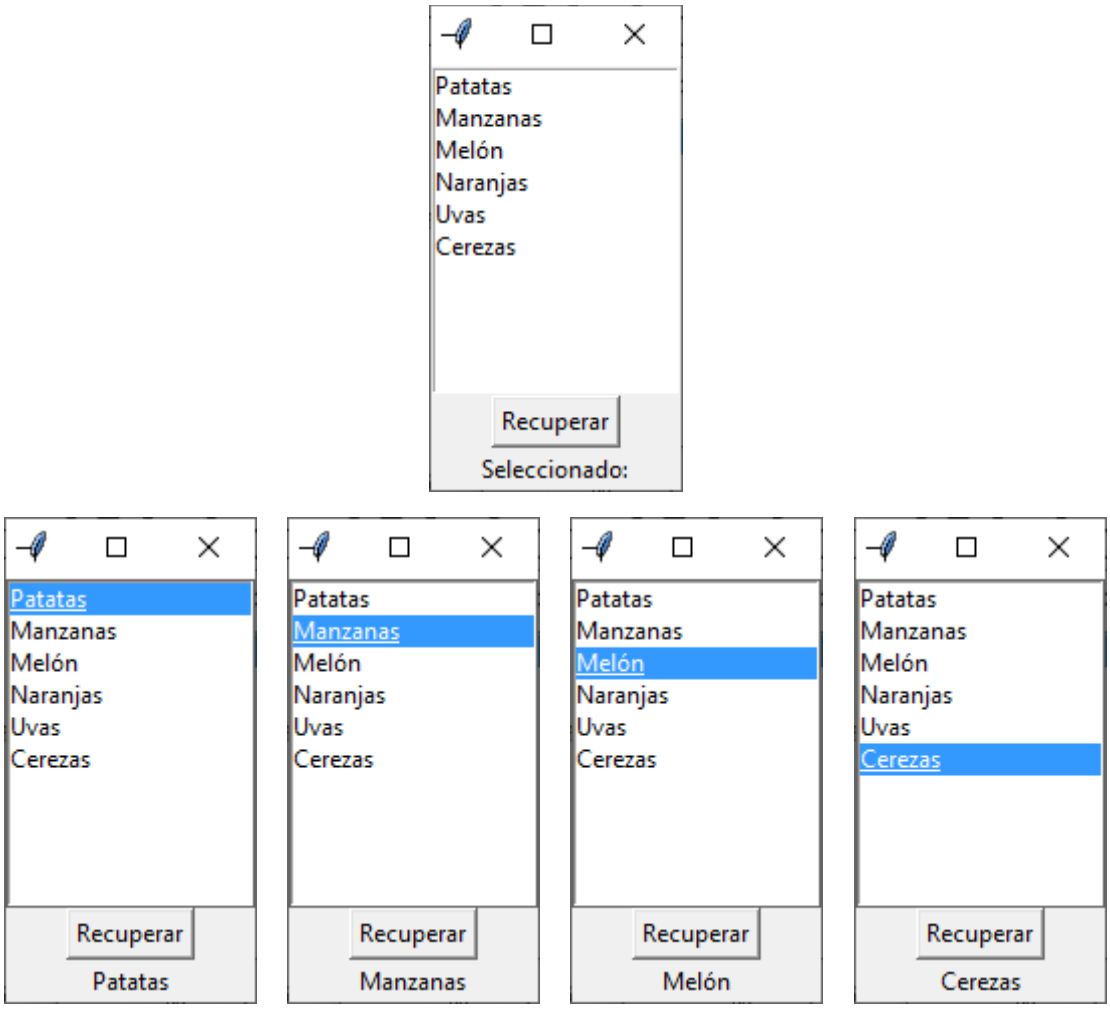
Vamos a comentar la función recuperar:

(len(self.listbox1.curselection())) → el número de objetos que hemos seleccionado.

self.listbox1curselection()[] → Retorna el número de la lista seleccionada que puede ser un valor del 0 hasta el 5.

Por ejemplo:
self.listbox1.get(0) retorna el primer elemento, Patatas.
self.listbox1.get(1) retorna el segundo elemento, Manzanas.
self.listbox1.get(2) retorna el tercer elemento, Melón.
self.listbox1.get(3) retorna el cuarto elemento, Naranjas.
self.listbox1.get(4) retorna el quinto elemento, Uvas.
self.listbox1.get(5) retorna el sexto elemento, Cerezas.

Este será el resultado:



Capítulo 230.- tkinter: Control Listbox – 2

Problema:

Disponer un Listbox con una serie de nombres de fruta.
Permitir la selección de varias frutas.
Cuando se presione un botón recuperar todas las frutas seleccionadas y mostrarlas en un Label.

Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.listbox1=tk.Listbox(self.ventana1, selectmode=tk.MULTIPLE)
        self.listbox1.grid(column=0, row=0)
        self.listbox1.insert(0,"Uvas")
        self.listbox1.insert(1,"Naranjas")
        self.listbox1.insert(2,"Limón")
        self.listbox1.insert(3,"Sandia")
        self.listbox1.insert(4,"Melón")
        self.listbox1.insert(5,"Durazno")
        self.boton1=tk.Button(self.ventana1, text="Recuperar",
command=self.recuperar)
        self.boton1.grid(column=0, row=1)
        self.label1=tk.Label(self.ventana1, text="Seleccionado:")
        self.label1.grid(column=0, row=2)
        self.ventana1.mainloop()

    def recuperar(self):
        if len(self.listbox1.curselection())!=0:
            todos=""
            for posicion in self.listbox1.curselection():
                todos+=self.listbox1.get(posicion)+"\n"
            self.label1.configure(text=todos)

# Bloque principal
aplicacion1=Aplicacion()
```

Comentamos la función recuperar(self):

if len(self.listbox1.curselection())!=0: → Si los elementos seleccionados son distintos a 0.

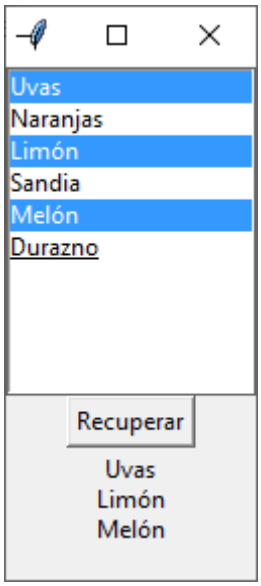
Definimos la variable todos="" → a vacío.

Realizamos un bucle recorre todos los elementos de la lista que están seleccionados y en la variable posición se almacena el número de la lista.

En la variable todos vamos acumulando las frutas que tenemos seleccionadas más un salto de línea.

Por último le asignamos a la etiqueta label1 en texto, todo el contenido de la variable todos.

Vamos a ejecutar:



Capítulo 231.- tkinter: Control Listbox – 3

Problema:

Barra de scroll en un Listbox

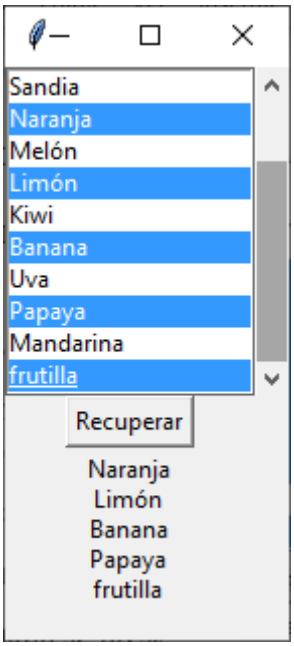
Por defecto no aparece una barra de scroll si la cantidad de elementos supera el tamaño del cuadro del Listbox.
Para que se muestre una barra de scroll la debemos crear y enlazar con el Listbox.
Resolveremos el mismo programa:
"Disponer un Listbox con una serie de nombres de frutas.
Permitir la selección de varias frutas.
Cuando se presione un botón de recuperar todas las frutas mencionadas y mostrarlas en un Label."

Vamos a programar:

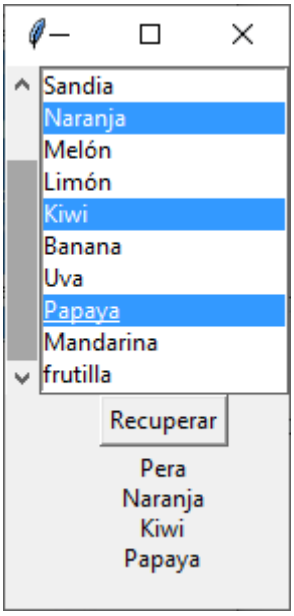
```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.scrollbar1=tk.Scrollbar(self.ventana1, orient=tk.VERTICAL)
        self.listbox1=tk.Listbox(self.ventana1, selectmode=tk.MULTIPLE,
yscrollcommand=self.scrollbar1.set)
        self.listbox1.grid(column=0, row=0)
        self.scrollbar1.configure(command=self.listbox1.yview)
        self.scrollbar1.grid(column=1, row=0, sticky="NS")
        self.listbox1.insert(0,"Patatas")
        self.listbox1.insert(1,"Manzanas")
        self.listbox1.insert(2,"Pera")
        self.listbox1.insert(3,"Sandia")
        self.listbox1.insert(4,"Naranja")
        self.listbox1.insert(5,"Melón")
        self.listbox1.insert(6,"Limón")
        self.listbox1.insert(7,"Kiwi")
        self.listbox1.insert(8,"Banana")
        self.listbox1.insert(9,"Uva")
        self.listbox1.insert(10,"Papaya")
        self.listbox1.insert(11,"Mandarina")
        self.listbox1.insert(12,"frutilla")
        self.boton1=tk.Button(self.ventana1, text="Recuperar",
command=self.recuperar)
        self.boton1.grid(column=0, row=1)
        self.label1=tk.Label(self.ventana1, text="Seleccionado:")
        self.label1.grid(column=0, row=2)
        self.ventana1.mainloop()

    def recuperar(self):
        if len(self.listbox1.curselection())!=0:
```

```
        todos=""
        for posicion in self.listbox1.curselection():
            todos+=self.listbox1.get(posicion)+"\n"
        self.label1.configure(text=todos)

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



Ahora te propongo que modifiques el código para modificar el formulario:



Capítulo 232.- tkinter: Control ListBox – 4

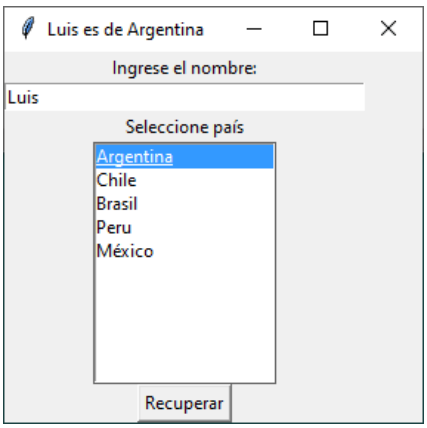
Problema propuesto:

Solicitar el ingreso del nombre de una persona y seleccionar de un control Listbox un país. Al presionar un botón mostrar en la barra de la ventana el nombre ingresado y el país seleccionado.

Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.label1=tk.Label(self.ventana1, text="Ingrese el nombre:" )
        self.label1.grid(column=0, row=0)
        self.nombre=tk.StringVar()
        self.entry1=tk.Entry(self.ventana1, width=40,
textvariable=self.nombre)
        self.entry1.grid(column=0, row=1)
        self.label2=tk.Label(self.ventana1, text="Seleccione país" )
        self.label2.grid(column=0, row=2)
        self.listbox1=tk.Listbox(self.ventana1)
        self.listbox1.grid(column=0, row=3)
        self.listbox1.insert(0, "Argentina")
        self.listbox1.insert(1, "Chile")
        self.listbox1.insert(2, "Brasil")
        self.listbox1.insert(3, "Peru")
        self.listbox1.insert(4, "México")
        self.boton1=tk.Button(self.ventana1, text="Recuperar",
command=self.recuperar)
        self.boton1.grid(column=0, row=4)
        self.ventana1.mainloop()

    def recuperar(self):
        if(len(self.listbox1.curselection()))!=0:
            pais= self.listbox1.get(self.listbox1.curselection()[0])
            nom=self.nombre.get()
            mensaje=nom + " es de " + pais
            self.ventana1.title(mensaje)

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



Capítulo 233.- Módulo ttk – 1

Tk hemos dicho que es una biblioteca de controles visuales que los podemos acceder desde Python y desde otros lenguajes de programación.

En la versión Tk 8.5 sumó una nueva serie de controles visuales (Notebook, Combobox, etc.) y modernizó los que hemos visto en los conceptos anteriores. Para hacer uso de esta nueva versión de biblioteca en Python se implementó un nuevo módulo y se lo agregó al paquete tkinter.

Para hacer uso de este conjunto de Widget (controles visuales) debemos importar el paquete ttk.

Todo lo que conocemos hasta ahora de los controles visuales del módulo tkinder funciona prácticamente sin cambios, lo que deberemos hacer es crear objetos de la clase Button, Entry, etc. recuperándolos ahora con el módulo tkinter.ttk.

Importar el módulo ttk

Debemos utilizar la siguiente sintaxis para importar el nuevo módulo:

from tkinter import ttk

Veamos algunos de los ejemplos ya resueltos pero haciendo uso de los Widget de este nuevo módulo.

Problema:

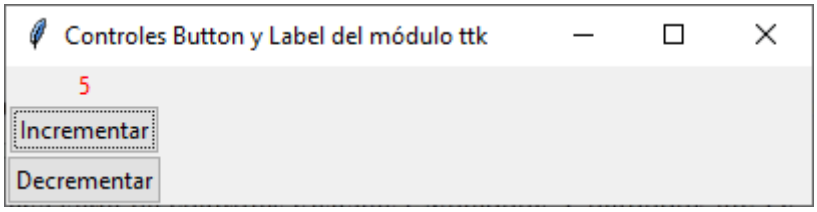
Mostrar una ventana y en su interior dos botones y un label utilizando el módulo ttk. El label muestra inicialmente el valor 1. Cada uno de los botones permiten incrementar o decrementar en uno el contenido del label.

Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.valor=1
        self.ventana1=tk.Tk()
        self.ventana1.title("Controles Button y Label del módulo ttk")
        self.label1=ttk.Label(self.ventana1, text=self.valor)
        self.label1.grid(column=0, row=0)
        self.label1.configure(foreground="red")

        self.boton1=ttk.Button(self.ventana1, text="Incrementar",
command=self.incrementar)
        self.boton1.grid(column=0, row=1)

        self.boton2=ttk.Button(self.ventana1, text="Decrementar",
command=self.decrementar)
        self.boton2.grid(column=0, row=2)
```

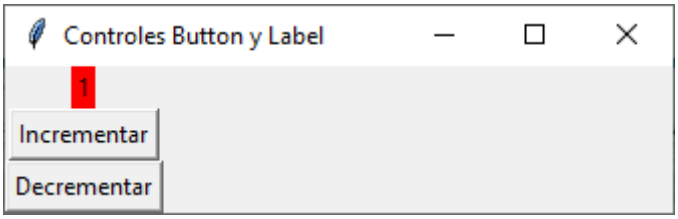
```
        self.ventana1.mainloop()

    def incrementar(self):
        self.valor+=1
        self.label1.configure(text=self.valor)

    def decrementar(self):
        self.valor-=1
        self.label1.configure(text=self.valor)

# Bloque principal
aplicacion1=Aplicacion()
```

Si ejecutamos este será el resultado:



Al seleccionar un botón este cambia de color, tiene un aspecto más a Windows.

En el ejemplo anterior del capítulo 215 este era el resultado:



Capítulo 234.- Módulo ttk – 2

Resolvemos otros problemas ya resueltos empleando el nuevo módulo de Widged propuesto en el módulo tkinter.ttk.

Problema:

Ingresar el nombre de usuario y clave en controles de tipo Entry.

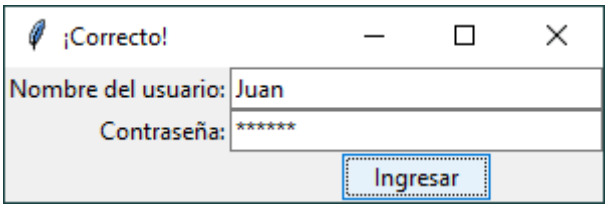
Si se ingresa la cadena (usuario: Juan y clave="abc123" luego mostrar en el título de la ventana el mensaje "Correcto" en caso contrario mostrar el mensaje "Incorrecto".

Utilizar Widget del módulo ttk.

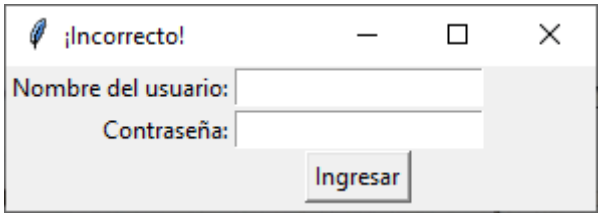
Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Usuario y contraseña")
        self.label1=ttk.Label(self.ventana1, text="Nombre del usuario:")
        self.label1.grid(column=0, row=0)
        self.usuario=tk.StringVar()
        self.entry1=ttk.Entry(self.ventana1, width=30,
textvariable=self.usuario)
        self.entry1.grid(column=1, row=0)
        self.label2=ttk.Label(self.ventana1,
text="               Contraseña:")
        self.label2.grid(column=0, row=1)
        self.contrasena=tk.StringVar()
        self.entry2=ttk.Entry(self.ventana1, width=30,
textvariable=self.contrasena, show="*")
        self.entry2.grid(column=1, row=1)
        self.boton1=ttk.Button(self.ventana1, text="Ingresar",
command=self.ingresar)
        self.boton1.grid(column=1, row=2)
        self.ventana1.mainloop()

    def ingresar(self):
        if self.usuario.get()=="Juan" and
self.contrasena.get()=="abc123":
            self.ventana1.title("¡Correcto!")
        else:
            self.ventana1.title("¡Incorrecto!")

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



Lo vamos a comparar con el ejercicio anterior utilizando el módulo tk.



Capítulo 235.- Módulo ttk – 3

Resolvemos otros problemas ya resueltos empleando el nuevo módulo de Widged propuesto en el módulo tkinter.tk.

Problema:

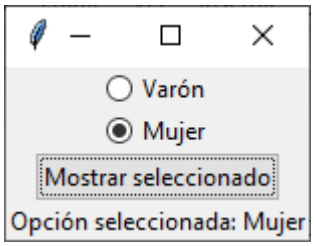
Mostrar dos controles de tipo Radiobuttom con las etiquetas "Varón" y "Mujer", cuando se presione un botón actualizar un label con el Radiobutton seleccionado.

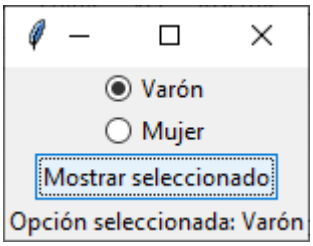
Utilizar Widgets del módulo ttk.

Vamos a programar:

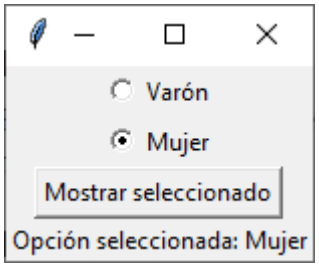
```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.seleccion=tk.IntVar()
        self.seleccion.set(2)
        self.radio1=ttk.Radiobutton(self.ventana1, text="Varón",
variable=self.seleccion, value=1)
        self.radio1.grid(column=0, row=0)
        self.radio2=ttk.Radiobutton(self.ventana1, text="Mujer",
variable=self.seleccion, value=2)
        self.radio2.grid(column=0, row=1)
        self.boton1=ttk.Button(self.ventana1, text="Mostrar
seleccionado", command=self.mostrarseleccionado)
        self.boton1.grid(column=0, row=2)
        self.label1=ttk.Label(self.ventana1, text="Opción seleccionada:")
        self.label1.grid(column=0, row=3)
        self.ventana1.mainloop()

    def mostrarseleccionado(self):
        if self.seleccion.get()==1:
            self.label1.configure(text="Opción seleccionada: Varón")
        if self.seleccion.get()==2:
            self.label1.configure(text="Opción seleccionada: Mujer")

# Bloque principal
aplicacion1=Aplicacion()
```

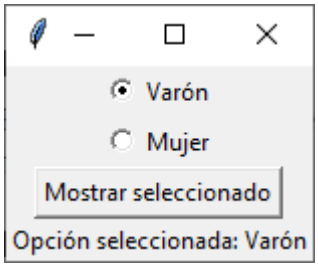
Este será el resultado:





En el ejemplo anterior usando el módulo tk.





Capítulo 236.- Módulo ttk – 4

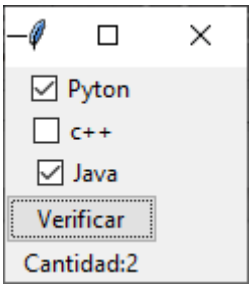
Problema:

Mostrar una ventana y en su interior tres controles de tipo Checkbutton cuyas etiquetas correspondan a distintos lenguajes de programación, cuando se presione un botón mostrar en un Label la cantidad de Checkbutton que se encuentran chequeados. Utilizar el módulo ttk.

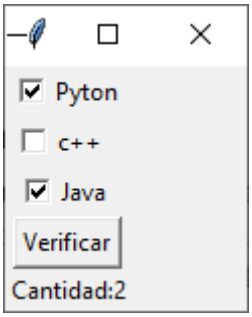
Vamos a programar:

```
class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.seleccion1=tk.IntVar()
        self.check1=ttk.Checkbutton(self.ventana1, text="Pyton",
variable=self.seleccion1)
        self.check1.grid(column=0, row=0)
        self.seleccion2=tk.IntVar()
        self.check2=ttk.Checkbutton(self.ventana1, text="c++  ",
variable=self.seleccion2)
        self.check2.grid(column=0, row=1)
        self.seleccion3=tk.IntVar()
        self.check3=ttk.Checkbutton(self.ventana1, text="Java ",
variable=self.seleccion3)
        self.check3.grid(column=0, row=2)
        self.boton1=ttk.Button(self.ventana1, text="Verificar",
command=self.verificar)
        self.boton1.grid(column=0, row=3)
        self.label1=ttk.Label(self.ventana1, text="Cantidad:")
        self.label1.grid(column=0, row=4)
        self.ventana1.mainloop()

    def verificar(self):
        cant=0
        if self.seleccion1.get()==1:
            cant=cant+1
        if self.seleccion2.get()==1:
            cant=cant+1
        if self.seleccion3.get()==1:
            cant=cant+1
        self.label1.configure(text="Cantidad:"+str(cant))

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



En el ejemplo anterior usando el módulo tk.



Capítulo 237.- Módulo ttk – 5

Acotaciones

El módulo ttk no implementa el Widget Listbox, pero podemos mezclar en una aplicación controles visuales de los dos paquetes.

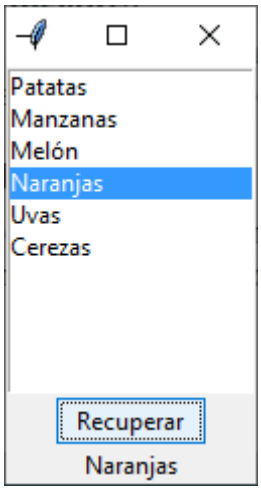
Problema:

Disponer un Listbox con una serie de nombres de frutas. Permitir la selección solo de uno de ellos. Cuando se presione un botón recuperar la fruta seleccionada y mostrarla en un Label.

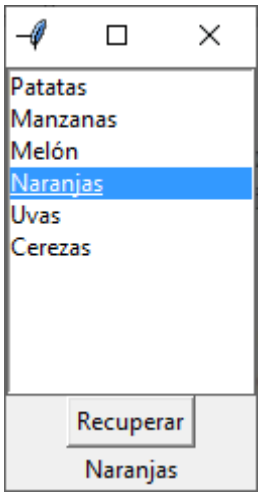
Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.listbox1=tk.Listbox(self.ventana1)
        self.listbox1.grid(column=0, row=0)
        self.listbox1.insert(0,"Patatas")
        self.listbox1.insert(1,"Manzanas")
        self.listbox1.insert(2,"Melón")
        self.listbox1.insert(3,"Naranjas")
        self.listbox1.insert(4,"Uvas")
        self.listbox1.insert(5,"Cerezas")
        self.boton1=ttk.Button(self.ventana1, text="Recuperar",
command=self.recuperar)
        self.boton1.grid(column=0, row=1)
        self.label1=ttk.Label(self.ventana1, text="Seleccionado:")
        self.label1.grid(column=0, row=2)
        self.ventana1.mainloop()

    def recuperar(self):
        if (len(self.listbox1.curselection()))!=0:
            self.label1.configure(text=self.listbox1.get(self.listbox1.cu
rselection()[0]))

# Bloque principal

aplicacion1=Aplicacion()
```

Este será el resultado:



En el ejemplo anterior usando el módulo tk.



Capítulo 238.- ttk: Combobox – 1

El control Combobox del paquete ttk permite seleccionar un string de un conjunto de elementos que se despliegan.

Podemos indicar cual elemento se muestre por defecto mediante el método 'current'.

Por defecto el operador puede además de seleccionar un elemento cargar por teclado cualquier cadena.

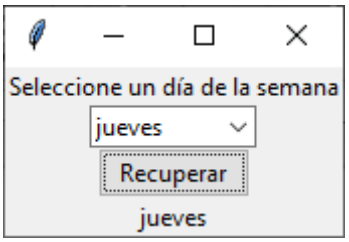
Problema:

Mostrar en una ventana un control de tipo Combobox con los días de la semana. Cuando se presione un botón actualiza un Label con el día seleccionado.
Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.label1=ttk.Label(self.ventana1, text="Seleccione un día de
la semana")
        self.label1.grid(column=0, row=0)
        self.opcion=tk.StringVar()
        diassemana=("lunes","martes","miercoles","jueves","viernes","saba
do","domingo")
        self.combobox1=ttk.Combobox(self.ventana1,
                                    width=10,
                                    textvariable=self.opcion,
                                    values=diassemana,
                                    state="readonly")
        self.combobox1.current(0)
        self.combobox1.grid(column=0, row=1)
        self.boton1=ttk.Button(self.ventana1, text="Recuperar",
command=self.recuperar)
        self.boton1.grid(column=0, row=2)
        self.label2=ttk.Label(self.ventana1, text="Días seleccionados:")
        self.label2.grid(column=0, row=3)
        self.ventana1.mainloop()

    def recuperar(self):
        self.label2.configure(text=self.opcion.get())

# Bloque principal
aplicacion1=Aplicacion()
```

No permite introducir un valor nuevo por teclado.

Este será el resultado:



Capítulo 239.- ttk: Control Combobox – 2

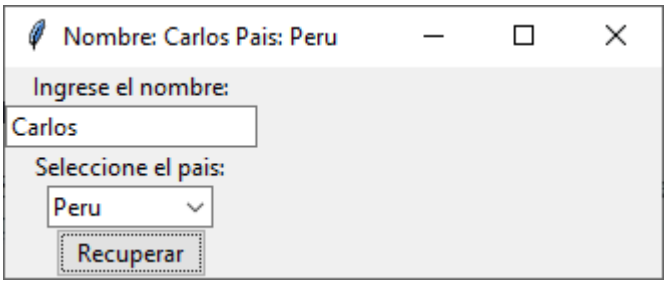
Problema propuesto:

Solicitar el ingreso del nombre de una persona y seleccionar de un control Combobox un país. Al presionar un botón mostrar en la barra de la ventana el nombre ingresado y el país seleccionado.

Vamos a programa:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.label1=ttk.Label(self.ventana1, text="Ingrese el nombre:")
        self.label1.grid(column=0, row=0)
        self.nombre=tk.StringVar()
        self.entry1=ttk.Entry(self.ventana1, width=20,
textvariable=self.nombre)
        self.entry1.grid(column=0, row=1)
        self.label2=ttk.Label(self.ventana1, text="Seleccione el pais:")
        self.label2.grid(column=0, row=2)
        paises=("Argentina", "Chile", "Brasil", "Peru", "México")
        self.opcion=tk.StringVar()
        self.combobox1=ttk.Combobox(self.ventana1,
                                    width=10,
                                    textvariable=self.opcion,
                                    values=paises,
                                    state="readonly")
        self.combobox1.grid(column=0, row=3)
        self.boton1=ttk.Button(self.ventana1, text="Recuperar",
command=self.recuperar)
        self.boton1.grid(column=0, row=4)
        self.ventana1.mainloop()

    def recuperar(self):
        mensaje="Nombre: " + self.nombre.get()
        mensaje+=" Pais: " + self.opcion.get()
        self.ventana1.title(mensaje)

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:

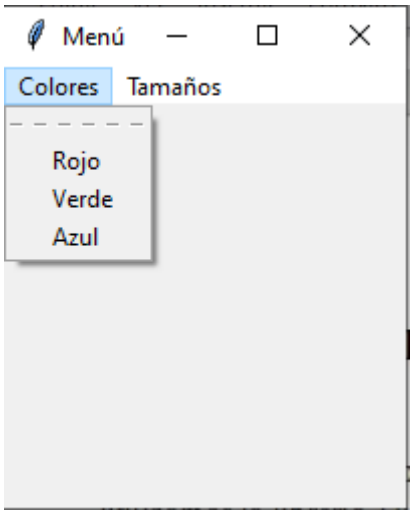


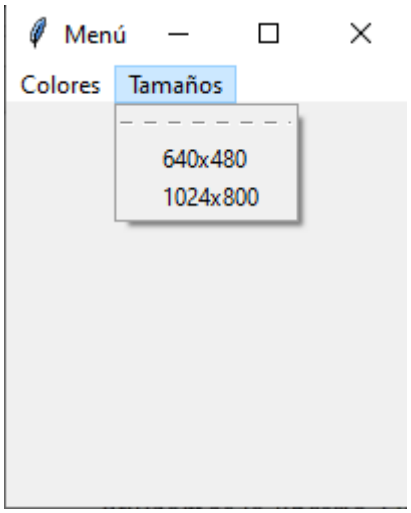
Capítulo 240.- tkinter: Control Menu – 1

Para implementar los típicos menú de barra horizontal que aparecen en las aplicaciones cuando utilizamos la librería Tk necesitamos crear objetos de la clase Menu que se encuentran declarada en el paquete tkinter y no es un paquete tkinter.ttk.

Problema:

Confeccionar una aplicación que muestre dos opciones de menú de la barra superior. La primera opción despliega un submenú que permita cambiar el color del formulario y la segunda permita cambiar el tamaño del formulario:



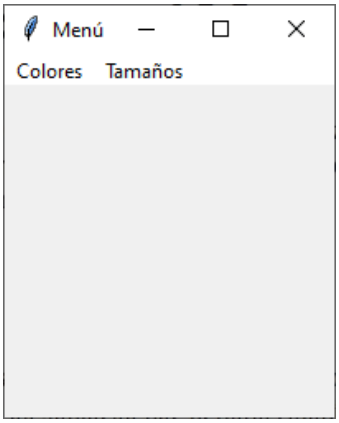


Vamos a programar:

```
import tkinter as tk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Menú")
        menubar1=tk.Menu(self.ventana1)
        self.ventana1.config(menu=menubar1)
        opciones1=tk.Menu(menubar1)
        opciones1.add_command(label="Rojo", command=self.fijarrojo)
        opciones1.add_command(label="Verde", command=self.fijarverde)
        opciones1.add_command(label="Azul", command=self.fijarazul)
        menubar1.add_cascade(label="Colores", menu=opciones1)
        opciones2=tk.Menu(menubar1)
        opciones2.add_command(label="640x480", command=self.ventanachica)
        opciones2.add_command(label="1024x800",command=self.ventanagrande
)
        menubar1.add_cascade(label="Tamaños", menu=opciones2)
        self.ventana1.mainloop()

    def fijarrojo(self):
        self.ventana1.configure(background="red")
```

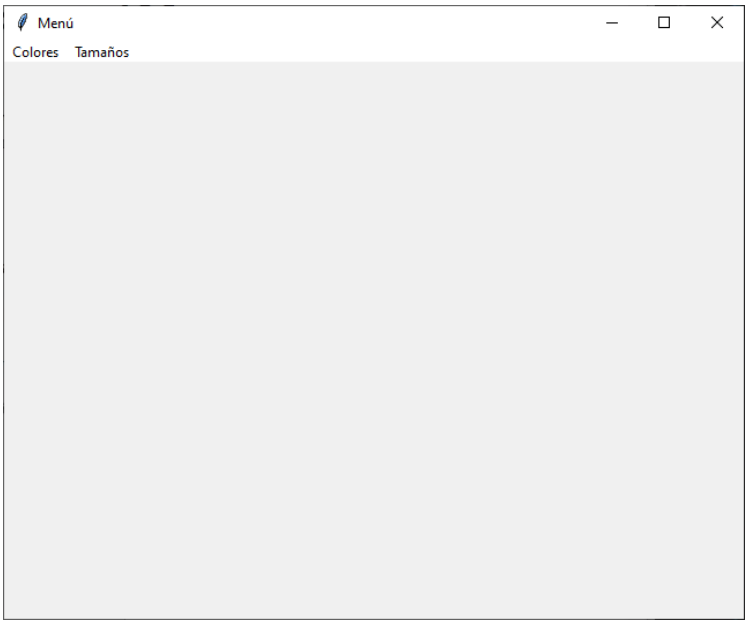
```
    def fijarverde(self):
        self.ventana1.configure(background="green")

    def fijarazul(self):
        self.ventana1.configure(background="blue")

    def ventanachica(self):
        self.ventana1.geometry("640x480")

    def ventanagrande(self):
        self.ventana1.geometry("1024x800")

# Bloque principal
aplicacion1=Aplicacion()
```

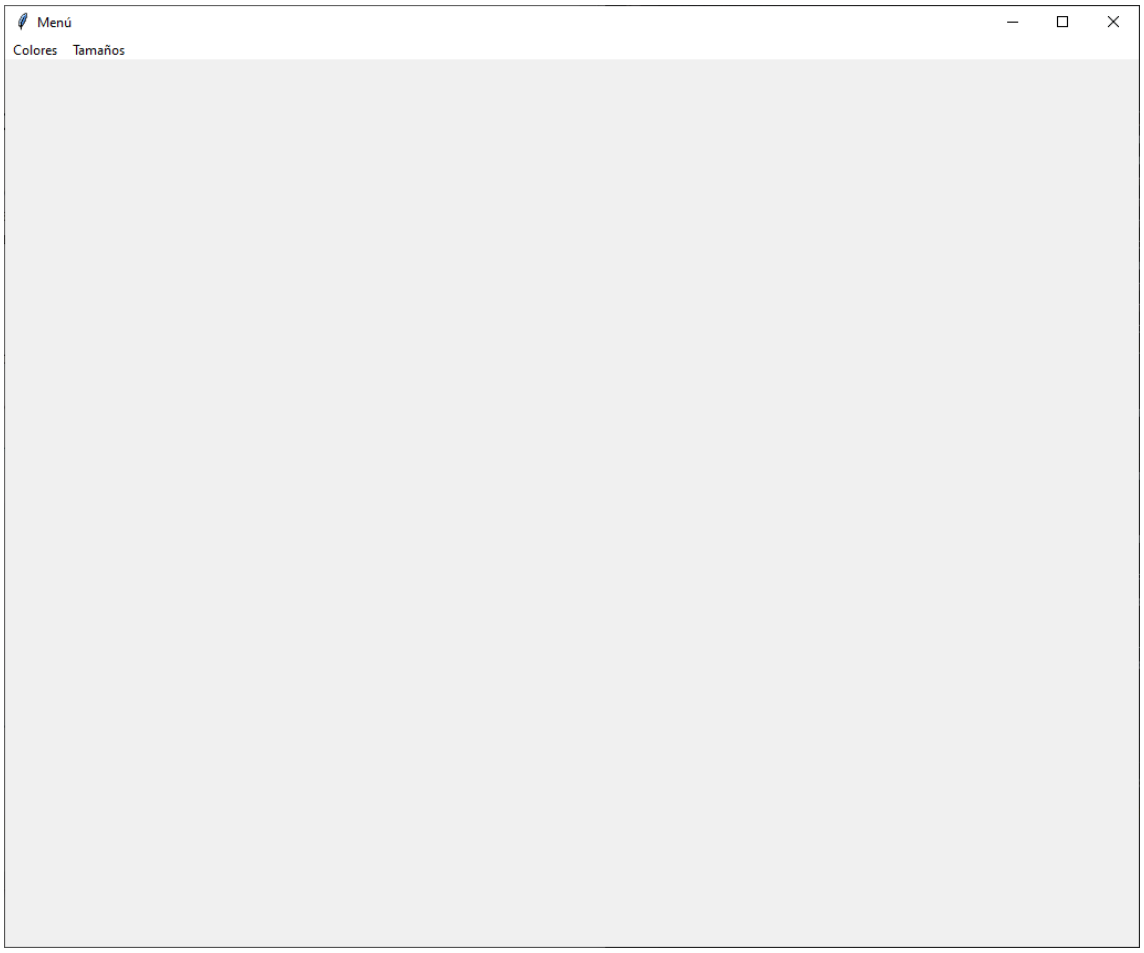
Este será el resultado:



Cambiar de color:

Cambiar de tamaño:



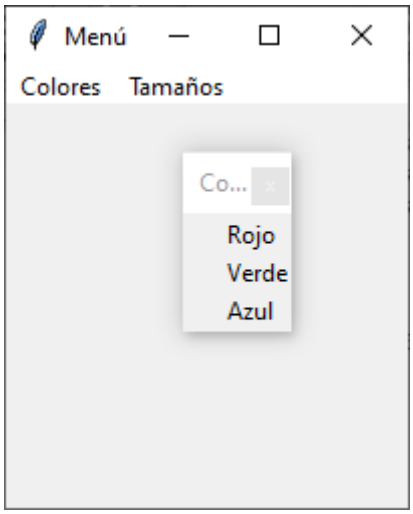


Variantes en un menú.

1.- Podemos agregar líneas separadoras entre opciones mediante la llamada al método 'add_separator':

```
opciones1 = tk.Menu(menubar1)
opciones1.add_command(label="Rojo", command=self.fijarrojo)
opciones1.add_command(label="Verde", command=self.fijarverde)
opcioones1.add_separator()
opciones.ade_command(label="Azul",  command=self.fijarazul)
```

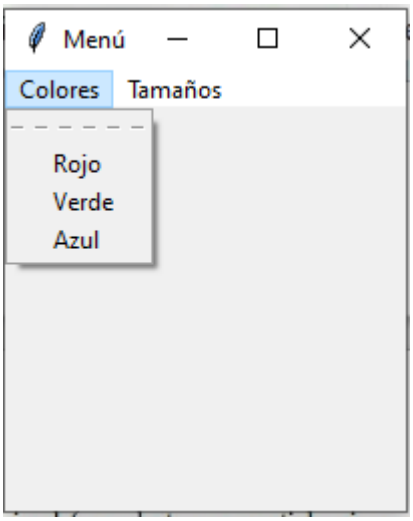
2.- Los menu desplegables muestran una línea de puntos antes de la primera opción. Si el operador hace clic sobre la misma el menú desplegable se muestra en una barra de herramientas fuera de la ventana principal (puede tener sentido si en un programa debemos seleccionar opciones de forma muy seguida y no queremos tener que estar abriendo el menú desplegable).



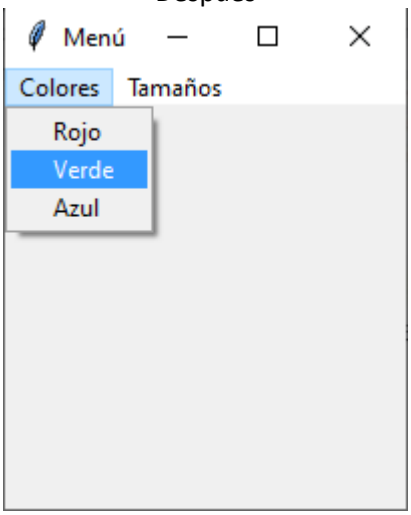
Si queremos evitar esta característica de los menús desplegables de tkinter debemos pasar el parámetro 'tearoff" con el valor 0:

```
opciones1 = tk.Menu(menubar1, tearoff=0)
opciones1.add_command(label="Rojo", command=self.fijarrojo)
opciones1.add_command(label="Verde", command=self.fijarverde)
opcioones1.add_separator()
opciones.ade_command(label="Azul",  command=self.fijarazul)
```

Antes



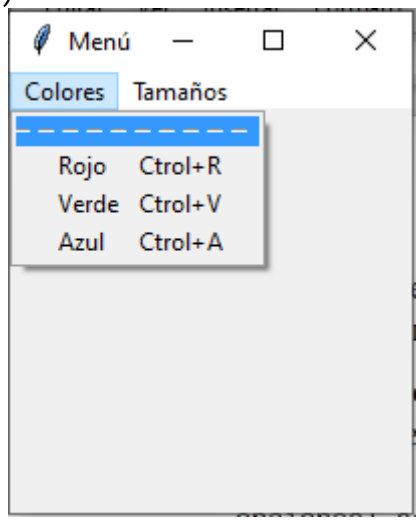
Después



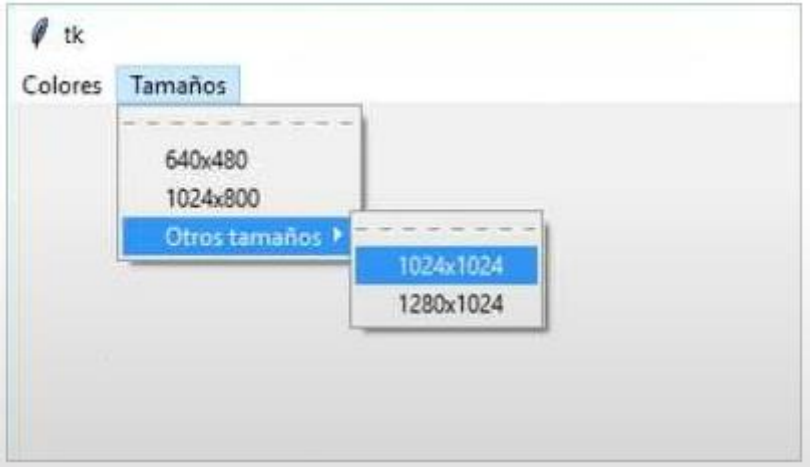
3.- Otra posibilidad es agregar teclas de acceso rápido a opciones de nuestro menú. Esto se resuelve la parte visual agregando el parámetro 'accelerator', y por otro lado asignar a cada una las teclas de acceso rápido la ejecución de un método:

```
import tkinter as tk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Menú")
        menubar1=tk.Menu(self.ventana1)
        self.ventana1.config(menu=menubar1)
        opciones1=tk.Menu(menubar1)
        opciones1.add_command(label="Rojo", command=self.fijarrojo,
accelerator="Ctrol+R")
        opciones1.add_command(label="Verde", command=self.fijarverde,
accelerator="Ctrol+V")
        opciones1.add_command(label="Azul", command=self.fijarazul,
accelerator="Ctrol+A")
        self.ventana1.bind_all("<Control-r>", self.cambiar)
        self.ventana1.bind_all("<Control-v>", self.cambiar)
        self.ventana1.bind_all("<Control-a>", self.cambiar)
        menubar1.add_cascade(label="Colores", menu=opciones1)
        opciones2=tk.Menu(menubar1)
        opciones2.add_command(label="640x480", command=self.ventanachica)
        opciones2.add_command(label="1024x800",command=self.ventanagrande
)
        menubar1.add_cascade(label="Tamaños", menu=opciones2)
        self.ventana1.mainloop()

    def cambiar(self, event):
        if event.keysym=="r":
            self.fijarrojo()
        if event.keysym=="v":
            self.fijarverde()
        if event.keysym=="a":
            self.fijarazul()

    def fijarrojo(self):
        self.ventana1.configure(background="red")

    def fijarverde(self):
        self.ventana1.configure(background="green")

    def fijarazul(self):
        self.ventana1.configure(background="blue")

    def ventanachica(self):
        self.ventana1.geometry("640x480")
```

```
    def ventanagrande(self):
        self.ventana1.geometry("1024x800")

# Bloque principal
aplicacion1=Aplicacion()
```

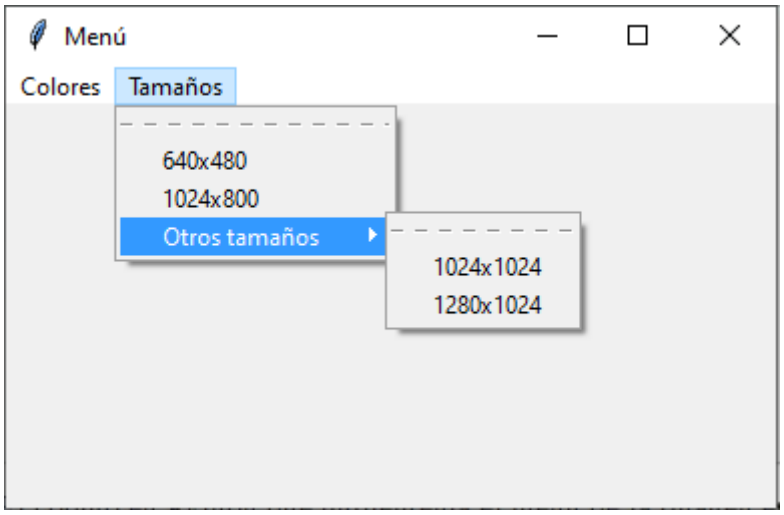


4.- Podemos disponer dentro de un menú desplegable una opción que despliegue otro submenú, la interface visual debe ser similar a esta:




```
import tkinter as tk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Menú")
        menubar1=tk.Menu(self.ventana1)
        self.ventana1.config(menu=menubar1)
        opciones1=tk.Menu(menubar1)
        opciones1.add_command(label="Rojo", command=self.fijarrojo,
accelerator="Ctrol+R")
        opciones1.add_command(label="Verde", command=self.fijarverde,
accelerator="Ctrol+V")
```

```
        opciones1.add_command(label="Azul", command=self.fijarazul,
accelerator="Ctrol+A")
        self.ventana1.bind_all("<Control-r>", self.cambiar)
        self.ventana1.bind_all("<Control-v>", self.cambiar)
        self.ventana1.bind_all("<Control-a>", self.cambiar)
        menubar1.add_cascade(label="Colores", menu=opciones1)
        opciones2=tk.Menu(menubar1)
        opciones2.add_command(label="640x480", command=self.ventanachica)
        opciones2.add_command(label="1024x800",command=self.ventanagrande
)
        submenu1=tk.Menu(menubar1)
        submenu1.add_command(label="1024x1024", command=self.tamano1)
        submenu1.add_command(label="1280x1024", command=self.tamano2)
        opciones2.add_cascade(label="Otros tamaños", menu=submenu1)
        menubar1.add_cascade(label="Tamaños", menu=opciones2)
        self.ventana1.mainloop()

    def cambiar(self, event):
        if event.keysym=="r":
            self.fijarrojo()
        if event.keysym=="v":
            self.fijarverde()
        if event.keysym=="a":
            self.fijarazul()

    def fijarrojo(self):
        self.ventana1.configure(background="red")

    def fijarverde(self):
        self.ventana1.configure(background="green")

    def fijarazul(self):
        self.ventana1.configure(background="blue")

    def ventanachica(self):
        self.ventana1.geometry("640x480")

    def ventanagrande(self):
        self.ventana1.geometry("1024x800")

    def tamano1(self):
        self.ventana1.geometry("1024x1024")

    def tamano2(self):
        self.ventana1.geometry("1280x1024")

# Bloque principal
aplicacion1=Aplicacion()
```


Menú
Colores
Tamaños
640x480
1024x800
Otros tamaños
1024x1024
1280x1024

Capítulo 241.- tkinter: Control Menu – 2

Problema propuesto:

Mediante dos controles de tipo Entry permitir el ingreso de dos números. Crear un menú que contenga una opción que cambie el tamaño de la ventana con los valores ingresados por teclado.

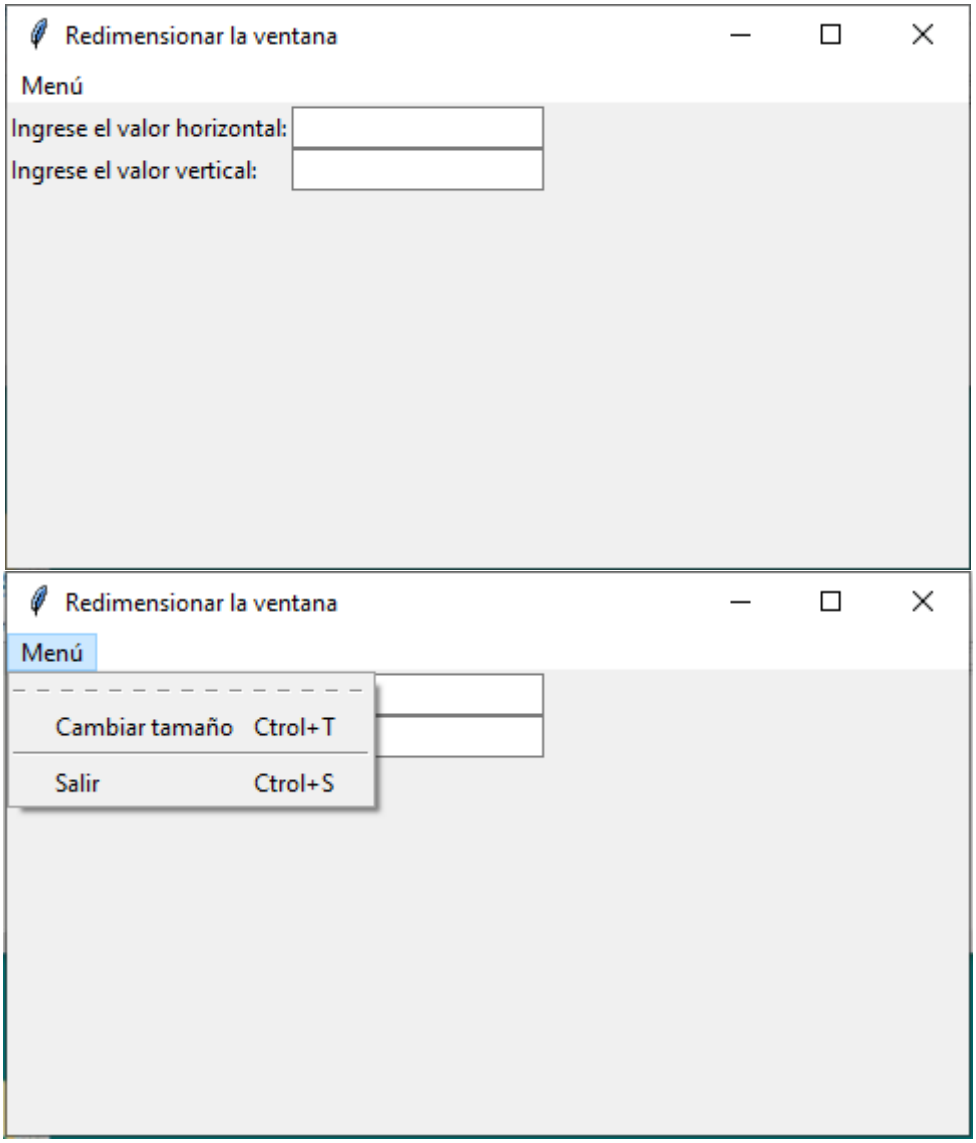
Finalmente disponer otra opción que finalice el programa.

Vamos a programar:

```
import tkinter as tk
from tkinter import ttk
import sys

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Redimensionar la ventana")
        self.ventana1.geometry("480x230")
        self.label1=ttk.Label(self.ventana1, text="Ingrese el valor
horizontal:")
        self.label1.grid(column=0, row=0)
        self.dimensionh=tk.StringVar()
        self.entry1=ttk.Entry(self.ventana1, width=20,
textvariable=self.dimensionh)
        self.entry1.grid(column=1, row=0)

        self.label2=ttk.Label(self.ventana1, text="Ingrese el valor
vertical:     ")
        self.label2.grid(column=0, row=1)
        self.dimensionv=tk.StringVar()
        self.entry1=ttk.Entry(self.ventana1, width=20,
textvariable=self.dimensionv)
        self.entry1.grid(column=1, row=1)

        menubar1=tk.Menu(self.ventana1)
        self.ventana1.config(menu=menubar1)
        opciones1=tk.Menu(menubar1)
        opciones1.add_command(label="Cambiar tamaño",
                              command=self.modificarTam,
                              accelerator="Ctrol+T")
        opciones1.add_separator()
        opciones1.add_command(label="Salir",
                              command=self.salir,
                              accelerator="Ctrol+S")
        menubar1.add_cascade(label="Menú", menu=opciones1)

        self.ventana1.bind_all("<Control-t>", self.cambiar)
        self.ventana1.bind_all("<Control-s>", self.cambiar)
```

```
        self.ventana1.mainloop()

    def cambiar(self, event):
        if event.keysym=="t":
            self.modificarTam()
        if event.keysym=="s":
            self.salir()

    def salir(self):
        sys.exit(0)

    def modificarTam(self):
        modificar=self.dimensionh.get()+"x"+self.dimensionv.get()
        self.ventana1.geometry(modificar)

# Bloque principal
aplicacion1=Aplicacion()
```

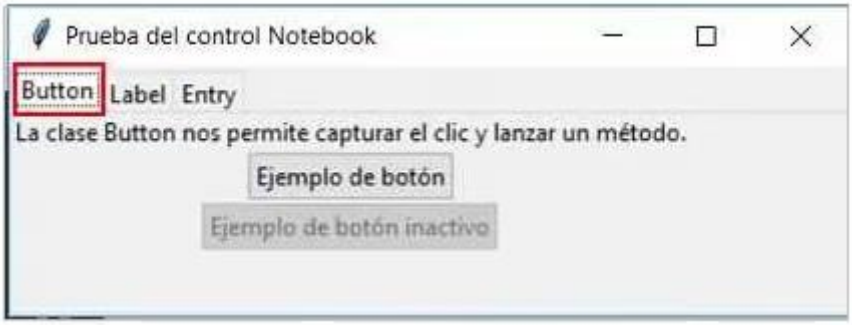
Este será el resultado:



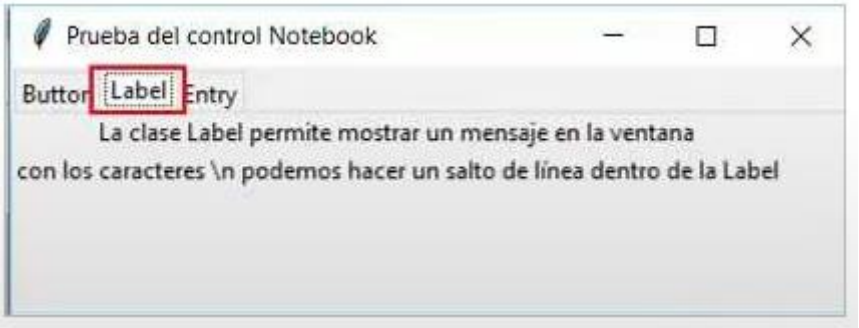
Capítulo 242.- ttk: Controles Notebook y Frame

La clase NoteBook nos permite crear un cuaderno con una serie de pestañas en la parte superior. En cada pestaña asociamos un objeto de la clase Frame y dentro de esta podemos disponer distintos controles visuales que hemos visto hasta ahora como pueden ser Label, Button, Radiubutton, Checkbutton, Entry, etc.

Eu ejemplo visual de un Notebook es:



Según la pestaña seleccionada se muestra un Frame con distintos controles visuales.



Problema:

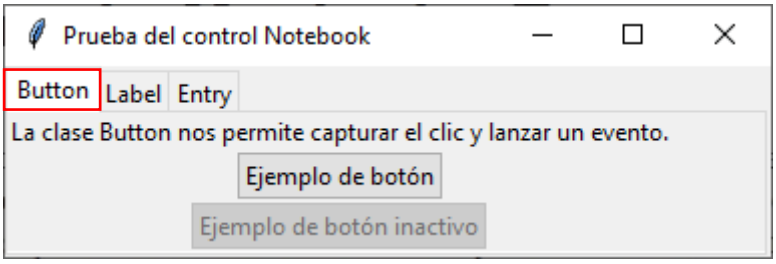
Confeccionar una aplicación que muestre un cuaderno con tres pestañas. Los títulos de cada pestaña deben ser 'Button', 'Label' y 'Entry'. Según la pestaña seleccionada mostrar un mensaje informando el objeto de la clase y un objeto de la misma.

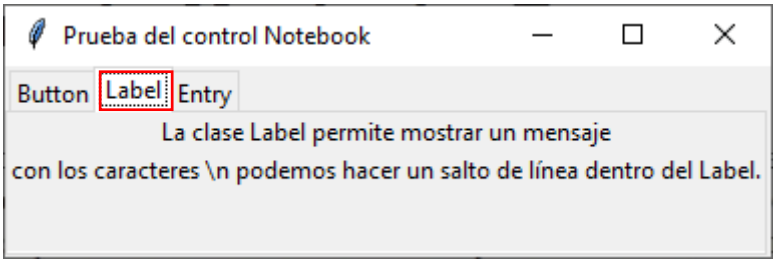
Vamos a programar:

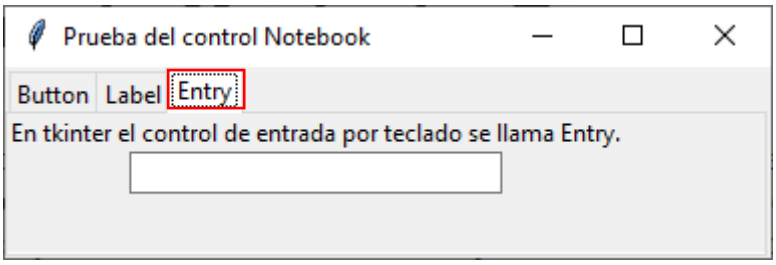
```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Prueba del control Notebook")
        self.cuaderno1=ttk.Notebook(self.ventana1)

        self.pagina1=ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina1, text="Button")
        self.label1=ttk.Label(self.pagina1, text="La clase Button nos
permite capturar el clic y lanzar un evento.")
```

```
        self.label1.grid(column=0, row=0)
        self.boton1=ttk.Button(self.pagina1, text="Ejemplo de botón")
        self.boton1.grid(column=0, row=1)
        self.boton2=ttk.Button(self.pagina1, text="Ejemplo de botón
inactivo", state="disabled")
        self.boton2.grid(column=0, row=2)

        self.pagina2=ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina2, text="Label")
        self.label2=ttk.Label(self.pagina2, text="La clase Label permite
mostrar un mensaje")
        self.label2.grid(column=0, row=0)
        self.label3=ttk.Label(self.pagina2, text="con los caracteres \\n
podemos hacer un salto de línea dentro del Label.")
        self.label3.grid(column=0, row=1)

        self.pagina3=ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina3, text="Entry")
        self.label4=ttk.Label(self.pagina3, text="En tkinter el control
de entrada por teclado se llama Entry.")
        self.label4.grid(column=0, row=0)
        self.entry1=ttk.Entry(self.pagina3, width=30)
        self.entry1.grid(column=0, row=1)

        self.cuaderno1.grid(column=0, row=0)

        self.ventana1.mainloop()

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:





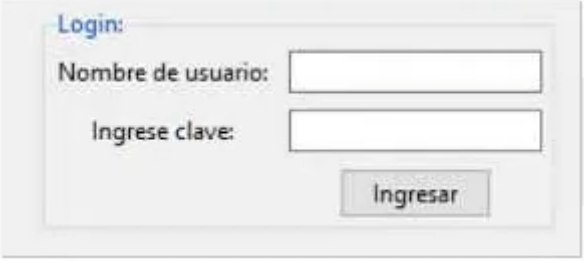

Prueba del control Notebook
Button
Label
Entry
En tkinter el control de entrada por teclado se llama Entry.

Capítulo 243.- ttk: Control LabelFrame – 1

El control visual LabelFrame tiene una funcionalidad casi idéntica a la componente Freme que vimos en el concepto anterior, la única diferencia es que agrega un texto en la parte superior del Frame y hace un recuadro alrededor del mismo.

El Winget LabelFrame es un contenedor donde podemos agregar en su interior otros Widget como Button, Label, Entry, Radiobutton, etc.

La representación visual de un LabelFrame que tiene 2 Label, 2 Entry y un Button es:

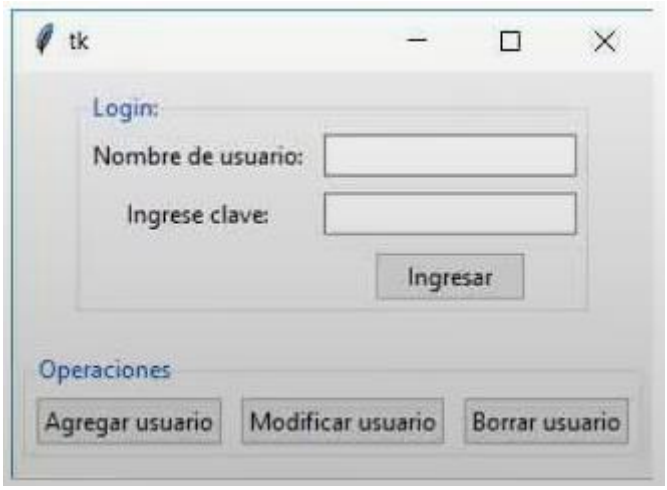


Problema:

Confeccionar una aplicación que muestre dos controles de tipo LabelFrame.

En el primero diponer 2 Label, 2 Entry y un Button, en el segundo LabelFrame disponer botones.

La representación visual debe ser:

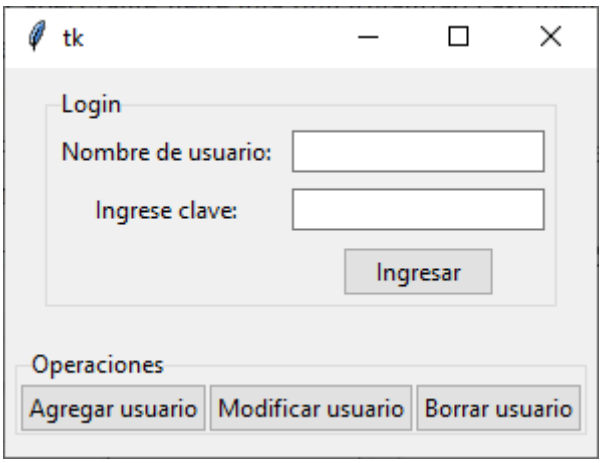


Vamos a programar:

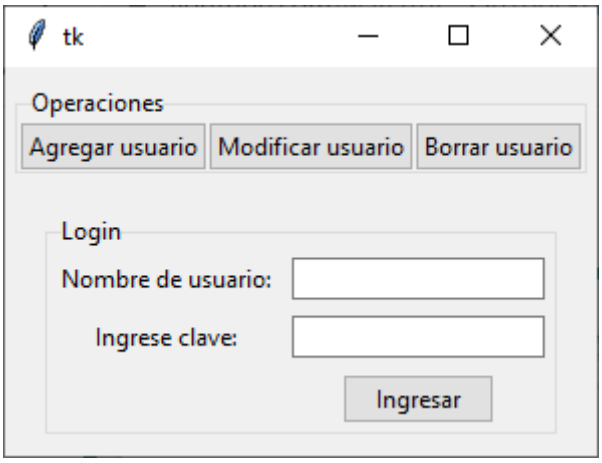
```
import tkinter as tk
from tkinter import ttk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.labelframe1=ttk.LabelFrame(self.ventana1, text="Login")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.login()
```

```
        self.labelframe2=ttk.LabelFrame(self.ventana1,
text="Operaciones")
        self.labelframe2.grid(column=0, row=1, padx=5, pady=10)
        self.operaciones()
        self.ventana1.mainloop()

    def login(self):
        self.label1=ttk.Label(self.labelframe1, text="Nombre de
usuario:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.entry1=ttk.Entry(self.labelframe1)
        self.entry1.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1, text="Ingrese clave:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.entry2=ttk.Entry(self.labelframe1, show="*")
        self.entry2.grid(column=1, row=1, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Ingresar")
        self.boton1.grid(column=1, row=2, padx=4, pady=4)

    def operaciones(self):
        self.boton2=ttk.Button(self.labelframe2, text="Agregar usuario")
        self.boton2.grid(column=0, row=0, padx=0, pady=0)
        self.boton3=ttk.Button(self.labelframe2, text="Modificar
usuario")
        self.boton3.grid(column=1, row=0, padx=0, pady=0)
        self.boton4=ttk.Button(self.labelframe2, text="Borrar usuario")
        self.boton4.grid(column=2, row=0, padx=0, pady=0)

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



Modifica el código para que quede de la siguiente forma:



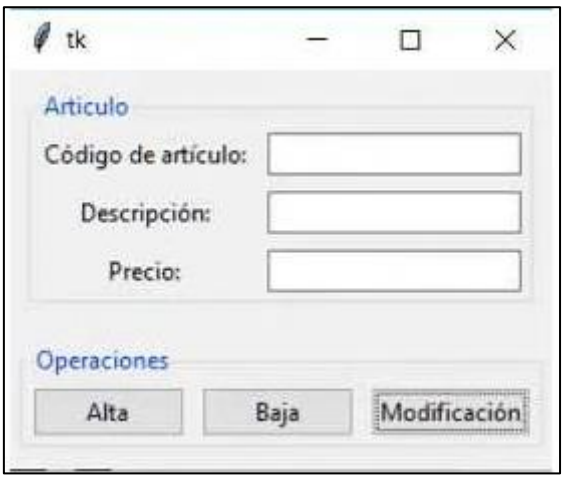
Solución:

```
class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.labelframe1=ttk.LabelFrame(self.ventana1, text="Login")
        self.labelframe1.grid(column=0, row=1, padx=5, pady=10)
        self.login()
        self.labelframe2=ttk.LabelFrame(self.ventana1,
text="Operaciones")
        self.labelframe2.grid(column=0, row=0, padx=5, pady=10)
        self.operaciones()
        self.ventana1.mainloop()
```

Capítulo 244.- ttk: Control LabelFrame – 2

Problema propuesto:

Mediante los controles de tipo LabelFrame implementa la siguiente inferfazz visual:

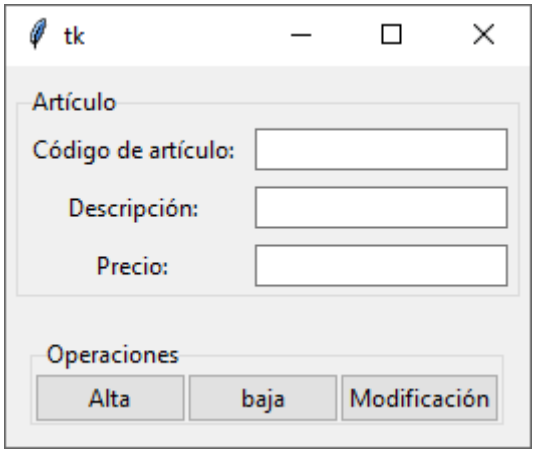


Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.labelframe1=ttk.LabelFrame(self.ventana1, text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.articulo()
        self.labelframe2=ttk.LabelFrame(self.ventana1,
text="Operaciones")
        self.labelframe2.grid(column=0, row=1, padx=5, pady=10)
        self.operaciones()
        self.ventana1.mainloop()

    def articulo(self):
        self.label1=ttk.Label(self.labelframe1, text="Código de
artículo:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.entry1=ttk.Entry(self.labelframe1)
        self.entry1.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1, text="Descripción:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.entry2=ttk.Entry(self.labelframe1)
        self.entry2.grid(column=1, row=1, padx=4, pady=4)
        self.label3=ttk.Label(self.labelframe1, text="Precio:")
        self.label3.grid(column=0, row=2, padx=4, pady=4)
        self.entry3=ttk.Entry(self.labelframe1)
        self.entry3.grid(column=1, row=2, padx=4, pady=4)
```

```
    def operaciones(self):
        self.boton2=ttk.Button(self.labelframe2, text="Alta")
        self.boton2.grid(column=0, row=0, padx=0, pady=0)
        self.boton3=ttk.Button(self.labelframe2, text="baja")
        self.boton3.grid(column=1, row=0, padx=0, pady=0)
        self.boton4=ttk.Button(self.labelframe2, text="Modificación")
        self.boton4.grid(column=2, row=0, padx=0, pady=0)

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



Te recuerdo que solo hemos realizado la interface gráfica no está la parte lógica de la programación.

Capítulo 245.- tkinter: Layout Manager (administrador de diseño) – pack

Una de las herramientas fundamentales cuando armamos interfaces visuales es la metodología que utilizamos para disponer los controles dentro del formulario. Hasta ahora hemos utlizado el administrador de diseño Grid.

En la librería GUI tkinter disponemos de tres Layout Manager para disponer contoles dentro de una ventana:

1. Grid
2. Pack
3. Place

Solo se puede utilizar uno de estos Layout Manager dentro de un contenedor, recordemos que un contenedor puede ser la ventana propiamente dicha, un Frame o un LabelFrame.

El gestor de diseño más completo y que se adapta en la mayoría de las situaciones es el Grid, pero podemos en muchos casos crear Frame o LabelFrame y definir dentro de estos Layout Manager de tipo Pack o Place.

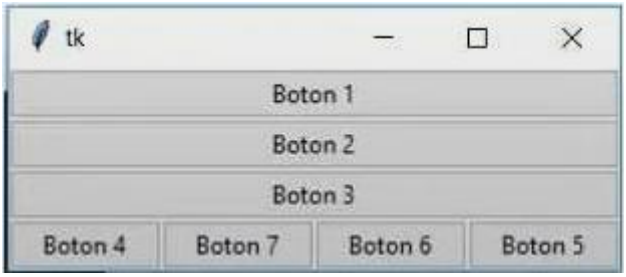
Layout Manager: Pack

Veamos con un ejemplo como se ubican los Widget utilizando Pack.

Problema:

Disponer una serie de botones utilizando el Layout Manager de tipo Pack.

La presentación visual debe ser:

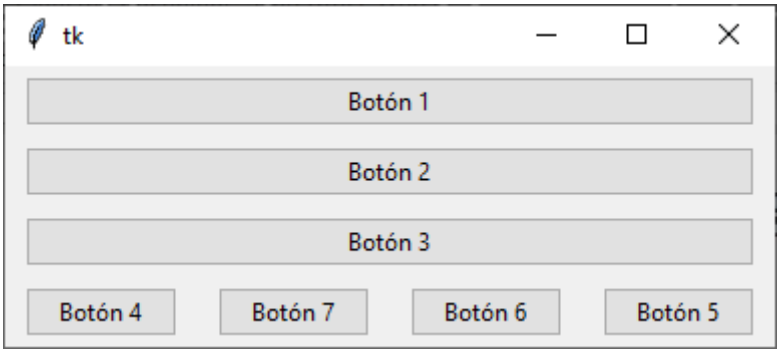


Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.boton1=ttk.Button(self.ventana1, text="Botón 1")
        self.boton1.pack(side=tk.TOP, fill=tk.BOTH, padx=10, pady=5)
        self.boton2=ttk.Button(self.ventana1, text="Botón 2")
        self.boton2.pack(side=tk.TOP, fill=tk.BOTH, padx=10, pady=5)
        self.boton3=ttk.Button(self.ventana1, text="Botón 3")
```

```
        self.boton3.pack(side=tk.TOP, fill=tk.BOTH, padx=10, pady=5)

        self.boton4=ttk.Button(self.ventana1, text="Botón 4")
        self.boton4.pack(side=tk.LEFT, padx=10, pady=5)
        self.boton5=ttk.Button(self.ventana1, text="Botón 5")
        self.boton5.pack(side=tk.RIGHT, padx=10, pady=5)
        self.boton6=ttk.Button(self.ventana1, text="Botón 6")
        self.boton6.pack(side=tk.RIGHT, padx=10, pady=5)
        self.boton7=ttk.Button(self.ventana1, text="Botón 7")
        self.boton7.pack(side=tk.RIGHT, padx=10, pady=5)

        self.ventana1.mainloop()

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



El parámetro side puede recibir alguno de estos cuatro valores:

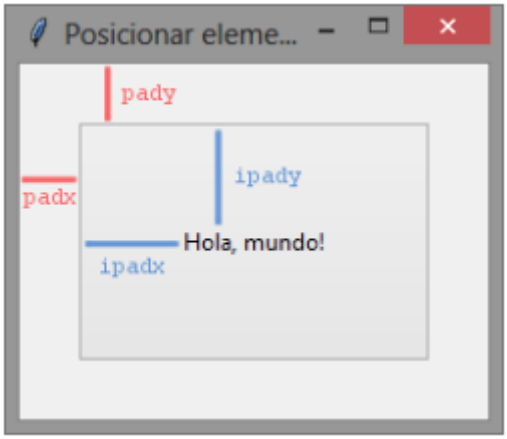
- tk.TOP → Arriba
- tk.LEFT → Izquierda
- tk.RIGHT → Derecha
- tk.BOTTOM → Abajo

Con el método fill=tk.BOTH → Conseguimos que el botón se expanda por ambas partes al modificar el tamaño de la ventana.

Otras propiedades incluyen padx, ipadx, pady y ipady que especifican (en píxeles) los margenes externos e internos de un elemento. Por ejemplo, en el siguiente código habrá un espacio de 30 píxeles entre el botón y la ventana (margen externo), pero un espacio de 50 píxeles entre el borde del botón y el texto del mismo (margen interno).

Ejemplo:

```
def __init__(self):

    self.button = ttk.Button(self, text="Hola, mundo!")
    self.button.pack(padx=30, pady=30, ipadx=50, ipady=50)
```


Posicionar eleme...
pady
ipady
padx
Hola, mundo!
ipadx

Capítulo 246.- tkinter: Layout Manager (administrador de diseño) – grid

1. Grid
2. Pack
3. Place

Layout Manager: Grid

Este tipo de Layout Manager lo hemos estado utilizando en muchos conceptos anteriores, veremos otras posibilidades que nos suministra.

Este tipo de Layout define una tabla con columnas y filas, cada vez que agregamos un Widget indicamos en que columna y fila se debe ubicar:

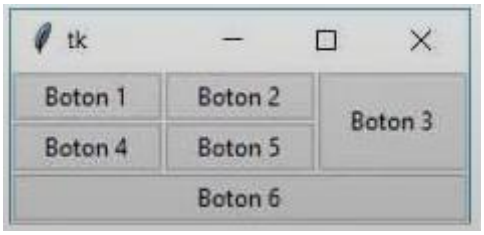
Column=0 row=0	Column=1 row=0	Column=2 row=0
Column=0 row=1	Column=1 row=1	Column=2 row=1
Column=0 row=2	Column=1 row=2	Column=2 row=2

Un tema que no vimos es que podemos expandir celdas de la tabla para que se ocupen más de una columna o más de una fila.

Problema:

Disponer una serie de botones utilizando el Layout Manager de tipo Grid.

La representación visual debe ser:

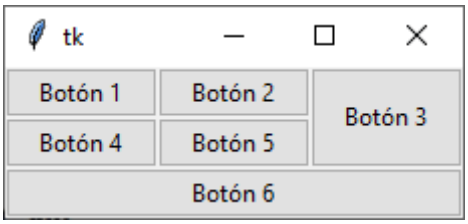


Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.boton1=ttk.Button(self.ventana1, text="Botón 1")
        self.boton1.grid(column=0, row=0)
        self.boton2=ttk.Button(self.ventana1, text="Botón 2")
        self.boton2.grid(column=1, row=0)
        self.boton3=ttk.Button(self.ventana1, text="Botón 3")
        self.boton3.grid(column=2, row=0, rowspan=2, sticky="sn")
        self.boton4=ttk.Button(self.ventana1, text="Botón 4")
        self.boton4.grid(column=0, row=1)
```

```
        self.boton5=ttk.Button(self.ventana1, text="Botón 5")
        self.boton5.grid(column=1, row=1)
        self.boton6=ttk.Button(self.ventana1, text="Botón 6")
        self.boton6.grid(column=0, row=2, columnspan=3, sticky="we")
        self.ventana1.mainloop()

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



Observarás que en el método sticky utilizamos "sn" y en el otro utilizamos el "we", esto hace referencia a los puntos cardenales.

N para el norte, S para el sur, W para el oeste y E para el este.

Capítulo 247.- tkinter: Layout Manager (administrador de diseño) – place

1. Grid
2. Pack
3. Place

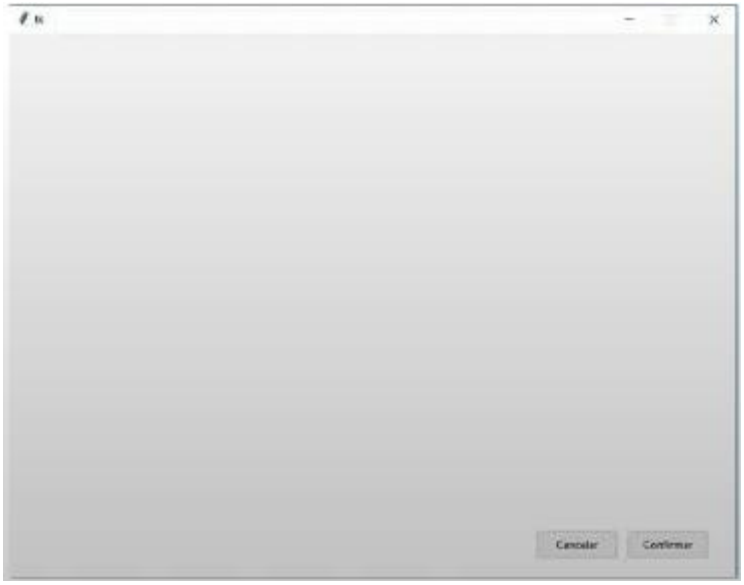
Layout Manager: Place

Este tipo de Layout Manager nos permite disponer un Widget en una posición y con un tamaño con valor absoluto a nivel de píxeles. Hay que tener cuidado en que casos utilizar este tipo de administrador de diseños ya que si agrandamos o reducimos el tamaño de la ventana puede ser que los controles queden fuera de la ventana y el operador no pueda visualizarlos.

Problema:

Disponer dos botones en la parte inferior derecha de la ventana utilizando el Layout Manager de tipo Place. En ancho y alto de la ventana debe ser de 800 por 600 píseles.

La representación visual debe ser:

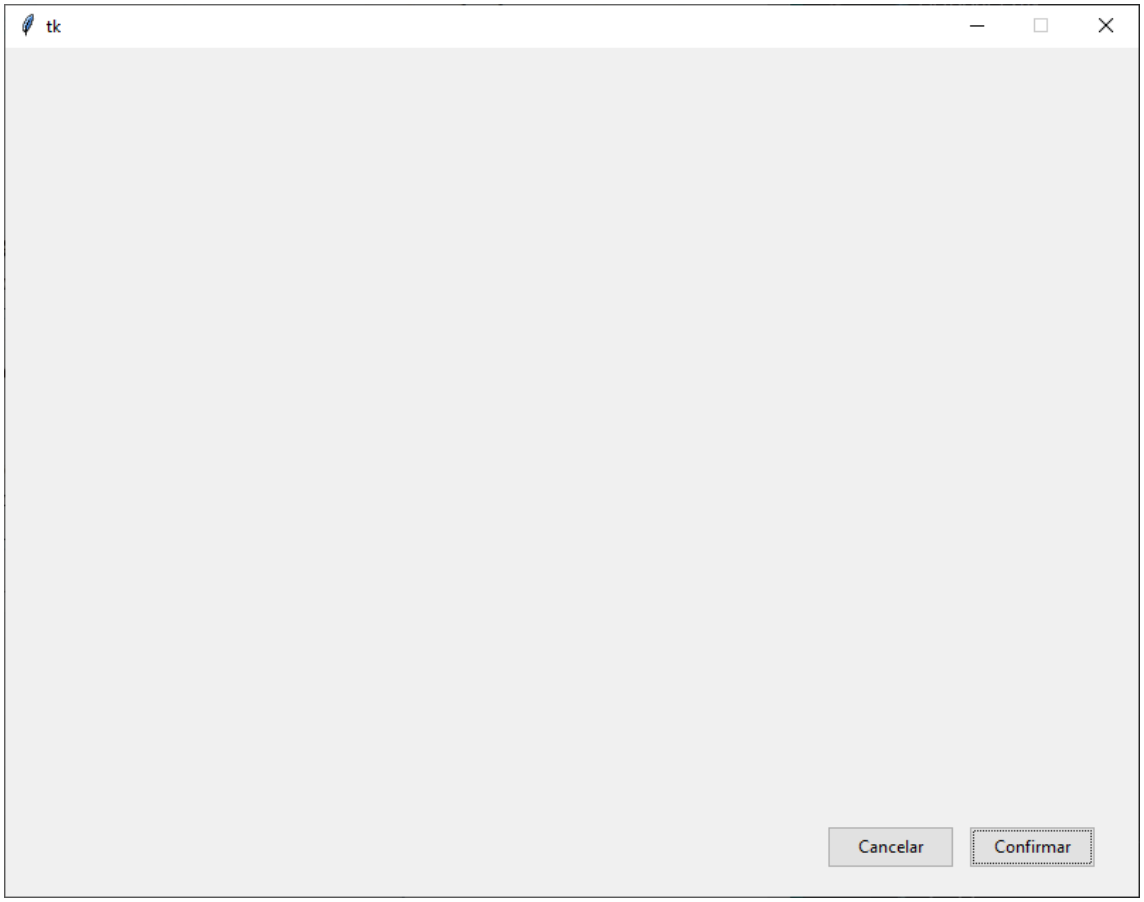


Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.geometry("800x600")
        self.ventana1.resizable(0,0)
        self.boton1=ttk.Button(self.ventana1, text="Confirmar")
        self.boton1.place(x=680, y=550, width=90, height=30)
        self.boton2=ttk.Button(self.ventana1, text="Cancelar")
        self.boton2.place(x=580, y=550, width=90, height=30)
        self.ventana1.mainloop()
```

```
# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



Capítulo 248.- tkinter: messagebox (ventana de mensajes) – 1

La librería tkinter provee un paquete llamado messagebox con una serie de funciones para la apertura de diálogo de información.

Para usar estos diálogos lo primero que debemos hacer es importar el paquete:

from tkinter import messagebox as mb

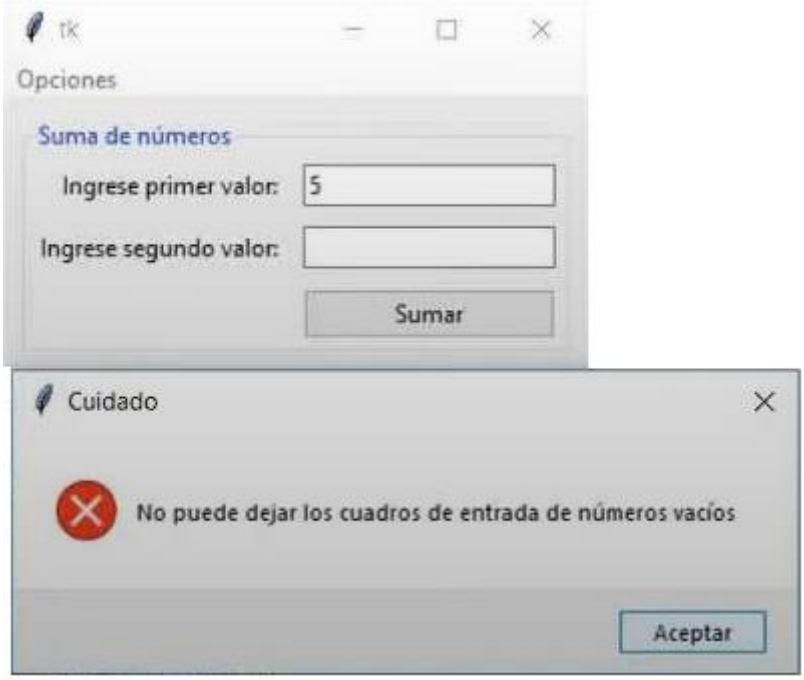
Podemos crear un alias para el nombre del paquete para no tener que escribir messagebox todas las veces.

Problema:

Confeccionar una aplicación que permita ingresar dos valores enteros y al presionar el botón nos muestre la suma en el título de la ventana. Si el operador no ingresa en alguno de los controles Entry datos informar mediante un diálogo el error que se está cometiendo.

Agregar además un menú de opciones que al ser seleccionado nos muestre información dl programa.

La representación visual cuando se presiona el botón "sumar" y no se ingresa alguno de los dos números debe ser:

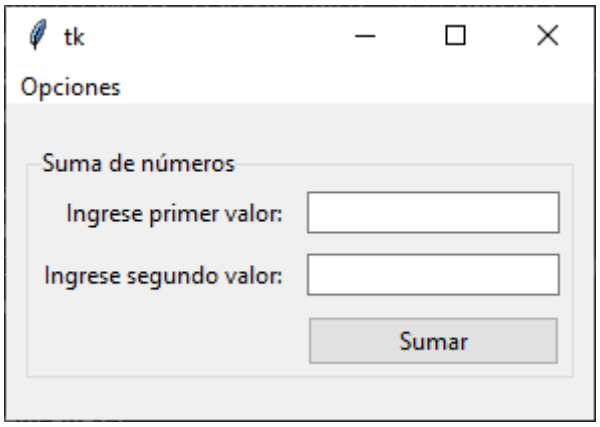


Vamos a programar:

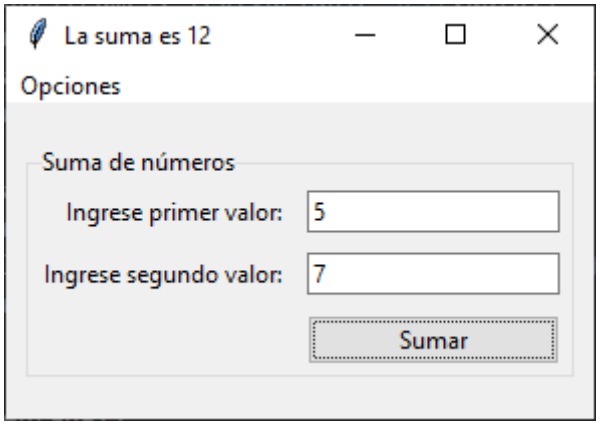
```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mb

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
```

```
        self.labelframe1=ttk.LabelFrame(self.ventana1, text="Suma de
números")
        self.labelframe1.grid(column=0, row=0, padx=10, pady=20)
        self.agregar_componentes()
        self.agregar_menu()
        self.ventana1.mainloop()

    def agregar_componentes(self):
        self.label1=ttk.Label(self.labelframe1, text="Ingrese primer
valor:")
        self.label1.grid(column=0, row=0, padx=5, pady=5, sticky="e")
        self.dato1=tk.StringVar()
        self.entry1=ttk.Entry(self.labelframe1, textvariable=self.dato1)
        self.entry1.grid(column=1, row=0, padx=5, pady=5)
        self.label2=ttk.Label(self.labelframe1, text="Ingrese segundo
valor:")
        self.label2.grid(column=0, row=1, padx=5, pady=5)
        self.dato2=tk.StringVar()
        self.entry2=ttk.Entry(self.labelframe1, textvariable=self.dato2)
        self.entry2.grid(column=1, row=1, padx=5, pady=5)
        self.boton1=ttk.Button(self.labelframe1, text="Sumar",
command=self.sumar)
        self.boton1.grid(column=1, row=2, padx=5, pady=5, sticky="we")

    def agregar_menu(self):
        self.menubar1=tk.Menu(self.ventana1)
        self.ventana1.config(menu=self.menubar1)
        self.opciones1=tk.Menu(self.menubar1, tearoff=0)
        self.opciones1.add_command(label="Acerca de...",
command=self.acerca)
        self.menubar1.add_cascade(label="Opciones", menu=self.opciones1)

    def sumar(self):
        if self.dato1.get()=="" or self.dato2.get()="":
            mb.showerror("Cuidado","No se puede ddejar los cuadros de
entrada de números vacios")
        else:
            suma=int(self.dato1.get()) + int(self.dato2.get())
            self.ventana1.title("La suma es "+str(suma))

    def acerca(self):
        mb.showinfo("Información", "Este programa fue desarrollado para
el aprendizaje de python y tkinter")

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:

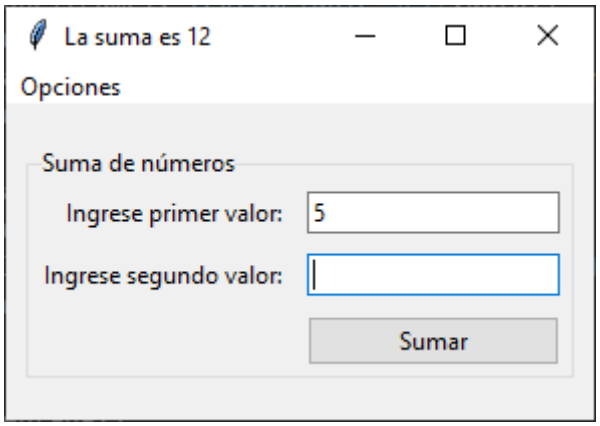


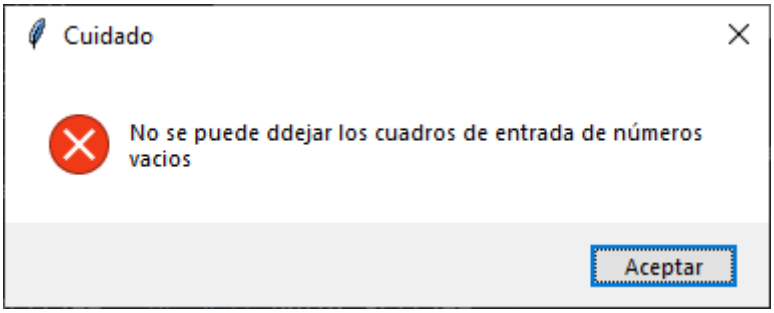
Introducimos dos números:



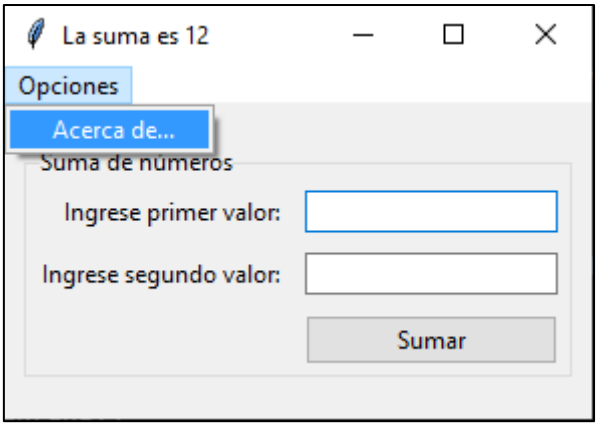
Muestra el resultado en el título de la ventana.

Nos dejamos de introducir un valor:

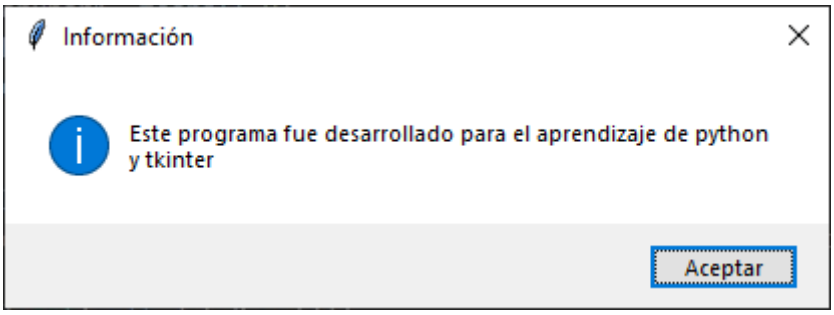




Seleccionamos el menú Opciones y de este Acerca de:



Mostrará la siguiente información:



Capítulo 249.- tkinter: messagebox (ventana de mensajes) – 2

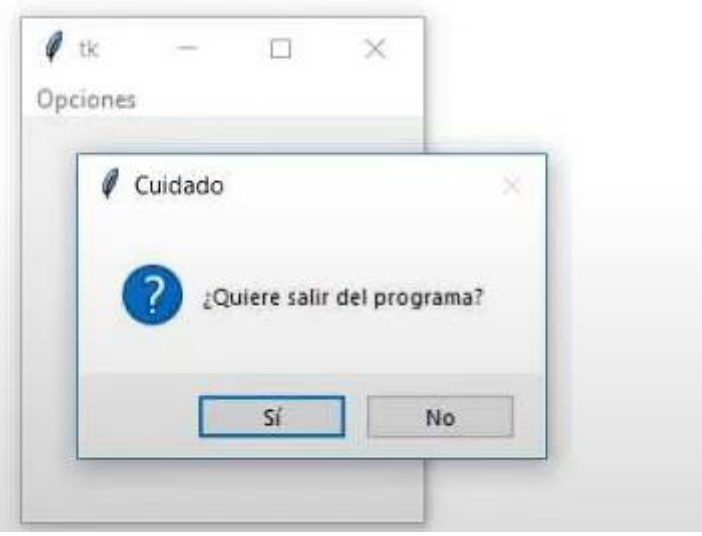
Diálogos para confirmar o rechazar.

El paquete 'messagebox' cuenta con otra función que nos muestra un diálogo con dos botones "Si" o "No", luego desde nuestro programa podemos identificar cual de los dos botones se ha presionado.

Problema:

Confeccionar un programa que tenga solo un menú de opciones que al ser presionado nos muestre un cuadro de mensaje que informe si queremos finalizar la ejecución del programa. Si se presiona "Si" se finaliza el programa en caso contrario no se hace nada.

La representación visual luego de seleccionar la opción del menú es:

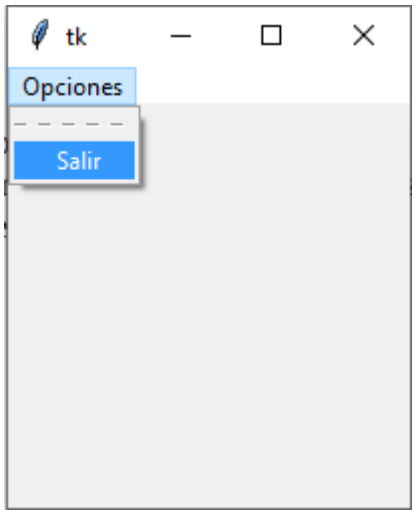


Vamos a programar:

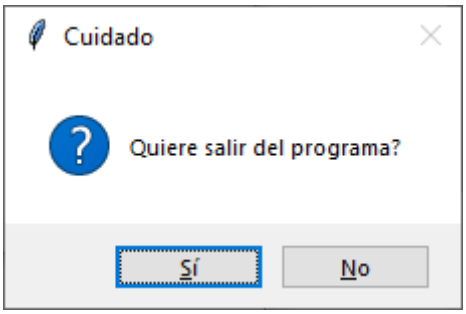
```
import tkinter as tk
from tkinter import messagebox as mb
import sys

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.agregar_menu()
        self.ventana1.mainloop()

    def agregar_menu(self):
        self.menubar1=tk.Menu(self.ventana1)
        self.ventana1.config(menu=self.menubar1)
        self.opciones1=tk.Menu(self.menubar1)
        self.opciones1.add_command(label="Salir", command=self.salir)
        self.menubar1.add_cascade(label="Opciones", menu=self.opciones1)

    def salir(self):
```

```
        respuesta=mb.askyesno("Cuidado", "Quiere salir del programa?")
        if respuesta==True:
            sys.exit(0)

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



Si del menú seleccionamos Salir.



Tendremos la opción de confirmar para salir, de lo contrario no saldremos.

Capítulo 250.- tkinter: Ventanas de diálogo (clase Toplevel)

Vimos en concepto anterior que tkinter nos provee una serie de diálogos para mostrar mensajes de información y error. En muchas situaciones podemos necesitar crear otras ventanas con más funcionalidades que los simples ventanas de mensajes.

Para crear diálogos en tkinter debemos crear un objeto de la clase TopLevel y pasar como parámetros la referencia de la ventana principal.

Un diálogo se asemeja mucho a lo que es la ventana principal, podemos disponer dentro de la misma objetos de la clase Label, Button, Entry etc.

Lo más común es que un diálogo tenga por objetivo la entrada de datos para luego ser utilizados en la ventana principal.

Problema:

Confeccionar una aplicación que muestre un diálogo cuando se seleccione una opción de un menú. El diálogo debe solicitar el ingreso de dos enteros que se utilizarán en la ventana principal para redimensionarla.

La representación visual de la ventana principal es muy sencilla:

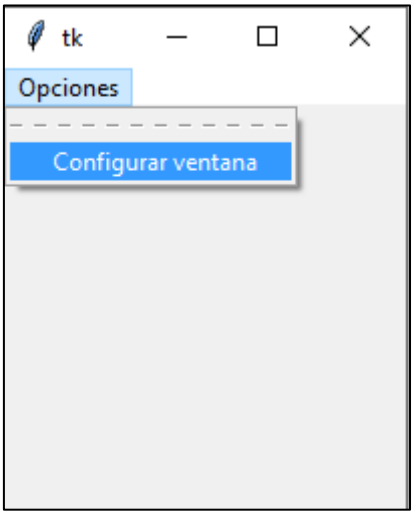
Cuando se abre el diálogo tenemos la siguiente imagen:

Vamos a programar:

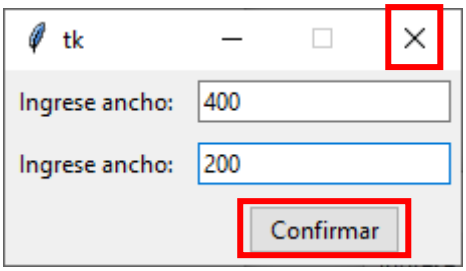
```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        # Llamaremos al método agregar_menu()
        self.agregar_menu()
        self.ventana1.mainloop()

    def agregar_menu(self):
        self.menubar1=tk.Menu(self.ventana1)
        self.ventana1.config(menu=self.menubar1)
        self.opciones1=tk.Menu(self.menubar1)
        self.opciones1.add_command(label="Configurar ventana",
command=self.configurar)
        self.menubar1.add_cascade(label="Opciones", menu=self.opciones1)

    def configurar(self):
        # Creamos un objeto de la clase DialogoTamaño
        dialogo1=DialogoTamaño(self.ventana1)
        # Desde dialogo1 llamamos al método mostar()
        # Nos retorna una tupla con dos valores
        s=dialogo1.mostrar()
        # Modifica las dimensiones de la ventana según los valores
        self.ventana1.geometry(s[0]+"x"+s[1])

class DialogoTamaño:
    # tendrá 2 Label, 2 Entry y 1 Button
    # Le pasaremso como argumento la ventana1
    def __init__(self, ventanaprincipal):
        # Craamos un objeto de tipo Toplevel
        self.dialogo=tk.Toplevel(ventanaprincipal)
        self.label1=ttk.Label(self.dialogo, text="Ingrese ancho:")
        self.label1.grid(column=0, row=0, padx=5, pady=5)
        self.dato1=tk.StringVar()
        self.entry1=ttk.Entry(self.dialogo, textvariable=self.dato1)
        self.entry1.grid(column=1, row=0, padx=5, pady=5)
        # El cursor se encontrará dentro de entry1
        self.entry1.focus()
        self.label2=ttk.Label(self.dialogo, text="Ingrese ancho:")
        self.label2.grid(column=0, row=1, padx=5, pady=5)
        self.dato2=tk.StringVar()
        self.entry2=ttk.Entry(self.dialogo, textvariable=self.dato2)
        self.entry2.grid(column=1, row=1, padx=5, pady=5)
```

```
        self.boton1=ttk.Button(self.dialogo, text="Confirmar",
command=self.confirmar)
        self.boton1.grid(column=1, row=2, padx=5, pady=5)
        # Llamamos al método potocol
        # "WM_DELETE_WINDOW" cuando presionemos le x de la ventana
        # también llamará el método confirmar.
        self.dialogo.protocol("WM_DELETE_WINDOW", self.confirmar)
        # Que no se pueda modificar la ventana de dialogo.
        self.dialogo.resizable(0,0)
        # El método grab_set() hace que esta ventana quede activa
        # y no se pueda volver a la ventana principal hasta que
        # cerremos la ventana de diálgo.
        self.dialogo.grab_set()

    def mostrar(self):
        # Hasta que no finalice el diálogo no ejecutará el return.
        self.dialogo.wait_window()
        # Retorna una tupla con los valores de ancho y alto.
        return (self.dato1.get(), self.dato2.get())

    def confirmar(self):
        # Cuando le damos al botón o a la x de cerrar.
        # la ventana de diálog se desctruye de este modo
        # el método mostrar retorna los valores.
        self.dialogo.destroy()

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



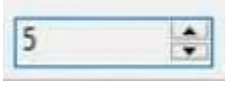
Le damos a configurar ventana:



Para confirmar las dimensiones de la ventana lo podemos hacer por mediación del botón confirmar o por la x de la ventana.

Capítulo 251.- ttk: Control Spinbox

Este control visual se muy utilizado para seleccionar un valor de una lista. Se dispone de dos botones para subir o bajar en la lista de valores posibles:



Se pueden mostrar listas de valores de tipo string, entero, float, etc.

Problema:

En una aduana hay una máquina que sortea las personas cuyo equipaje serán revisados.

La persona selecciona la cantidad de bultos (hacer dicha selección mediante un Spinbox).

Luego se presiona el botón sortear y aparece al lado de este botón un Label de color rojo o verde (En caso de ser rojo se revisa el equipaje, en caso de verde, no se revisa)

Para el sorteo generar un valor aleatorio entre 1 y 3. Si se genera un 1 se revisa, si se genera un 2 o un 3 no se revisa, mostrar un mensaje de error si el Spinbox tiene un cero.

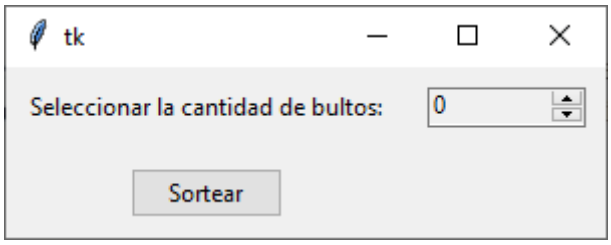
La interfaz visual debe ser similar a esta:

Vamos a programar:

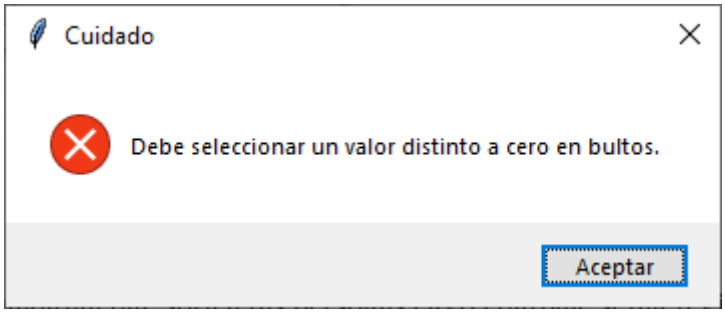
```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mb
import random

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.label1=ttk.Label(self.ventana1, text="Seleccionar la
cantidad de bultos:")
        self.label1.grid(column=0, row=0, padx=10, pady=10)
        self.spinbox1=ttk.Spinbox(self.ventana1, from_=0, to=3, width=10,
state="readonly")
        self.spinbox1.set(0)
        self.spinbox1.grid(column=1, row=0, padx=10, pady=10)
        self.boton1=ttk.Button(self.ventana1, text="Sortear",
command=self.sortear)
        self.boton1.grid(column=0, row=1, padx=10, pady=10)
        self.label2=ttk.Label(self.ventana1, text="")
```

```
        self.label2.grid(column=1, row=1, padx=10, pady=10)
        self.ventana1.mainloop()

    def sortear(self):
        if int(self.spinbox1.get())==0:
            mb.showerror("Cuidado", "Debe seleccionar un valor distinto a
cero en bultos.")
        else:
            valor=random.randint(1,3)
            if valor==1:
                self.label2.configure(text="Se deben revisar")
                self.label2.configure(background="red")
            else:
                self.label2.configure(text="No revisar")
                self.label2.configure(background="green")

# Bloque principal
aplicacion1=Aplicacion()
```

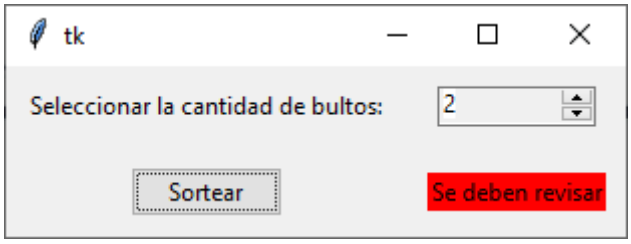
Este será el resultado:



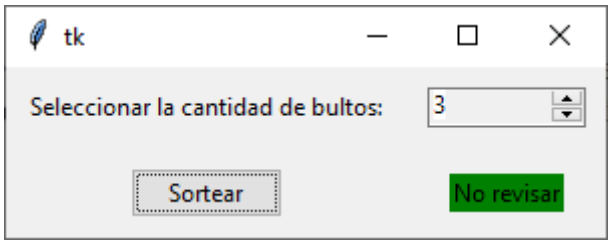
Si seleccionamos el botón Sortear con 0 bultos este es el mensaje de error:



Vamos a poner 2 bultos:



Repetimos con tres bultos:



Los mensajes de 'Se debe revistar' y 'No revisar' son totalmente aleatorios.

Capítulo 252.- tkinter: ScrolledText (editor multilínea)

Hemos visto anteriormente el control Entry que nos permite ingresar una cadena de caracteres por teclado. El Widget ScrolledText es similar al Widget Entry con la salvedad que nos permite ingresar múltiples líneas.

Cuenta con métodos sofisticados para extraer trozos del texto ingresado en el objeto.

Para poder utilizar esta nuevo objeto debemos importar el módulo scrolledtext del paquete tkinter (podemos crear un alias para no tener que escribir scrolldextext):

from tkinter import scrolledtext as st

Funcionamiento

Para extraer cadenas de caracteres de un control ScrolledText debemos llamar al método get y pasar dos String. Por ejemplo si queremos extraer todos los caracteres de la primera fila hasta los 10 primeros caracteres de la tercera fila deberemos codificar:

datos=self.scrolledtext1.get("1.0", "3.10) # Fila 1 Columna0 hasta Fila 3 Columna 10

En nuestro problema estos cuatro números los estamos cargando por teclado en los controles Entry.

Para borrar todo el contenido de un ScrolldedText debemos llamar al método delete y pasa estos dos parámetros.

self.scrolledtext2.delete("1.0", tk.END)

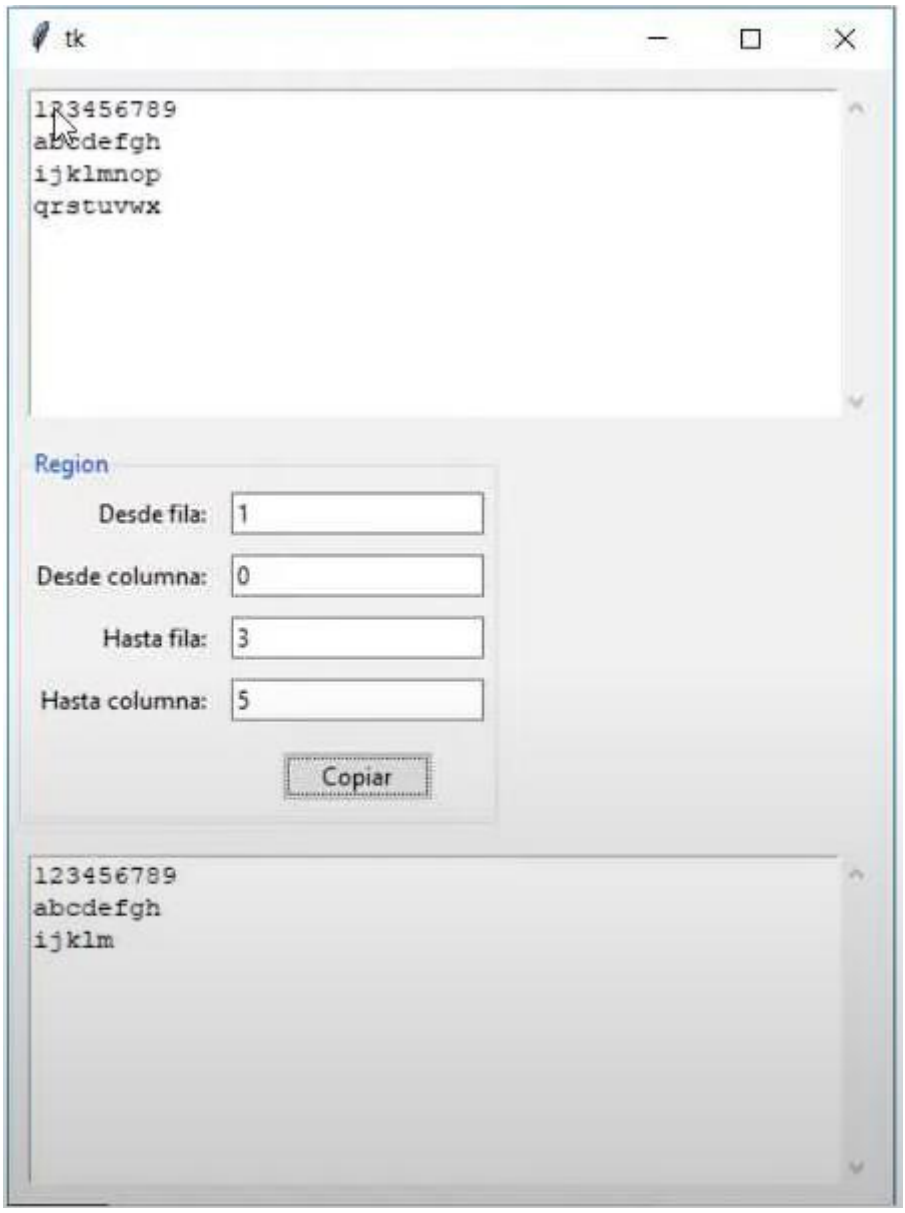
Para inserter en cualquier parte de un ScrolledText empleamos el método insert:

self.scrolledtext2.insert("1.0", datos)

Problema:

Confeccionar un programa que contenga dos controles de tipo ScrolledText. En el primero ingresamos por teclado cualquier texto. Mediante 4 controles de tipo Entry indicar desde que fila y columna hasta que fila y columna extraer caracteres del primer ScrolledText y copiarlos al segundo ScrolledText cuando se presione el botón.

La interfaz visual debe ser similar a esta:



Vamos a programar:

```
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext as st

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.scrolledtext1=st.ScrolledText(self.ventana1, width=50,
heigh=10)
        self.scrolledtext1.grid(column=0, row=0, padx=10, pady=10)
        self.framecopia()
        self.scrolledtext2=st.ScrolledText(self.ventana1, width=50,
height=10)
        self.scrolledtext2.grid(column=0, row=2, padx=10, pady=10)
```

```
        self.ventana1.mainloop()

    def framecopia(self):
        self.labelframe1=ttk.LabelFrame(self.ventana1, text="Región")
        self.labelframe1.grid(column=0, row=1, padx=5, pady=5,
sticky="w")
        self.label1=ttk.Label(self.labelframe1, text="Desde fila:")
        self.label1.grid(column=0, row=0, padx=5, pady=5, sticky="e")
        self.dato1=tk.StringVar()
        self.entry1=ttk.Entry(self.labelframe1, textvariable=self.dato1)
        self.entry1.grid(column=1, row=0, padx=5, pady=5, sticky="e")

        self.label2=ttk.Label(self.labelframe1, text="Desde columna:")
        self.label2.grid(column=0, row=1, padx=5, pady=5, sticky="e")
        self.dato2=tk.StringVar()
        self.entry2=ttk.Entry(self.labelframe1, textvariable=self.dato2)
        self.entry2.grid(column=1, row=1, padx=5, pady=5, sticky="e")

        self.label3=ttk.Label(self.labelframe1, text="Hasta fila:")
        self.label3.grid(column=0, row=2, padx=5, pady=5, sticky="e")
        self.dato3=tk.StringVar()
        self.entry3=ttk.Entry(self.labelframe1, textvariable=self.dato3)
        self.entry3.grid(column=1, row=2, padx=5, pady=5, sticky="e")

        self.label4=ttk.Label(self.labelframe1, text="Hasta columna:")
        self.label4.grid(column=0, row=3, padx=5, pady=5, sticky="e")
        self.dato4=tk.StringVar()
        self.entry4=ttk.Entry(self.labelframe1, textvariable=self.dato4)
        self.entry4.grid(column=1, row=3, padx=5, pady=5, sticky="e")

        self.boton1=ttk.Button(self.labelframe1, text="Copiar",
command=self.copiar)
        self.boton1.grid(column=1, row=4, padx=10, pady=10)

    def copiar(self):
        iniciofila=self.dato1.get()
        iniciocolumna=self.dato2.get()
        finfila=self.dato3.get()
        fincolumna=self.dato4.get()
        datos=self.scrolledtext1.get(iniciofila+"."+iniciocolumna,finfila
+"."+fincolumna)
        self.scrolledtext2.delete("1.0", tk.END)
        self.scrolledtext2.insert("1.0", datos)

# Bloque principal
aplicacion1=Aplicacion()
```

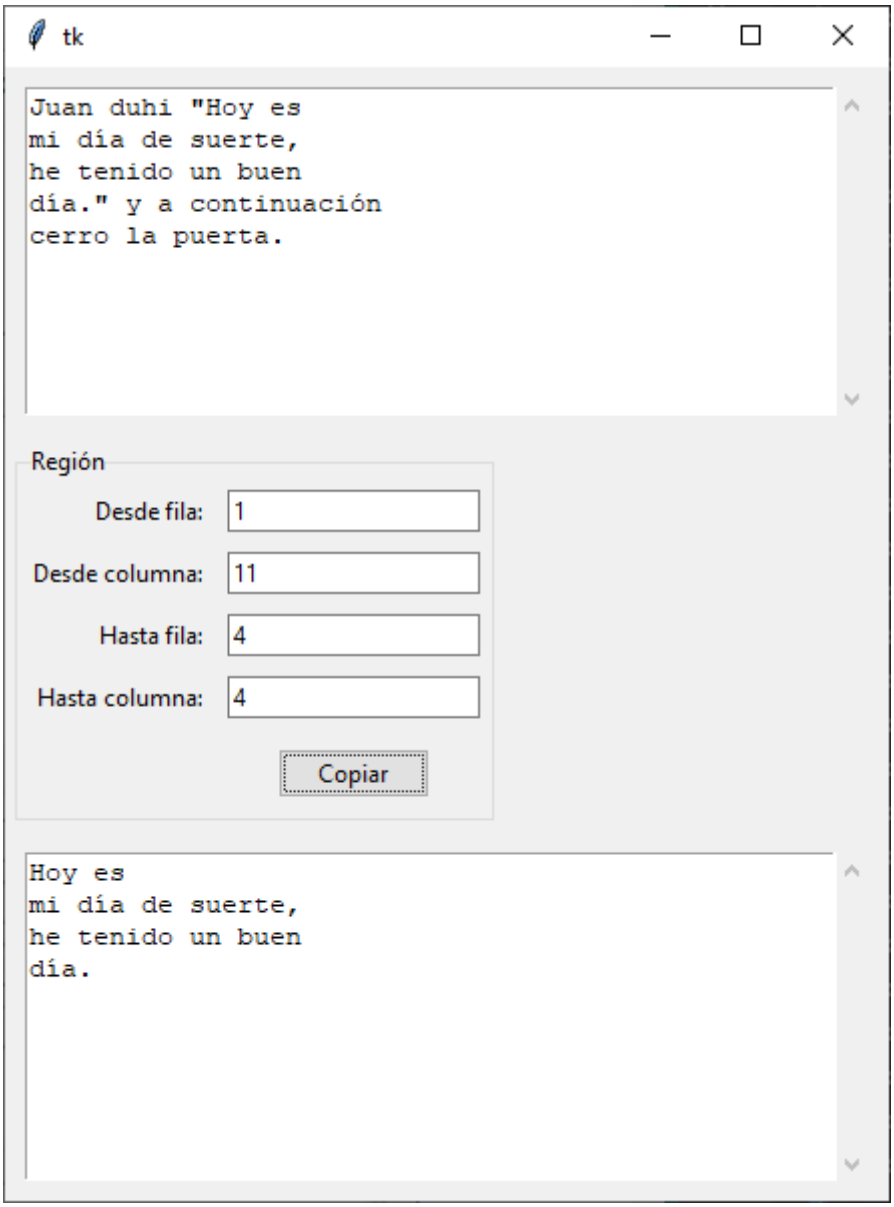
Este será el resultado:

Del siguiente texto:

Juan dijo "Hoy es
mi día de suerte,
he tenido un buen
día." y a continuación
cerró la puerta.

Queremos recuperar el texto que se encuentra entre comillas dobles.

Hoy es
mi día de suerte,
he tenido un buen
día.



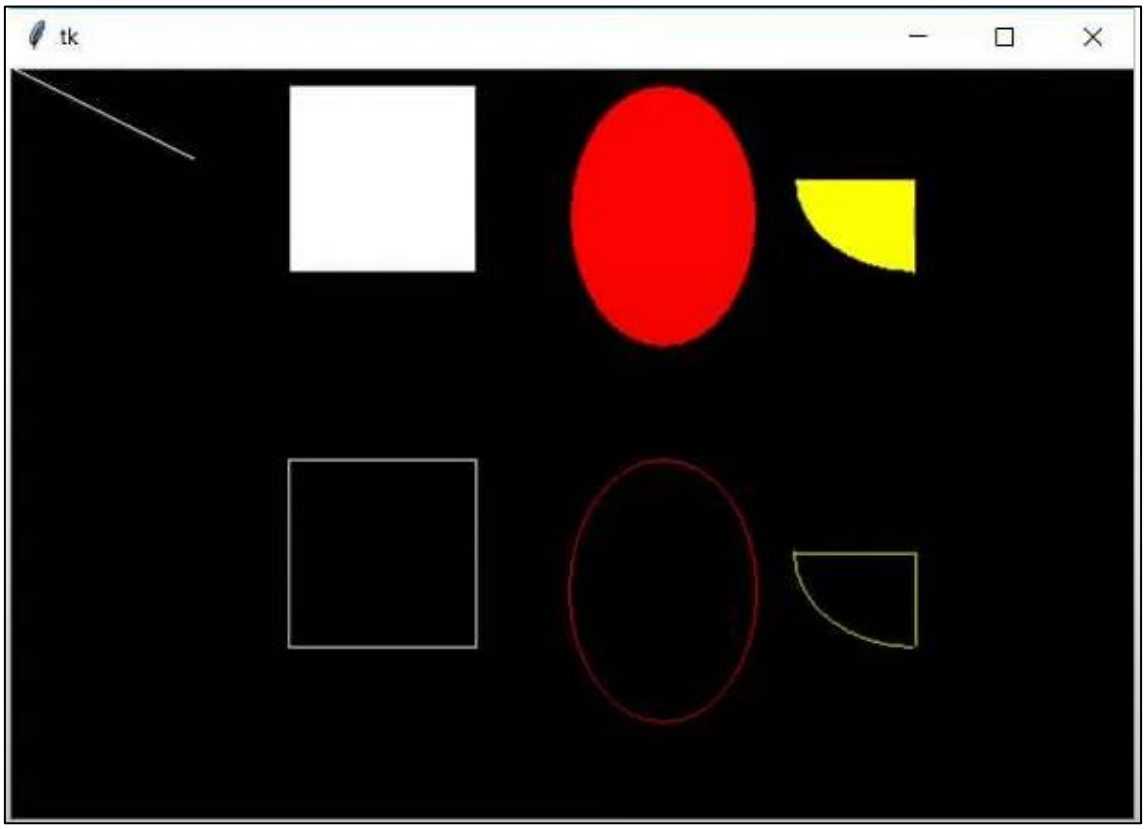
Capítulo 253.- tkinter: Control Canvas – 1

El control Canvas nos permite acceder a una serie de primitivas gráficas: líneas, rectángulos, óvalos, arcos, etc. para dibujar dentro de la misma.

Problema:

Confeccionar un programa que cree un objeto de la clase Canvas y muestre la funcionalidad de las principales primitivas gráficas.

La interfaz visual debe ser similar a esta:



Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400,
background="black")
        self.canvas1.grid(column=0, row=0)
        self.canvas1.create_line(0,0,100,50, fill="white")
        self.canvas1.create_rectangle(150,10,250,110, fill="white")
        self.canvas1.create_oval(300,10,400,150, fill="red")
        self.canvas1.create_arc(420,10,550,110, fill="yellow", start=180,
extent=90)
        self.canvas1.create_rectangle(150,210,250,310, outline="white")
```

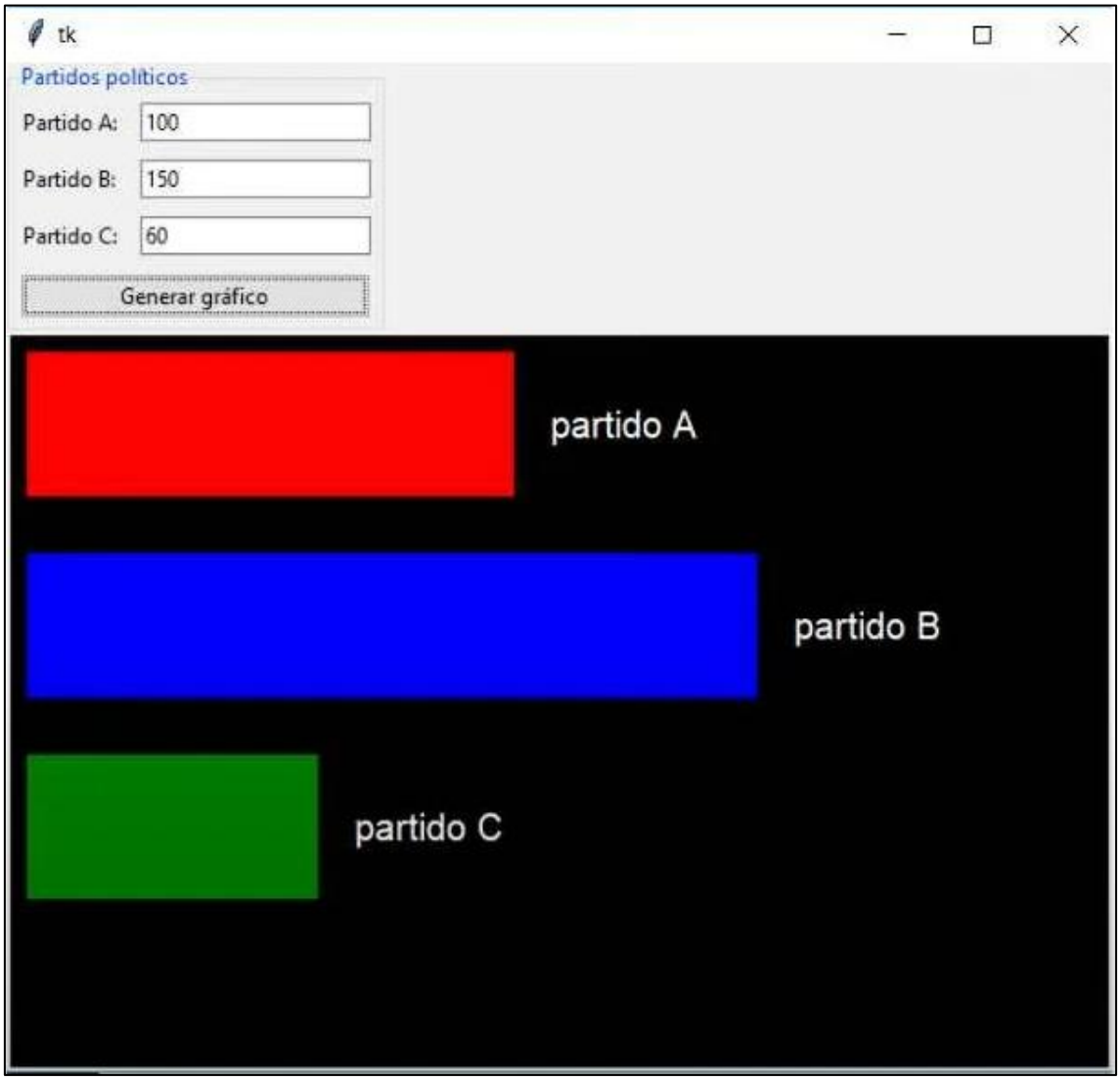
```
        self.canvas1.create_oval(300,210,400,350, outline="red")
        self.canvas1.create_arc(420,210,550,310, outline="yellow",
start=180, extent=90)
        self.ventana1.mainloop()

# Bloque principal
aplicación1=Aplicación()
```

Capítulo 254.- tkinter: Control Canvas – 2

Problema:

Crear una aplicación que solicite el ingreso de tres valores por teclado que representan las cantidades de votos obtenidos por tres partidos políticos. Luego mostrar un gráfico de barras horizontales.

La interfaz visual debe ser similar a esta:



Vamos a programar:

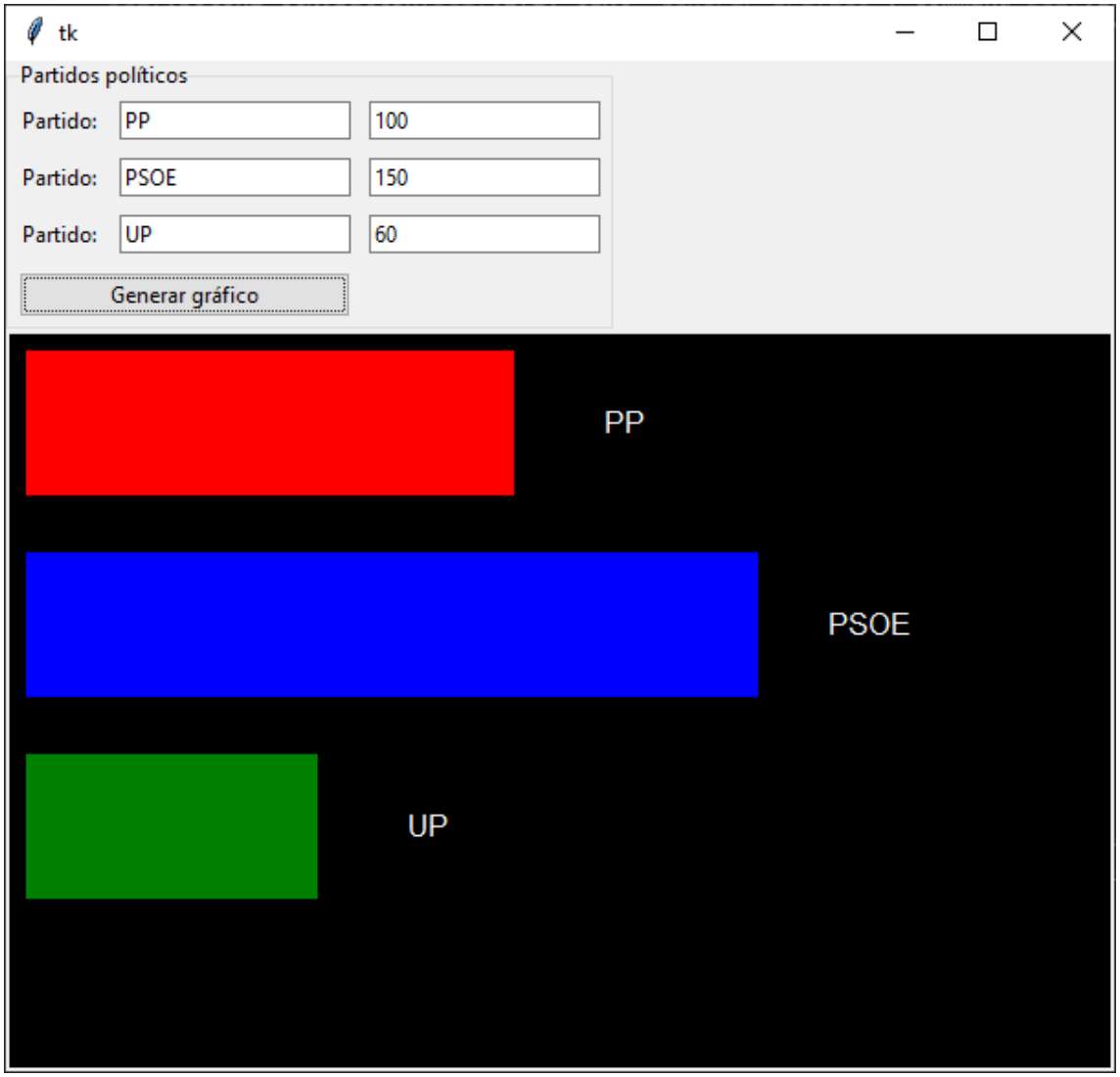
```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.entradadatos()
```

```
        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400,
background="black")
        self.canvas1.grid(column=0, row=1)
        self.ventana1.mainloop()

    def entradadatos(self):
        self.lf1=ttk.LabelFrame(self.ventana1, text="Partidos políticos")
        self.lf1.grid(column=0, row=0, sticky="w")
        self.label1=ttk.Label(self.lf1, text="Partido A:")
        self.label1.grid(column=0, row=0, padx=5, pady=5)
        self.dato1=tk.StringVar()
        self.entry1=ttk.Entry(self.lf1,textvariable=self.dato1)
        self.entry1.grid(column=1, row=0, padx=5, pady=5)

        self.label2=ttk.Label(self.lf1, text="Partido B:")
        self.label2.grid(column=0, row=1, padx=5, pady=5)
        self.dato2=tk.StringVar()
        self.entry2=ttk.Entry(self.lf1,textvariable=self.dato2)
        self.entry2.grid(column=1, row=1, padx=5, pady=5)

        self.label3=ttk.Label(self.lf1, text="Partido C:")
        self.label3.grid(column=0, row=2, padx=5, pady=5)
        self.dato3=tk.StringVar()
        self.entry3=ttk.Entry(self.lf1,textvariable=self.dato3)
        self.entry3.grid(column=1, row=2, padx=5, pady=5)

        self.boton1=ttk.Button(self.lf1, text="Generar gráfico",
command=self.grafico_barra)
        self.boton1.grid(column=0, row=3, columnspan=2, padx=5, pady=5,
sticky="we")
        self.entry1.focus()

    def grafico_barra(self):
        self.canvas1.delete(tk.ALL)
        valor1=int(self.dato1.get())
        valor2=int(self.dato2.get())
        valor3=int(self.dato3.get())
        if valor1>valor2 and valor1>valor3:
            mayor=valor1
        else:
            if valor2>valor3:
                mayor=valor2
            else:
                mayor=valor3
        largo1=valor1/mayor*400
        largo2=valor2/mayor*400
        largo3=valor3/mayor*400
        self.canvas1.create_rectangle(10,10,10+largo1,90, fill="red")
```

```
        self.canvas1.create_rectangle(10, 120, 10+largo2, 200,
fill="blue")
        self.canvas1.create_rectangle(10,230, 10+largo3, 310,
fill="green")
        self.canvas1.create_text(largo1+70,50, text="Partido A",
fill="white", font="Arial")
        self.canvas1.create_text(largo2+70,160, text="Partido B",
fill="white", font="Arial")
        self.canvas1.create_text(largo3+70, 270, text="Partido C",
fill="white", font="Arial")

# Bloque principal
aplicacion1=Aplicacion()
```

Ahora te hago la siguiente propuesta:

Modificando el código anterior quiero este resultado.

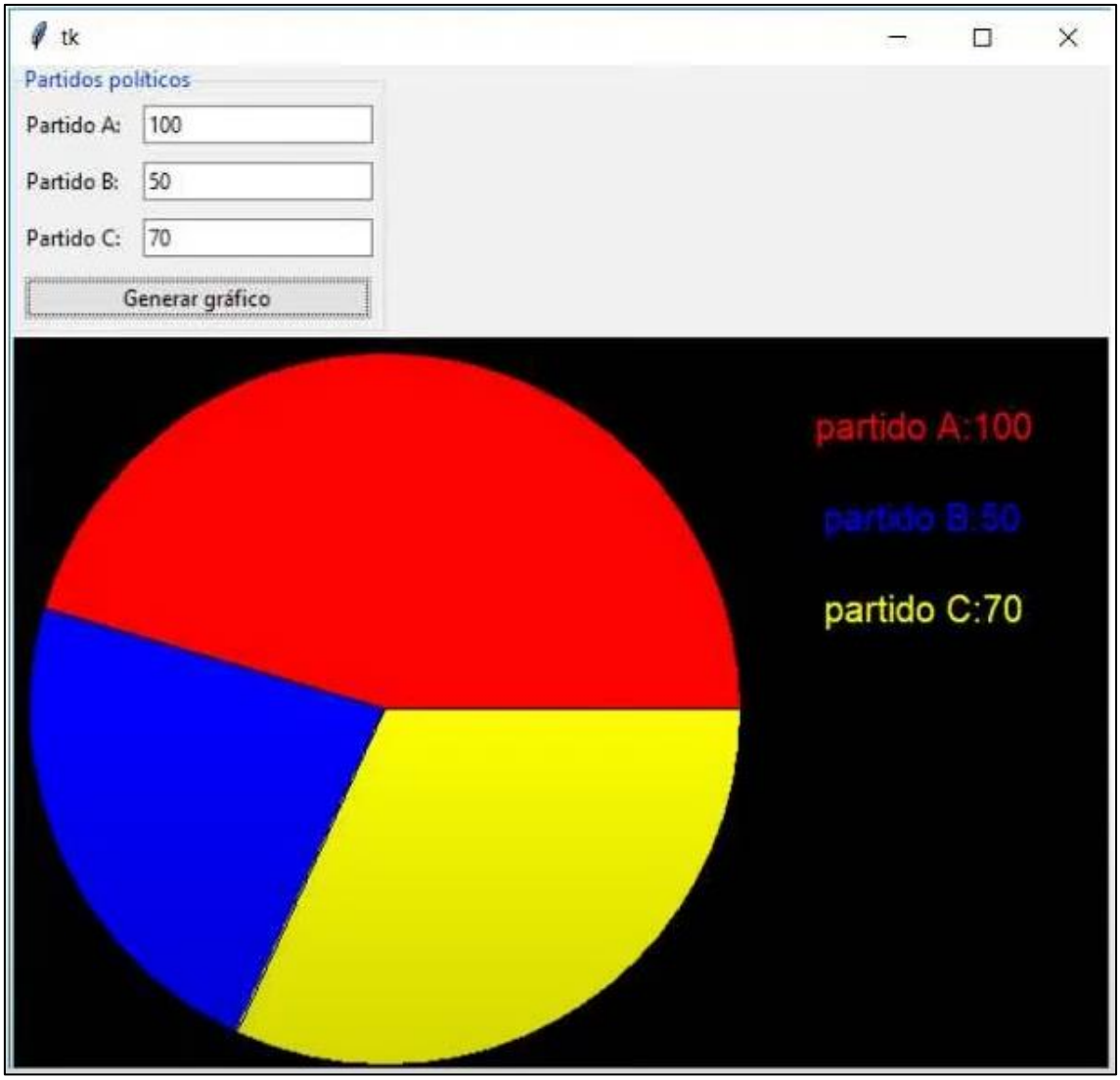


Capítulo 255.- tkinter: Control Canvas – 3

Problema:

Crear una aplicación que solicite el ingreso de tres valores por teclado que representan las cantidades de botos obtenidos por tres particos políticos. Luego mostrar un gráfico de tartas:

La interfaz visual debe ser similar a esta:



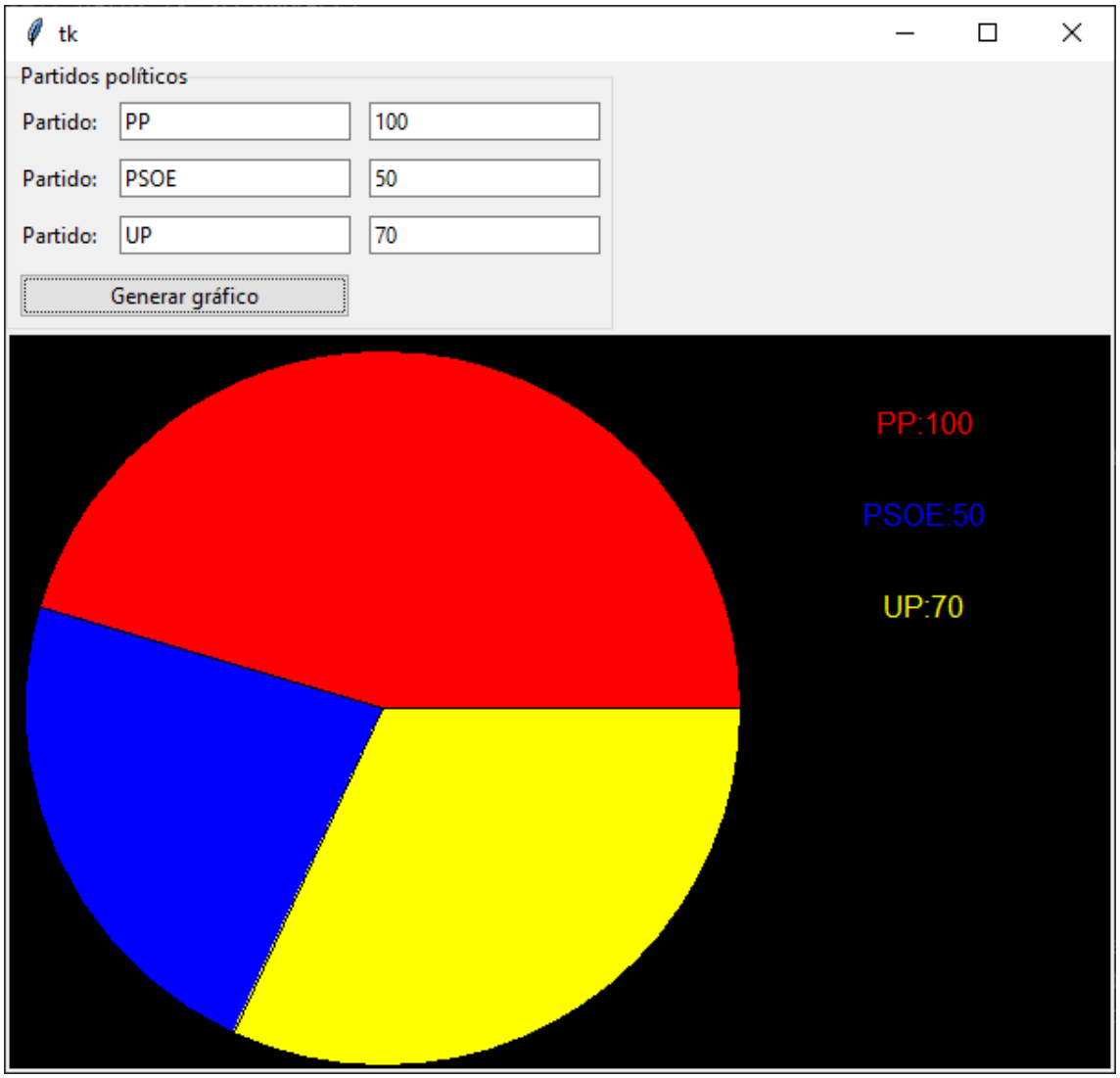
Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.entradadatos()
        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400,
background="black")
```

```
        self.canvas1.grid(column=0, row=1)
        self.ventana1.mainloop()

    def entradadatos(self):
        self.lf1=ttk.LabelFrame(self.ventana1, text="Partidos políticos")
        self.lf1.grid(column=0, row=0, sticky="w")
        self.label1=ttk.Label(self.lf1, text="Partido A:")
        self.label1.grid(column=0, row=0, padx=5, pady=5)
        self.dato1=tk.StringVar()
        self.entry1=ttk.Entry(self.lf1,textvariable=self.dato1)
        self.entry1.grid(column=1, row=0, padx=5, pady=5)

        self.label2=ttk.Label(self.lf1, text="Partido B:")
        self.label2.grid(column=0, row=1, padx=5, pady=5)
        self.dato2=tk.StringVar()
        self.entry2=ttk.Entry(self.lf1,textvariable=self.dato2)
        self.entry2.grid(column=1, row=1, padx=5, pady=5)

        self.label3=ttk.Label(self.lf1, text="Partido C:")
        self.label3.grid(column=0, row=2, padx=5, pady=5)
        self.dato3=tk.StringVar()
        self.entry3=ttk.Entry(self.lf1,textvariable=self.dato3)
        self.entry3.grid(column=1, row=2, padx=5, pady=5)

        self.boton1=ttk.Button(self.lf1, text="Generar gráfico",
command=self.grafico_tarta)
        self.boton1.grid(column=0, row=3, columnspan=2, padx=5, pady=5,
sticky="we")
        self.entry1.focus()

    def grafico_tarta(self):
        self.canvas1.delete(tk.ALL)
        valor1=int(self.dato1.get())
        valor2=int(self.dato2.get())
        valor3=int(self.dato3.get())
        suma=valor1+valor2+valor3
        grados1=valor1/suma*360
        grados2=valor2/suma*360
        grados3=valor3/suma*360
        self.canvas1.create_arc(10,10,400,400, fill="red", start=0,
extent=grados1)
        self.canvas1.create_arc(10,10,400,400, fill="blue",
start=grados1, extent=grados2)
        self.canvas1.create_arc(10,10, 400, 400, fill="yellow",
start=grados1+grados2, extent=grados3)
        self.canvas1.create_text(500,50, text="Partido A:"+str(valor1),
fill="red", font="Arial")
        self.canvas1.create_text(500,100, text="Partido B:"+str(valor2),
fill="blue", font="Arial")
```

```
        self.canvas1.create_text(500,150, text="Partido C:"+str(valor3),
fill="yellow", font="Arial")


# Bloque principal
aplicacion1=Aplicacion()
```

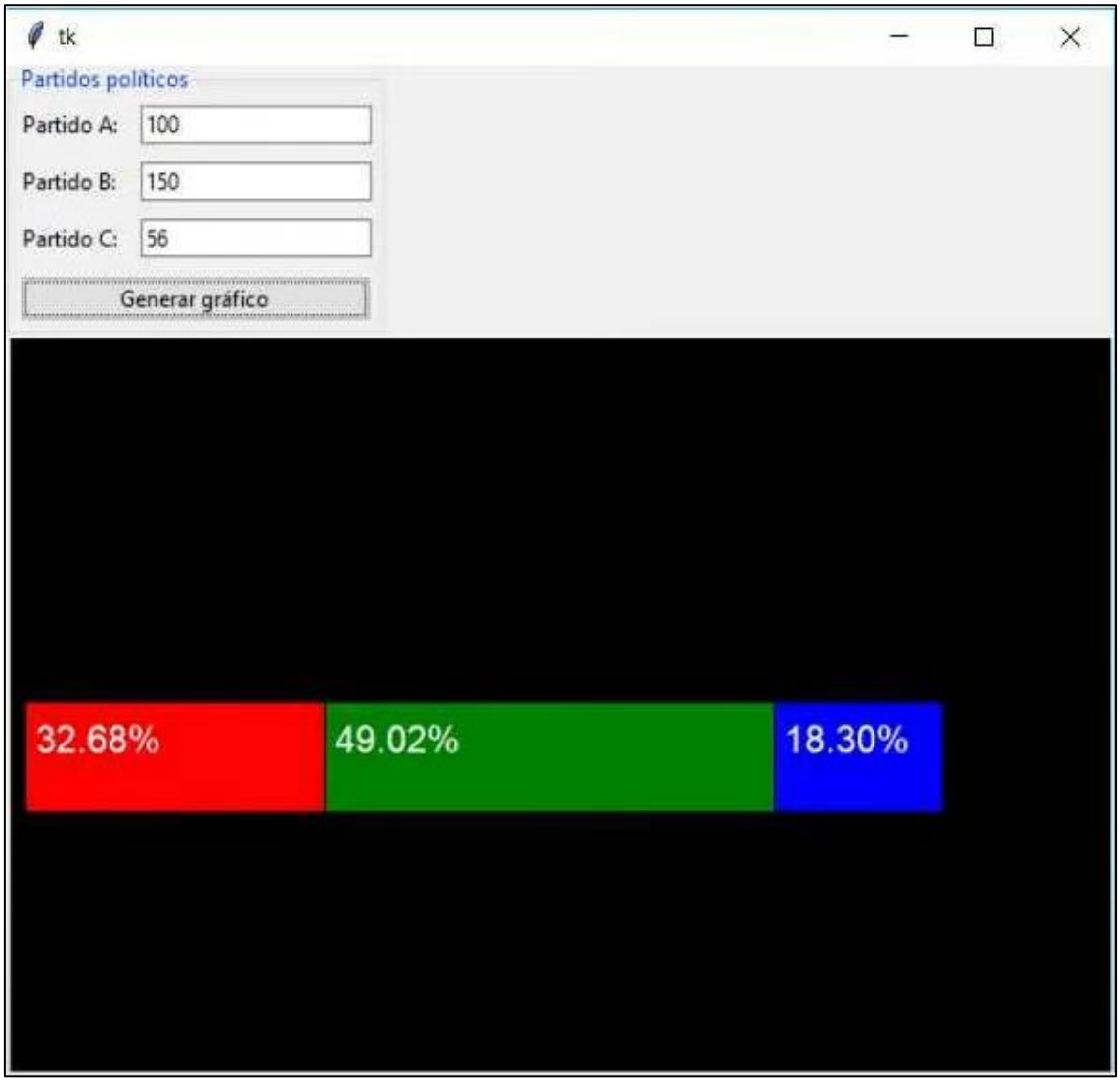
Igual que en capítulo anterior rectifica el programa para obtener el siguiente resultado:



Capítulo 256.- tkinter: Control Canvas – 4

Problema propuesto:

Implementar un gráfico estadístico de tipo "Barra Porcentual":



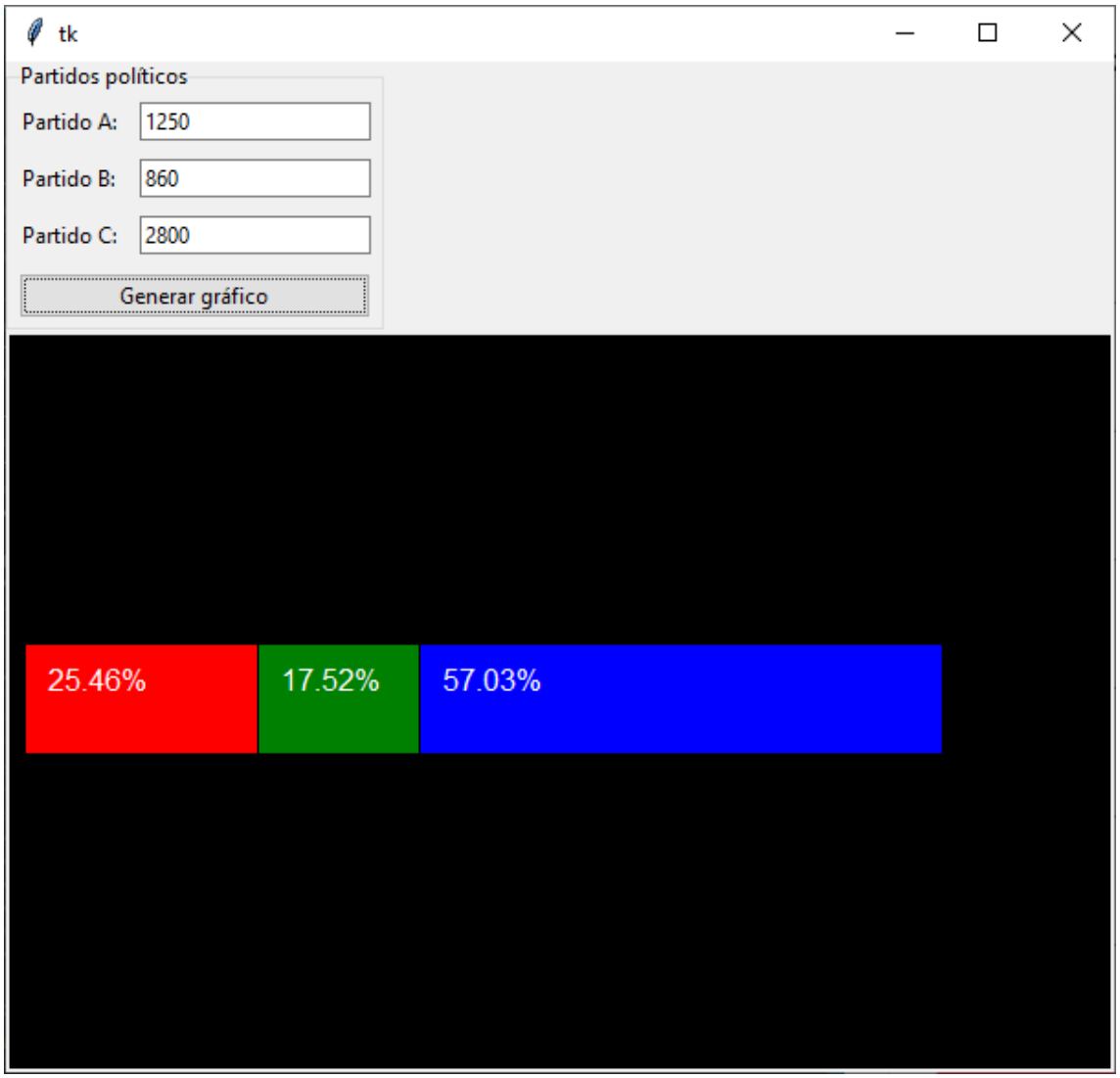
Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.entradadatos()
        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400,
background="black")
        self.canvas1.grid(column=0, row=1)
        self.ventana1.mainloop()
```

```
    def entradadatos(self):
        self.lf1=ttk.LabelFrame(self.ventana1, text="Partidos políticos")
        self.lf1.grid(column=0, row=0, sticky="w")
        self.label1=ttk.Label(self.lf1, text="Partido A:")
        self.label1.grid(column=0, row=0, padx=5, pady=5)
        self.dato1=tk.StringVar()
        self.entry1=ttk.Entry(self.lf1,textvariable=self.dato1)
        self.entry1.grid(column=1, row=0, padx=5, pady=5)

        self.label2=ttk.Label(self.lf1, text="Partido B:")
        self.label2.grid(column=0, row=1, padx=5, pady=5)
        self.dato2=tk.StringVar()
        self.entry2=ttk.Entry(self.lf1,textvariable=self.dato2)
        self.entry2.grid(column=1, row=1, padx=5, pady=5)

        self.label3=ttk.Label(self.lf1, text="Partido C:")
        self.label3.grid(column=0, row=2, padx=5, pady=5)
        self.dato3=tk.StringVar()
        self.entry3=ttk.Entry(self.lf1,textvariable=self.dato3)
        self.entry3.grid(column=1, row=2, padx=5, pady=5)

        self.boton1=ttk.Button(self.lf1, text="Generar gráfico",
command=self.grafico_porcentual)
        self.boton1.grid(column=0, row=3, columnspan=2, padx=5, pady=5,
sticky="we")
        self.entry1.focus()

    def grafico_porcentual(self):
        self.canvas1.delete(tk.ALL)
        valor1=int(self.dato1.get())
        valor2=int(self.dato2.get())
        valor3=int(self.dato3.get())
        suma=valor1+valor2+valor3
        porcentaje1=valor1*100/suma
        porcentaje2=valor2*100/suma
        porcentaje3=valor3*100/suma



        self.canvas1.create_rectangle(10,170,10+(porcentaje1*500/100),230
, fill="red")
        self.canvas1.create_rectangle(10+(porcentaje1*500/100), 170,
10+(porcentaje2*500/100)+(porcentaje1*500/100), 230, fill="green")
        self.canvas1.create_rectangle(10+(porcentaje1*500/100)+(porcentaj
e2*500/100),170,
10+(porcentaje3*500/100)+(porcentaje1*500/100)+(porcentaje2*500/100),
230, fill="blue")
        self.canvas1.create_text(50,190,
text="{0:.2f}".format(porcentaje1)+"%", fill="white", font="Arial")
```

```
        self.canvas1.create_text(50+(porcentaje1*500/100),190,
text="{0:.2f}".format(porcentaje2)+"%", fill="white", font="Arial")
        self.canvas1.create_text(50+(porcentaje1*500/100)+(porcentaje2*50
0/100),190, text="{0:.2f}".format(porcentaje3)+"%", fill="white",
font="Arial")

# Bloque principal
aplicacion1=Aplicacion()
```

Vamos a probar con otros valores:



Capítulo 257.- tkinter: Control Canvas y la captura de eventos del mouse – 1

Una actividad muy común es poder detectar cuando hemos presionado el mouse dentro de un control Canvas (o en cualquier otro Widget que hayamos visto) y a partir de eso disparar un evento.

Veremos una serie de ejemplos las posibilidades de tkinter nos provee.

Problema:

Confeccionar un programa que cree un objeto de la clase Canvas y nos muestre en el título de la ventana las coordenada actual del mouse dentro del control Canvas y al presionar el botón izquierdo del mouse se dibuje un círculo en dicha posición.

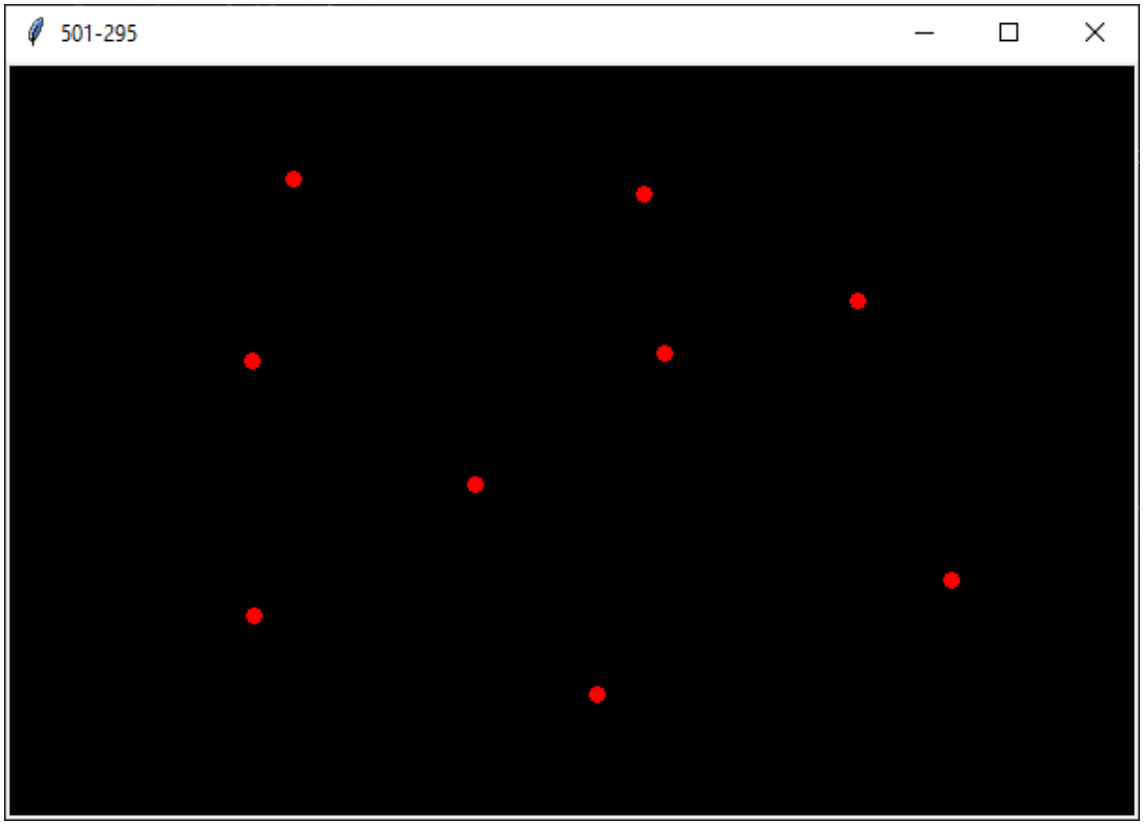
La interfaz visual debe ser similar a esta:

Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400,
background="black")
        self.canvas1.grid(column=0, row=0)
        # Cuando se produce el evento de mover el ratón
```

```
        self.canvas1.bind("<Motion>", self.mover_mouse)
        # Cuando se produce el evento de presionar botón izquierdo
        self.canvas1.bind("<Button-1>", self.presion_mouse)
        self.ventana1.mainloop()

    def mover_mouse(self, evento):
        self.ventana1.title(str(evento.x)+"-"+str(evento.y))

    def presion_mouse(self, evento):
        self.canvas1.create_oval(evento.x-5, evento.y-5, evento.x+5,
evento.y+5, fill="red")

#Bloque principal
aplicacion1=Aplicacion(
```

Este será el resultado:



Acotaciones

La diversidad de eventos que podemos capturar es muy grande, veamos algunos ejemplos:

- Para capturar el evento clic del botón izquierdo del mouse indicamos el método bind <Button-1>, el central <Button-2> y el botón derecho del mouse <Button-3>.
- Si necesitamos capturar el evento clic del botón derecho del mouse y a su vez que encuentre presionada la tecla Shift tenemos que codificar:

self.canvas1.bind("<Shift Button-1>", self.presion_mouse)

En lugar de Shift Podemos verificar si se está presionando la tecla control.

self.canvas1.bind("<Control Button-1>", self.presion_mouse)

Inclusive detector el evento si se presiona Shift, Control y el botón izquierdo del mouse:

self.canvas1.bind("<Control Shift Button-1>", self.presion_mouse)

La Tecla Alt, Shift, Control y el botón izquierdo del mouse:

self.canvas1.bind("<Control Shift Alt Button-1>", self.presion_mouse)

- Si necesitamos hacer algo cuando la flecha del mouse entra al control podemos plantear la captura del evento:

self.canvas1.bind("<Enter>", self.entrada)

Y si queremos detector cuando la flecha del mouse sale de la componete:

self.canvas1.bind("<Leavel>", self.salida)

- Para detector el doble clic de un botón del mouse:

self.canvas1.bind("<Double-Button-1>", self.presion_mouse)

En el código anterior vamos a cambiar el evento <Boton-1> por <Double-Botton-1>

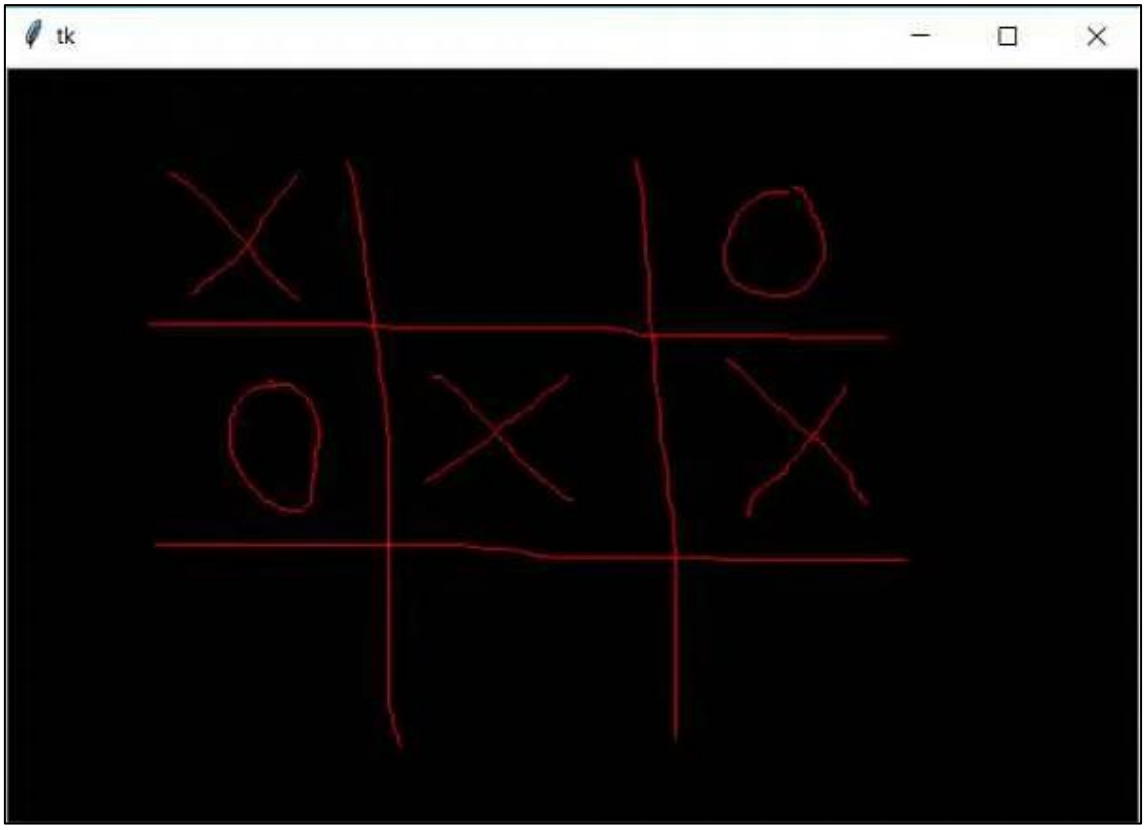
```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400,
background="black")
        self.canvas1.grid(column=0, row=0)
        # Cuando se produce el evento de mover el ratón
        self.canvas1.bind("<Motion>", self.mover_mouse)
        # Cuando se produce el evento de presionar botón izquierdo
        self.canvas1.bind("<Double-Button-1>", self.presion_mouse)
        self.ventana1.mainloop()

    def mover_mouse(self, evento):
        self.ventana1.title(str(evento.x)+"-"+str(evento.y))

    def presion_mouse(self, evento):
        self.canvas1.create_oval(evento.x-5, evento.y-5, evento.x+5,
evento.y+5, fill="red")

#Bloque principal
aplicacion1=Aplicacion()
```

Capítulo 258.- tkinter: Control Canvas y la captura de eventos del mouse – 2

Problema:

Confeccionar un programa que cree un objeto de la clase Canvas y nos permita dibujar a mano alzada dentro del mismo.

La interfaz visual debe ser similar a esta luego de dibujar unos trazos:

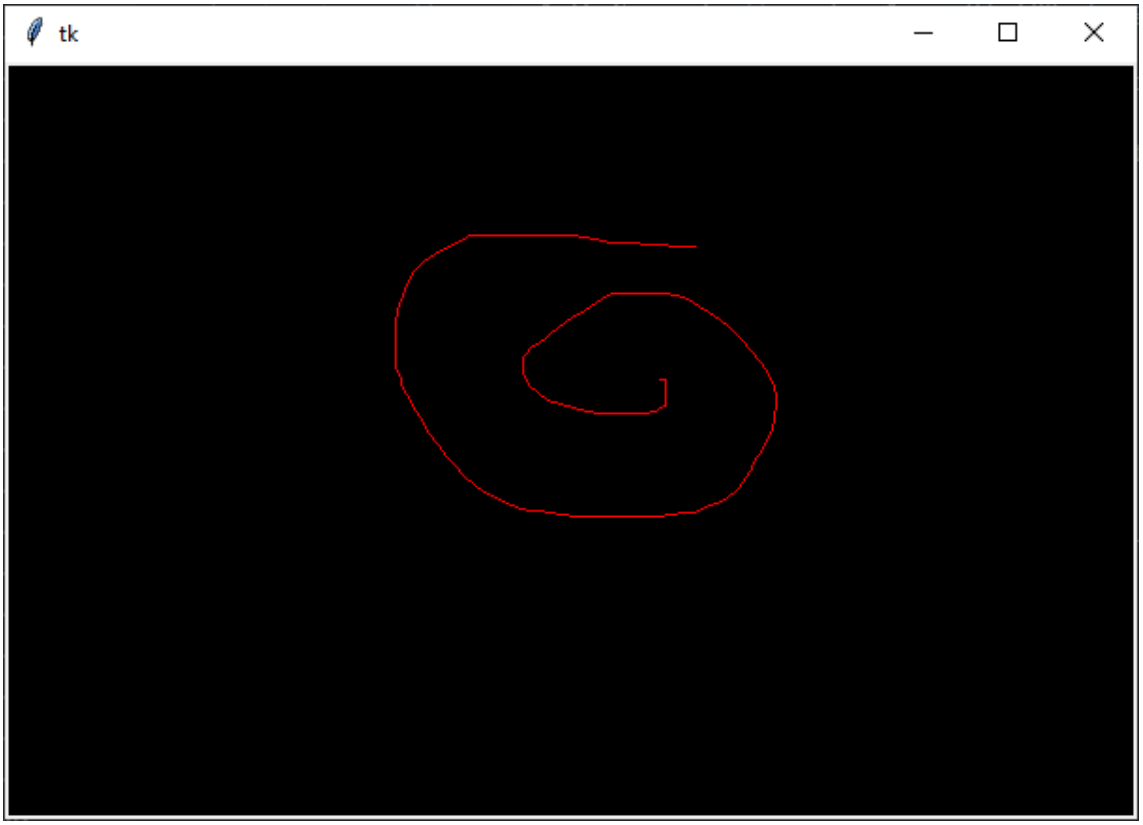


Vamos a programar:

```
import tkinter as tk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400,
background="black")
        self.canvas1.grid(column=0, row=0)
        self.canvas1.bind("<ButtonPress-1>", self.boton_presion)
        self.canvas1.bind("<Motion>", self.mover_mouse)
        self.canvas1.bind("<ButtonRelease-1>", self.boton_soltar)
        self.presionado=False
        self.ventana1.mainloop()

    def boton_presion(self, evento):
        self.presionado=True
```

```
        self.origenx=evento.x
        self.origeny=evento.y

    def mover_mouse(self, evento):
        if self.presionado:
            self.canvas1.create_line(self.origenx, self.origeny,
evento.x, evento.y, fill="red")
            self.origenx=evento.x
            self.origeny=evento.y

    def boton_soltar(self, evento):
        self.presionado=False

# Bloque principal
aplicacion=Aplicacion()
```

Este será el resultado con un nuevo dibujo:



Capítulo 259.- Canvas: Borrar figuras mediante Ids y Tags

Cuando creamos una figura con los métodos que provee la clase Canvas, el mismo nos retorna un Id (identificador) de dicha figura. Luego podemos referencias a la misma mediante ese Id.

Podemos asociar además a una o más figuras con un Tag (marca) y luego borrar todas las figuras que tienen dicho Tag.

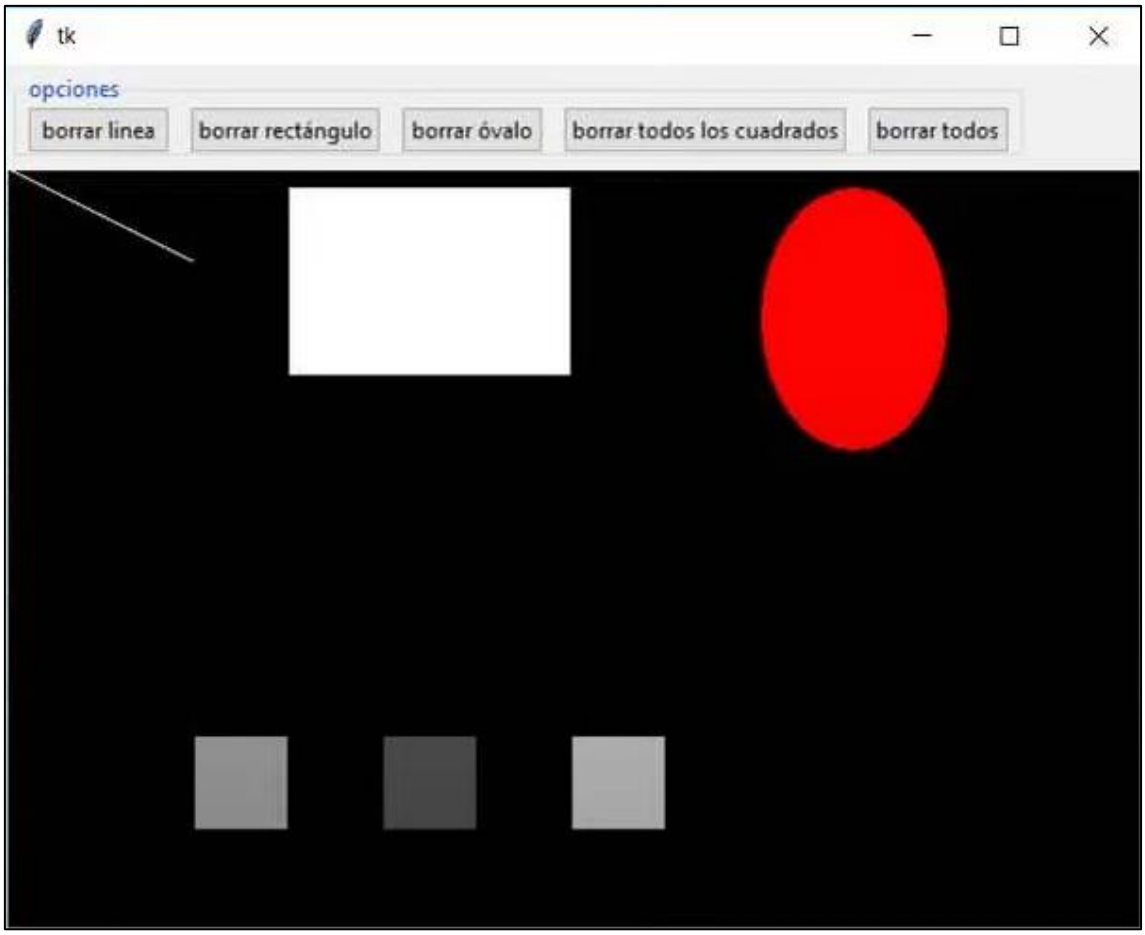
Problema:

Confeccionar un programa que cree un objeto de la clase Canvas y luego dibuje una línea, un rectángulo y un óvalo, almacenar el Id de cada dibujo en un atributo.

Crear también 3 cuadros y definir el parámetro tag con el mismo valor para cada uno de ellos.

Mediante cinco botones permitir: borrar la línea, borrar el rectángulo, borrar el óvalo, borrat todos los cuadrados y borrar todas las figuras contenidas en el objeto Canvas.

La interfaz visual debe ser similar a esta:



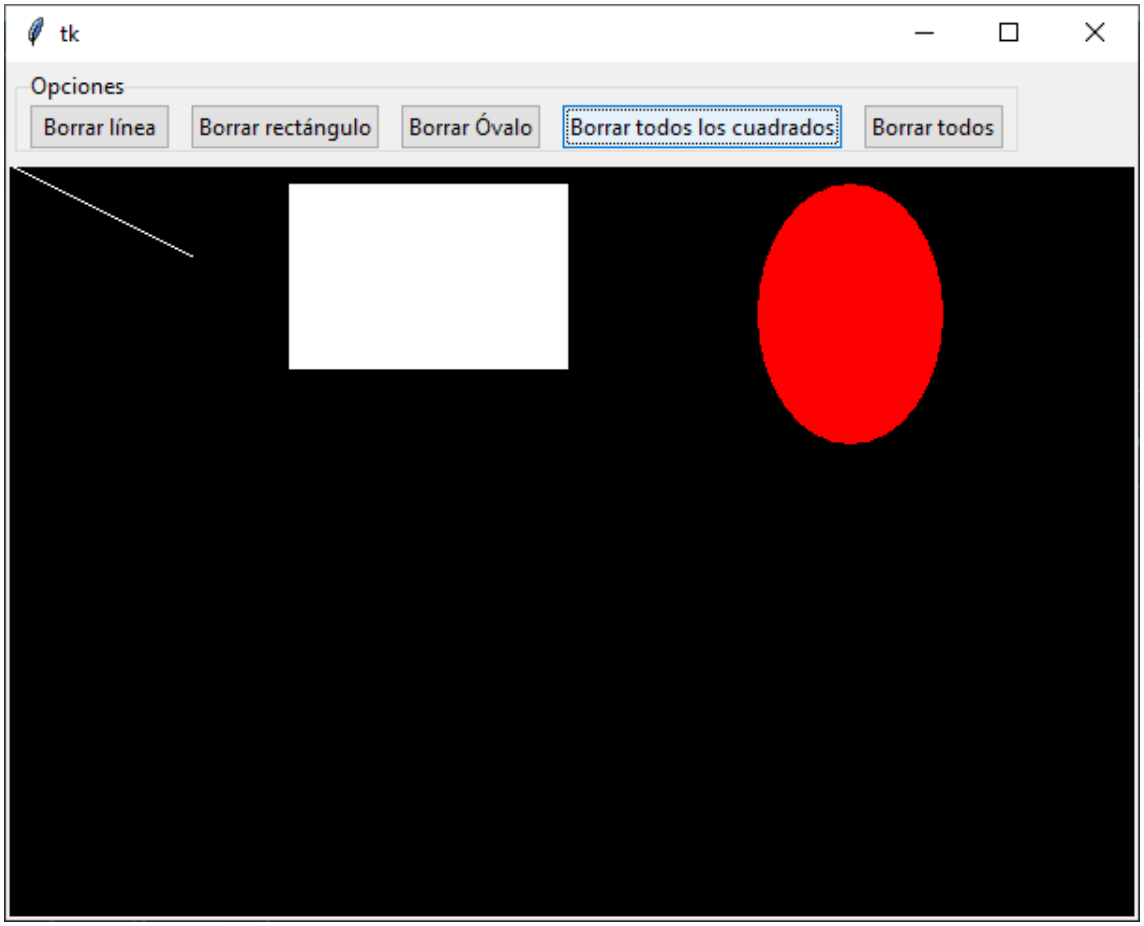
Vamos a programar:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:
```

```
    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400,
background="black")
        self.canvas1.grid(column=0, row=1)
        self.linea=self.canvas1.create_line(0,0,100,50, fill="white")
        self.ovalo=self.canvas1.create_oval(400,10,500,150, fill="red")
        self.rectangulo=self.canvas1.create_rectangle(150,10, 300, 110,
fill="white")
        self.canvas1.create_rectangle(100,300,150,350, fill="#aaaaaa",
tag="cuadrado")
        self.canvas1.create_rectangle(200,300,250, 350, fill="#555555",
tag="cuadrado")
        self.canvas1.create_rectangle(300,300, 350,350, fill="#cccccc",
tag="cuadrado")
        self.crear_botones()
        self.ventana1.mainloop()

    def crear_botones(self):
        self.labelframe1=ttk.LabelFrame(self.ventana1, text="Opciones")
        self.labelframe1.grid(column=0, row=0, sticky="w", padx=5,
pady=5)
        self.boton1=ttk.Button(self.labelframe1, text="Borrar línea",
command=self.borrar_linea)
        self.boton1.grid(column=0, row=0, padx=5)
        self.boton2=ttk.Button(self.labelframe1, text="Borrar
rectángulo", command=self.borrar_rectangulo)
        self.boton2.grid(column=1, row=0, padx=5)
        self.boton3=ttk.Button(self.labelframe1, text="Borrar Óvalo",
command=self.borrar_ovalo)
        self.boton3.grid(column=2, row=0, padx=5)
        self.boton4=ttk.Button(self.labelframe1, text="Borrar todos los
cuadrados", command=self.borrar_cuadrados)
        self.boton4.grid(column=3, row=0, padx=5)
        self.boton5=ttk.Button(self.labelframe1, text="Borrar todos",
command=self.borrar_todos)
        self.boton5.grid(column=4, row=0, padx=5)

    def borrar_linea(self):
        self.canvas1.delete(self.linea)

    def borrar_rectangulo(self):
        self.canvas1.delete(self.rectangulo)

    def borrar_ovalo(self):
        self.canvas1.delete(self.ovalo)

    def borrar_cuadrados(self):
        self.canvas1.delete("cuadrado")
```

```
    def borrar_todos(self):
        self.canvas1.delete(tk.ALL)

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado una vez seleccionado el botón 'borrar todos los cuadrados'.



Los identificadores los guardamos en variables.

```
self.linea=self.canvas1.create_line(0,0,100,50, fill="white")
```

Para eliminar un objeto:

```
self.canvas1.delete(self.linea)
```

Para asignar marcas a los objetos:

```
self.canvas1.create_rectangle(100,300,150,350, fill="#aaaaaa",
tag="cuadrado")
```

Para eliminar los objetos con la misma marca:

```
self.canvas1.delete("cuadrado")
```

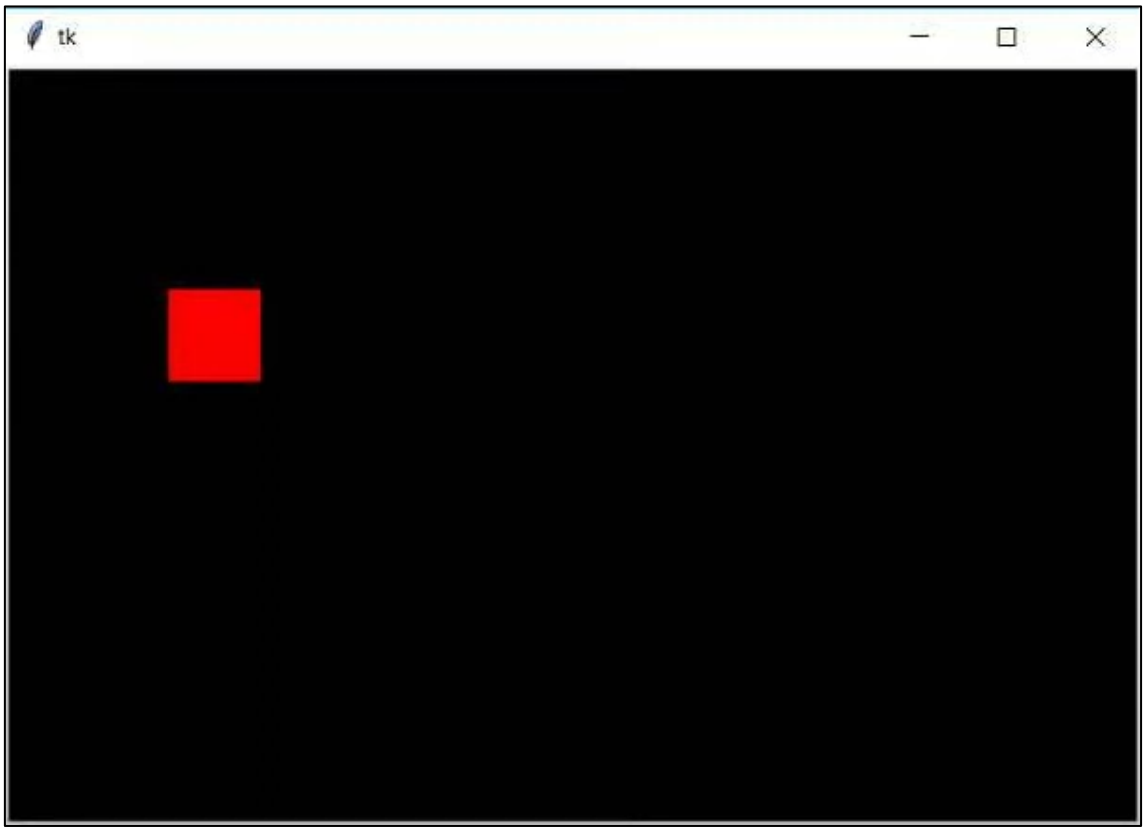
Capítulo 260.- Canvas: Desplazar una figura mediante el método move – 1

Otra situación donde tiene sentido almacenar el Id de una figura es cuando en un momento posterior queremos desplazarla a otra posición dentro del objeto Canvas.

Problema:

Confeccionar un programa que muestre un cuadrado dentro de un Canvas. Cuando se presione alguna de las teclas de flecha proceder a desplazar la figura 4 píxeles teniendo en cuenta la dirección de la tecla de flecha presionada.

La interfaz visual debe ser similar a esta:

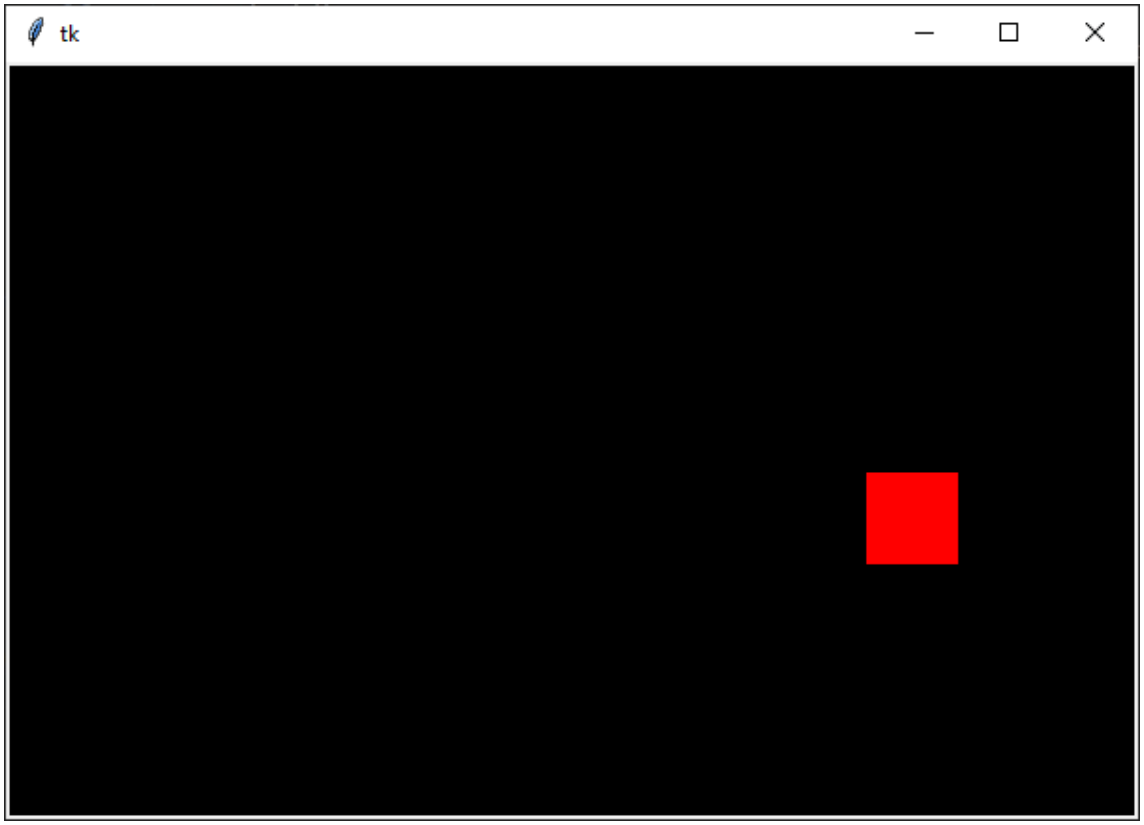


Vamos a programar:

```
class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400,
background="black")
        self.canvas1.grid(column=0, row=0)
        self.cuadrado=self.canvas1.create_rectangle(150,10,200, 60,
fill="red")
        self.ventana1.bind("<KeyPress>", self.presion_tecla)
        self.ventana1.mainloop()
```

```
    def presion_tecla(self,evento):
        if evento.keysym=="Right":
            self.canvas1.move(self.cuadrado,4, 0)
        if evento.keysym=="Left":
            self.canvas1.move(self.cuadrado,-4,0)
        if evento.keysym=="Down":
            self.canvas1.move(self.cuadrado,0,4)
        if evento.keysym=="Up":
            self.canvas1.move(self.cuadrado, 0,-4)

# Bloque principal
aplicacion=Aplicacion()
```

Este será el resultado cuando lo movemos por el Canvas:



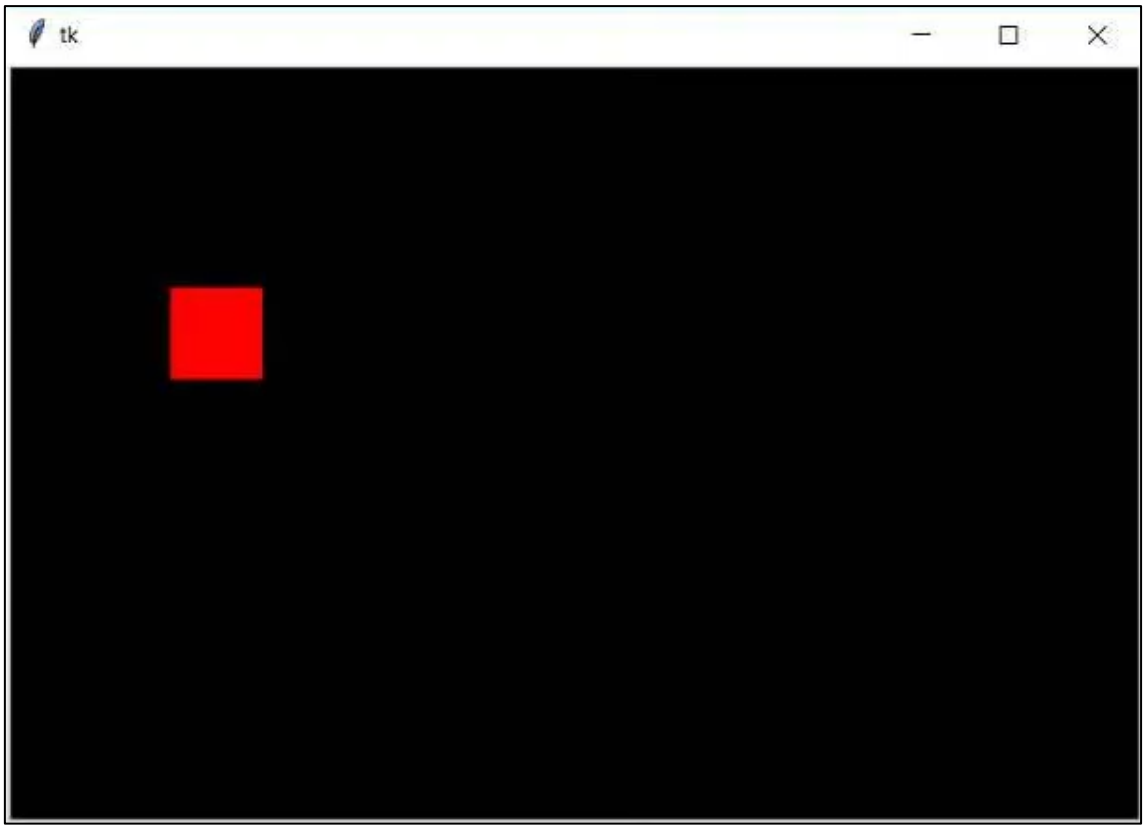
Capítulo 261.- Canvas: Desplazar una figura mediante el método move – 2

Problema propuesto:

Modificar el problema que desplaza un cuadrado con las flechas del tal modo que la figura no pueda salir del espacio definido por el Canvas.

Para saber la posición actual de una figura de clase Canvas cuenta con el método 'coords':

x1, y1, x2, y2 = self.canvas1.coords(self.cuadrado)



Vamos a modificar el programa anterior:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=600, height=400,
background="black")
        self.canvas1.grid(column=0, row=0)
        self.cuadrado=self.canvas1.create_rectangle(150,10,200, 60,
fill="red")
        self.ventana1.bind("<KeyPress>", self.presion_tecla)
        self.ventana1.mainloop()
```

```
    def presion_tecla(self,evento):
        x1, y1, x2, y2 = self.canvas1.coords(self.cuadrado)
        if evento.keysym=="Right":
            if x2+4<=600:
                self.canvas1.move(self.cuadrado,4, 0)
        if evento.keysym=="Left":
            if x1-4>=0:
                self.canvas1.move(self.cuadrado,-4,0)
        if evento.keysym=="Down":
            if y2+4<400:
                self.canvas1.move(self.cuadrado,0,4)
        if evento.keysym=="Up":
            if y1-4>=0:
                self.canvas1.move(self.cuadrado, 0,-4)

# Bloque principal
aplicacion=Aplicacion()
```

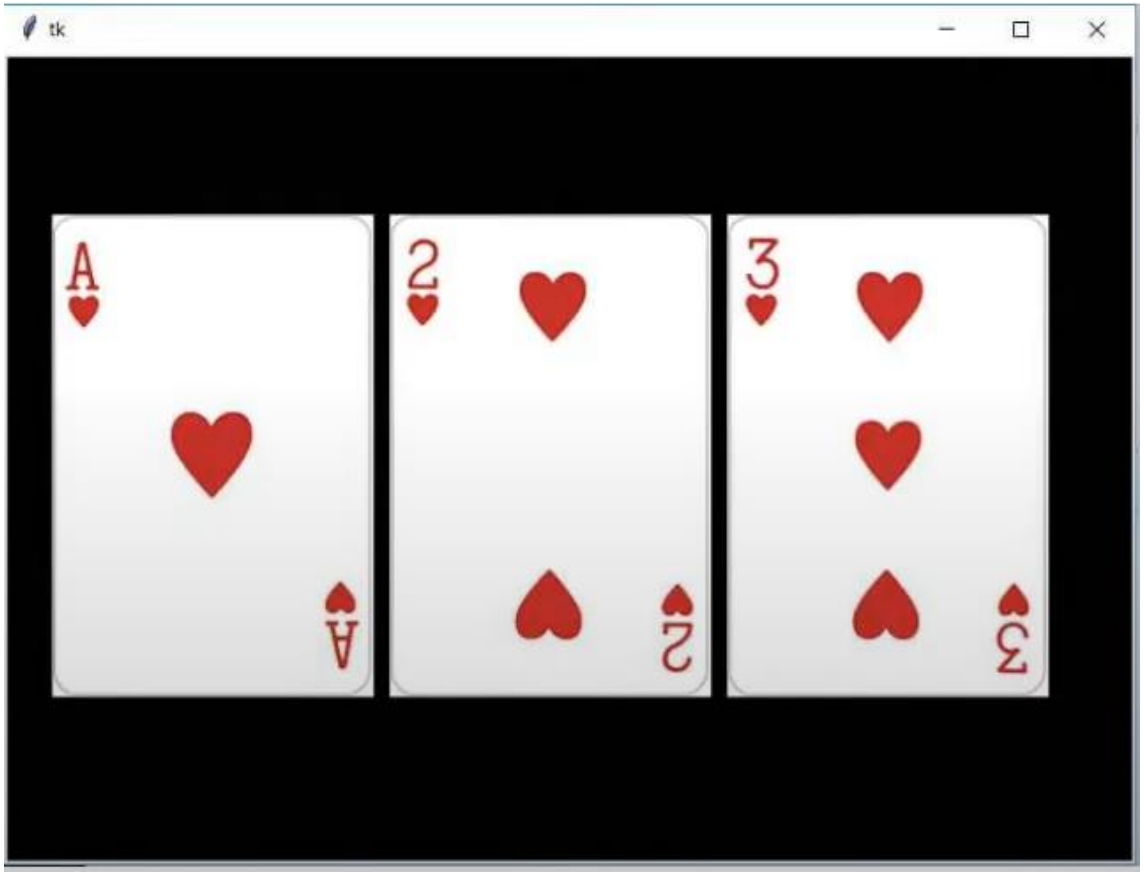
Ahora cuando lo ejecutemos el cuadrado no saldrá del Canvas.

Capítulo 262.- Canvas: Mostrar una imagen – 1

Si necesitamos mostrar una imagen dentro de un componente tipo Canvas disponemos del método 'PhotoImage' que se encuentra en el módulo tkinter para su lectura, y luego mediante el método 'create_image' de la clase Canvas para su visualización.

Problema

Se cuenta con tres archivos de tipo png con las imágenes de distintas cartas. Mostrarlas a cada una dentro de un componente de tipo Canvas.



Vamos a programar:

```
import tkinter as tk

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=700, height=500,
background="black")
        self.canvas1.grid(column=0, row=0)
        archi1=tk.PhotoImage(file="carta1.png")
        self.canvas1.create_image(30,100, image=archi1, anchor="nw")
        archi2=tk.PhotoImage(file="carta2.png")
        self.canvas1.create_image(250,100, image=archi2, anchor="nw")
        archi3=tk.PhotoImage(file="carta3.png")
```

```
        self.canvas1.create_image(450,100, image=archi3, anchor="nw")
        self.ventana1.mainloop()

# Bloque principal
aplicacion1=Aplicacion()
```

Las imágenes se deben encontrar en la misma carpeta donde se encuentra el archivo, si se encontrara en una subcarpeta habría que indicarle la ruta:

```
archi1=tk.PhotoImage(file="imagenes/carta1.png")
```

Suponiendo que se encuentra en una subcarpeta llamada imágenes.

También se puede expecificar la ruta completa:

```
archi1=tk.PhotoImage(file="c:/pythonya/imagenes/carta1.png")
```

Por defecto el parámetro anchor tiene el valor "center".

Los valores posibles del parámetro anchor son: "n", "ne", "e", "se", "s", "sw", "w", "nw" y "center".

Anotaciones

Los formatos reconocidos de la clase PhotoImage son: GIF, PNG, PGM y PPM.

Si el archivo se encuentra en otra carpeta debemos indicar el path del mismo:

archi1=tk.PhotoImage(file="C:/programaspython/carta1.png")

Capítulo 263.- Canvas: Mostrar una imagen – 2

Problema:

Disponer un botón y mostrar al azar una de las tres cartas del problema anterior. Cada vez que se presione el botón generar un valor aleatorio y a partir de dicho valor mostrar una carta.

Vamos a programar:

```
import tkinter as tk
from tkinter import ttk
import random

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=300, height=500,
background="black")
        self.canvas1.grid(column=0, row=0)
        self.boton1=ttk.Button(self.ventana1, text="Mostrar carta
aleatoria", command=self.mostrar)
        self.boton1.grid(column=0, row=1)
        self.archi1=tk.PhotoImage(file="carta1.png")
        self.archi2=tk.PhotoImage(file="carta2.png")
        self.archi3=tk.PhotoImage(file="carta3.png")
        self.ventana1.mainloop()

    def mostrar(self):
        valor=random.randint(1,3)
        if valor==1:
            self.canvas1.create_image(60,100, image=self.archi1,
anchor="nw")
        elif valor==2:
            self.canvas1.create_image(60,100, image=self.archi2,
anchor="nw")
        elif valor==3:
            self.canvas1.create_image(60,100, image=self.archi3,
anchor="nw")

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:

tk
Mostrar carta aleatoria

Capítulo 264.- Canvas: Mover una figura – 1

Otro algoritmo útil cuando trabajamos con figuras dentro del un control Canvas es la posibilidad de arrastrar y soltar figuras.

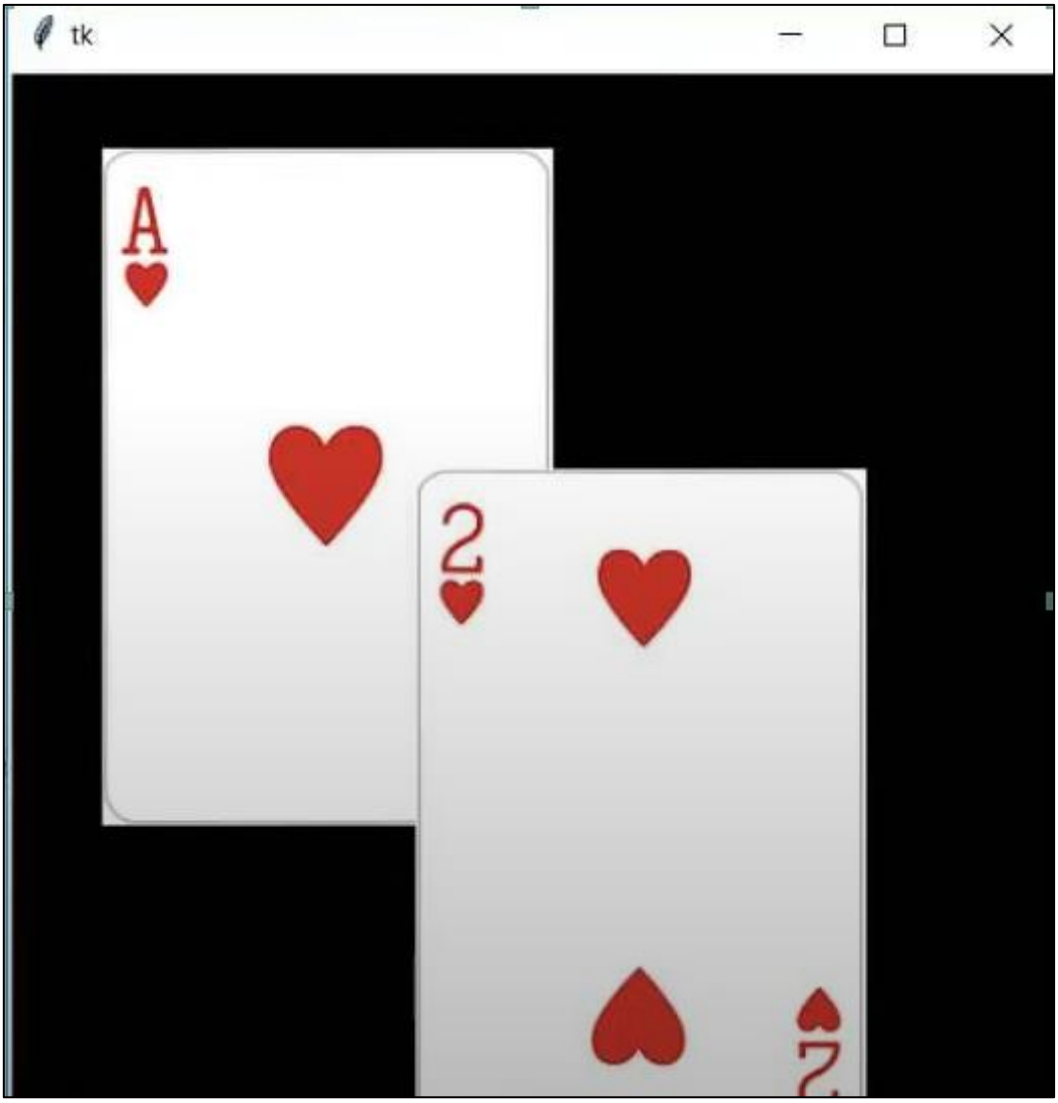
Debemos capturar los eventos cuando se hace clic dentro de la figura y luego cada vez que desplaza la flecha de mouse.

Problema:

Se cuenta con dos archivos de tipo png con las imágenes de distintas cartas. Mostrarlas a cada una dentro de un componente de tipo Canvas y permitir moverlas con el mouse dentro del control mediante el mouse.

La interfaz visual debe ser similar a esta, luego mover con el mouse las imágenes:

Por defecto el parámetro anchor tiene el valor "center".

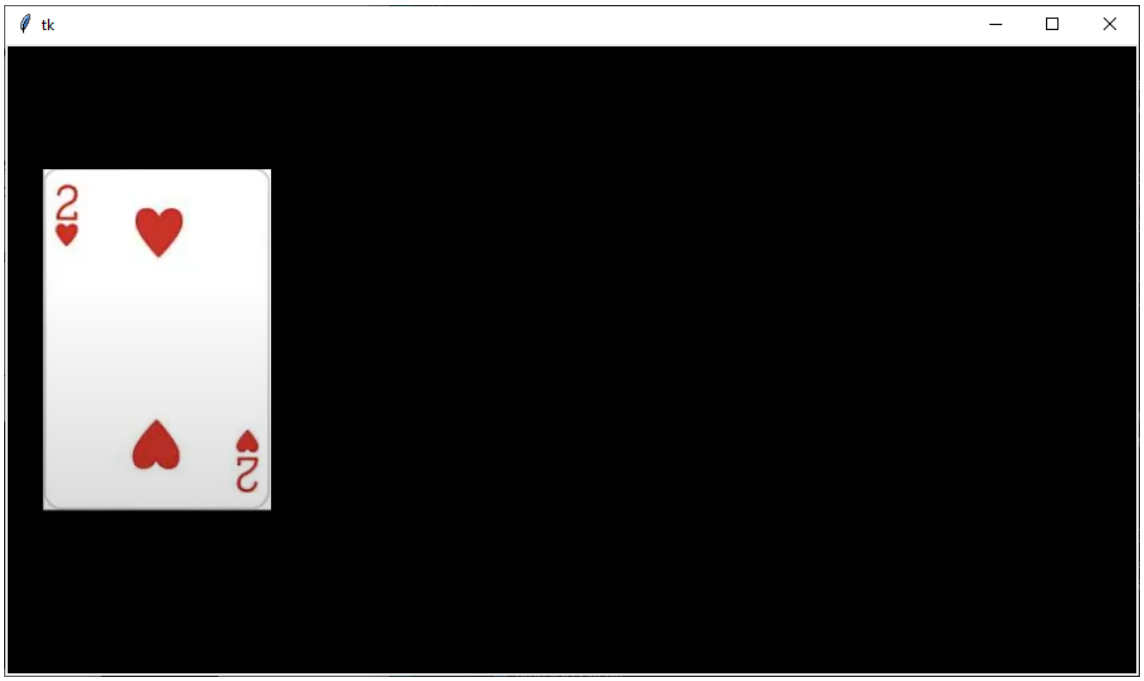


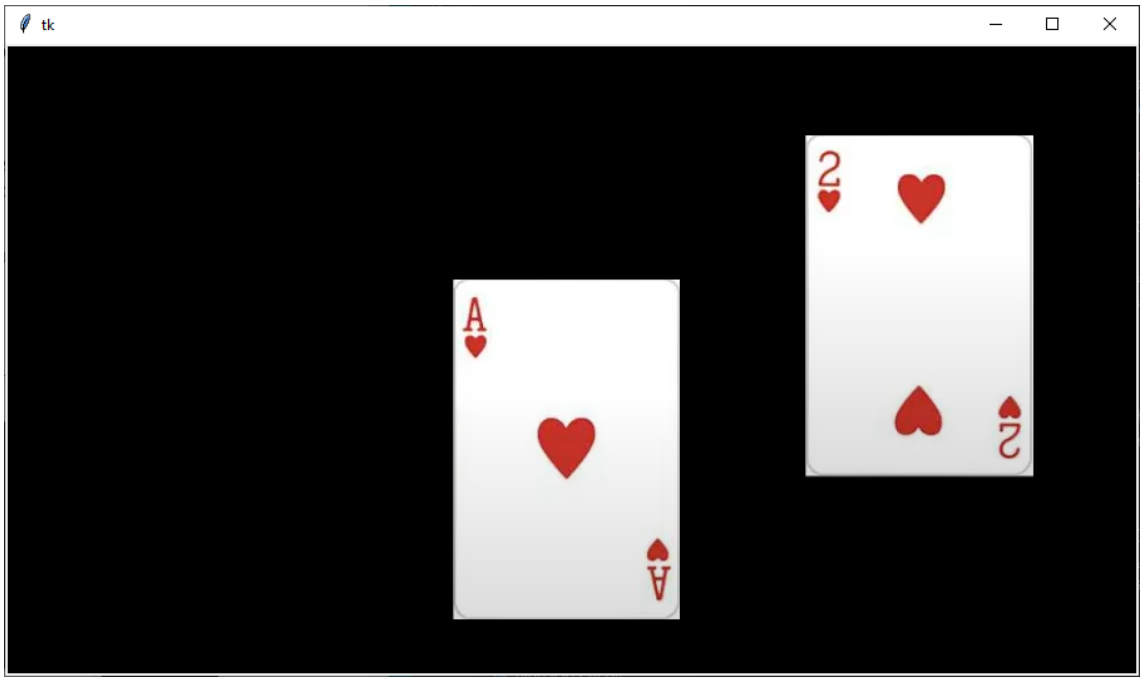
Vamos a programar:

```
import tkinter as tk
```

```
class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=900, height=500,
background="black")
        self.canvas1.grid(column=0, row=0)

        archi1=tk.PhotoImage(file="carta1.png")
        self.canvas1.create_image(30,100, image=archi1, anchor="nw",
tags="movil")
        archi2=tk.PhotoImage(file="carta2.png")
        self.canvas1.create_image(30,100, image=archi2, anchor="nw",
tags="movil")

        self.canvas1.tag_bind("movil", "<ButtonPress-1>",
self.presion_boton)
        self.canvas1.tag_bind("movil", "<Button1-Motion>", self.mover)
        self.carta_seleccionada=None
        self.ventana1.mainloop()

    def presion_boton(self, evento):
        # La variable carta almacena la carta actual que se ha
presionado.
        carta=self.canvas1.find_withtag(tk.CURRENT)
        # En la variable carta_seleccionada almacenamos una tupla
        # que es la carta que se acaba de presionar y sus coordenadas.
        self.carta_seleccionada=(carta, evento.x, evento.y)

    def mover(self, evento):
        x,y=evento.x, evento.y
        # Recuperamos los valores en tupla de la variable
        # carta_seleccionada.
        carta, x1, y1=self.carta_seleccionada
        # Movemos la carta a la posicion x,y a nueva posición que será
x1, y1
        self.canvas1.move(carta, x-x1, y-y1)
        # Actualizamos la carta seleccionada con la nueva posición x,y
        self.carta_seleccionada=(carta, x, y)

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:





Ya podemos mover las cartas.

Capítulo 265.- Canvas: Mover una figura – 2

Problema propuesto:

Crea 100 cuadrados de color rojo y disponerlos en el control Canvas en posiciones aleatorias. Permitir desplazar con el mouse cualquiera de los cuadros.

Vamos a programar:

```
import tkinter as tk
import random

class Aplicacion:

    def __init__(self):
        self.ventana1=tk.Tk()
        self.canvas1=tk.Canvas(self.ventana1, width=900, height=600,
background="black")
        self.canvas1.grid(column=0, row=0)
        self.dibujar_cuadros()
        self.ventana1.mainloop()

    def dibujar_cuadros(self):
        for i in range(100):
            x=random.randint(10, 880)
            y=random.randint(10,580)
            self.cuadrado=self.canvas1.create_rectangle(x,y, x+20, y+20,
fill="red", outline="black", tags="movil")
            self.canvas1.tag_bind("movil", "<ButtonPress-
1>",  self.presion_boton)
            self.canvas1.tag_bind("movil", "<Button1-Motion>",
self.mover)
            self.cuadro_seleccionado=None

    def presion_boton(self, evento):
        cuadra=self.canvas1.find_withtag(tk.CURRENT)
        self.cuadro_seleccionado=(cuadra, evento.x, evento.y)

    def mover(self, evento):
        x,y=evento.x, evento.y
        cuadra, x1, y1=self.cuadro_seleccionado
        self.canvas1.move(cuadra, x-x1, y-y1)
        self.cuadro_seleccionado=(cuadra, x, y)

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:Aplicaciónes mover los cuadrados.

Capítulo 266.- Archivos de texto: Creación, escritura y lectura – 1

Una actividad muy común en un programa es el almacenamiento y recuperación de datos almacenado en un dispositivo secundario (disco duro).

Existen muchos modos de almacenar datos como son los archivos de texto, archivos binarios, base de datos, etc.

Veremos en este concepto como almacenar y recuperar datos de un archivo de texto.

Archivo de texto

Un archivo de texto contiene un conjunto de caracteres estructurados en distintas líneas. Es un formato de archivo ampliamente utilizado como pueden ser:

- El código fuente de un script en Python se almacena en un archivo de texto (igual que cualquier otro lenguaje de programación)
- Archivos HTML, CSS, XML se almacenan en archivos de texto.
- Archivos JSON etc.

Creación de un archivo de texto y almacenamiento de datos

Como es una actividad tan común en todo programa de lenguaje Python incluye por defecto todas las funcionalidades para trabajar con archivos de texto.

Veamos con un ejemplo como crear y almacenar caracteres en un archivo de texto.

Problema:

Crear un archivo de texto llamado 'datos.txt', almacenar tres líneas de texto. Abrir luego el archivo creado con un editor de texto.

Vamos a programar:

ejercicio267.py > ...

```
1   archi1=open("datos.txt", "w")
2   archi1.write("Primera linea\n")
3   archi1.write("Segunda linea\n")
4   archi1.write("Tercera linea\n")
5   archi1.close()
6   print("Fin de programa")
```

Lo vamos a ejecutar:

En la consola nos avisa que ya se ha ejecutado el programa.

Con el mismo editor de Visual Studio Code podemos ver el archivo.

datos.txt

```
1   Primera linea
2   Segunda linea
3   Tercera linea
```

Vamos a modificar el programa agregando una cuarta línea.

ejercicio267.py > ...

```
1   archi1=open("datos.txt", "w")
2   archi1.write("Primera linea\n")
3   archi1.write("Segunda linea\n")
4   archi1.write("Tercera linea\n")
5   archi1.write("Cuarta linea\n")
6   archi1.close()
7   print("Fin de programa")
```

Lo vamos a ejecutar de nuevo:

datos.txt

```
1   Primera linea
2   Segunda linea
3   Tercera linea
4   Cuarta linea
```

El contenido del archivo ha sido reemplazado.

En la línea 1 le decimos el nombre del archivo, si este se tiene que guardar en otra ubicación de donde está el programa tendremos que especificarle la ruta.

Con el método write guardamos los datos en el archivo de texto.

En la línea 6 cerramos el archivo, esto siempre es recomendable.

Capítulo 267.- Archivos de texto: Creación, escritura y lectura – 2

Lectura de un archivo de texto.

Em método read permite leer todo o un conjunto de caracteres del archivo de texto.

Problema:

Leer el contenido del archivo de texto 'datos.txt'.

Vamos a programar:

```
ejercicio268.py > ...
1   archi1=open("datos.txt", "r")
2   linea=archi1.readline()
3   while linea!="":
4       print(linea, end="")
5       linea=archi1.readline()
6   archi1.close()
```

Si lo ejecutamos este será el resultado en la consola:

```
PS D:\Curso completo de Python> & C:/Users/pmver/AppData/Local/Programs
/Python/Python311/python.exe "d:/Curso completo de Python/ejercicio268.
py"
Primera linea
Segunda linea
Tercera linea
Cuarta linea
PS D:\Curso completo de Python>
```

En la línea 4 le decimos que cuando imprima la línea no realice el salto de línea con la instrucción end="", si los quitamos este será el resultado:

```
PS D:\Curso completo de Python> & C:/Users/pmver/AppData/Local/Programs
/Python/Python311/python.exe "d:/Curso completo de Python/ejercicio268.
py"
Primera linea

Segunda linea

Tercera linea

Cuarta linea
```

Hay otra forma de poder realizar el mismo programa:

ejercicio268.py > ...

```python
archi1=open("datos.txt", "r")
for linea in archi1:
    print(linea, end="")
archi1.close()
```

Si lo ejecutamos el resultado será el mismo.

Capítulo 268.- Archivos de texto: Creación, escritura y lectura – 3

Problema:

Leer el contenido del archivo de texto 'datos.txt' y almacenar sus líneas en una lista. Imprimir la cantidad de líneas que tiene el archivo y su contenido.

Vamos a programar:

```
ejercicio269.py > ...
1   archi1=open("datos.txt", "r")
2   lineas=archi1.readlines()
3   archi1.close()
4   print("La cantidad de líneas que tiene el archivo de texto:", len(lineas))
5   print("Contenido del archivo de texto")
6   for linea in lineas:
7       print(linea, end="")
```

Si ejecutamos este será el resultado:

```
PS D:\Curso completo de Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/
python.exe "d:/Curso completo de Python/ejercicio269.py"
La cantidad de líneas que tiene el archivo de texto: 4
Contenido del archivo de texto
Primera linea
Segunda linea
Tercera linea
Cuarta linea
PS D:\Curso completo de Python>
```

En la línea 4 creamos una lista llamada líneas que guardará todas las líneas que tiene el archivo de texto "datos.txt", a continuación cerramos el archivo ya que la información la tenemos en la lista líneas.

En la línea 4 con la función len(líneas) podemos saber el número de líneas que contiene la lista.

En la línea 5 hacemos un ciclo que recorre por todos los elementos de la lista para imprimir su contenido.

Capítulo 269.- Archivos de texto: Creación, escritura y lectura – 4

Abrir un archivo de texto para añadir líneas.

Hemos visto que cuando llamamos a la función 'open' el segundo parámetro puede ser "w", "r" y si queremos que se abra para añadir sin borrar las líneas actuales del archivo debemos abrirlo con el parámetro "a" (append).

Problema:

Abrir el archivo de texto 'datos.txt' y luego agregar 2 líneas. Imprimir luego el archivo completo.

Vamos a programar:

```
ejercicio270.py > ...
archi1=open("datos.txt", "a")
archi1.write("quita linea\n")
archi1.write("sexta linea\n")
archi1.close()
archi1=open("datos.txt", "r")
contenido=archi1.read()
archi1.close()
print(contenido)
```

Si ejecutamos este será el resultado:

```
PS D:\Curso completo de Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/
python.exe "d:/Curso completo de Python/ejercicio270.py"
Primera linea
Segunda linea
Tercera linea
Cuarta linea
quita linea
sexta linea
```

Abrir un archivo para leer y agregar datos.

Hay una cuarta forma de abrir un archivo indicando en el segundo parámetro de la función open el string "r+", con dicha opción podemos leer y escribir.

Problema:

Abrir un archivo de texto con el parámetro "r+", imprimir su contenido actual, agregar luego dos líneas al final.

```
ejercicio270.py > ...
archi1=open("datos.txt", "r+")
contenido=archi1.read()
archi1.write("septima linea\n")
archi1.write("octava linea\n")
archi1.close()
```

Cuando lo ejecutemos y abrimos el archivo de texto:

datos.txt

```
1   Primera linea
2   Segunda linea
3   Tercera linea
4   Cuarta linea
5   quita linea
6   sexta linea
7   septima linea
8   octava linea
```

Capítulo 270.- Archivos de texto: Creación, escritura y lectura – 5

Codificación de caracteres utf-8.

Actualmente se utiliza mucho la codificación de caracteres utf-8 la que nos permite representar una infinidad de caracteres de distintos idiomas y símbolos.

En Python debemos indicar cuando abrimos el archivo de texto mediante el parámetro 'encoding' la codificación de caracteres utilizada.

Problema:

Crear un archivo de texto llamado 'datos.txt' con una codificación utf-8, almacenar tres líneas de texto y de caracteres que representa el caballo en el juego de ajedrez.

Abrir luego el archivo creado con el editor VS Code u otro editor que permita leer archivos con formato utf-8.

Vamos a programar:

```
ejercicio271.py > ...
1   archi1=open("datos.txt", "w", encoding="utf-8")
2   archi1.write("Línea 1\n")
3   archi1.write("Línea 2\n")
4   archi1.write("Línea 3\n")
5   archi1.write("♞\n")
6   archi1.close()
```

Ejecutamos:

Desde VS Code vamos a ver el contenido del archivo:

```
datos.txt
1   Línea 1
2   Línea 2
3   Línea 3
4   ♞
```

Ha reconocido los acentos y el carácter del caballo de ajedrez.

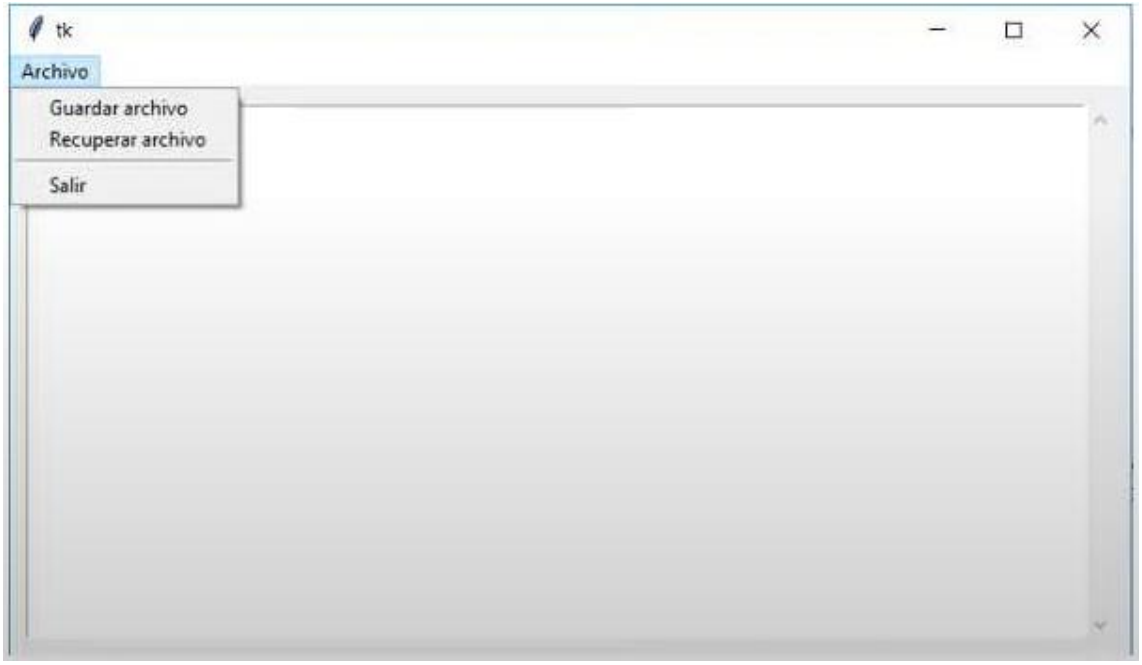
Capítulo 271.- Archivos de texto: Interfaz visual tkinter para mostrar y recuperar archivos de texto

Conocido como poder crear, leer y escribir en un archivo de texto, vamos a combinarlo con los conceptos de la librería visual tkinter para implementar un programa que nos permita escribir un testo y guardarlo en un archivo de texto.

Problema:

Confeccionar un interfaz visual que contenga un menú de opciones que permita "Guardar archivo", "Recuperar archivo" y "Salir del programa".

En la ventana principal debe aparecer un control de tipo 'scrolledtext' donde el operador pueda escribir un texto para luego grabarlo en un archivo de texto. También el control 'scrolledtext' debe cargarse con el contenido del archivo existente en el disco duro.



Vamos a programar:

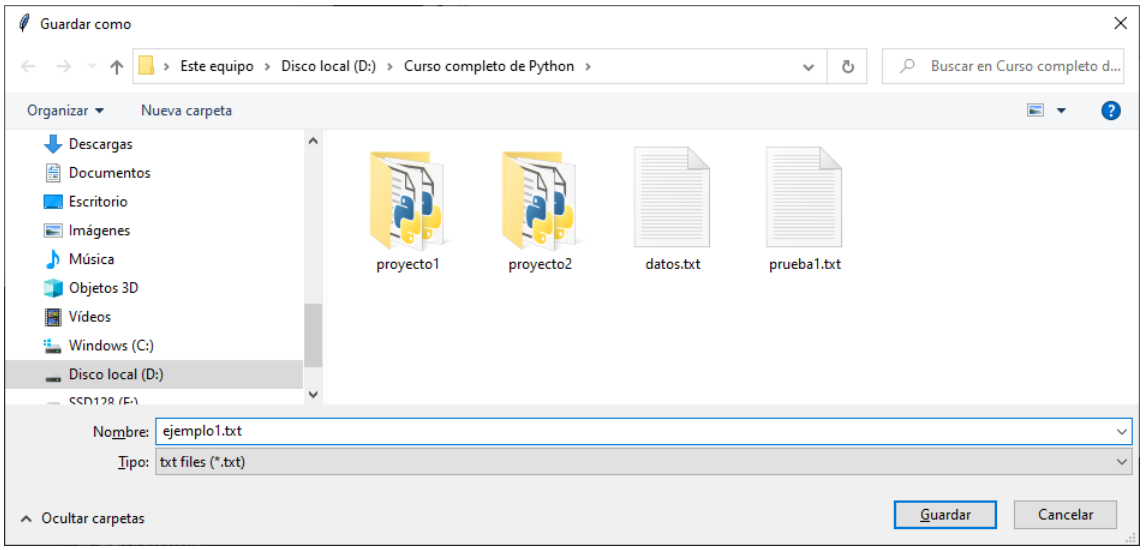
```
import tkinter as tk
from tkinter import scrolledtext as st
import sys
from tkinter import filedialog as fd
from tkinter import messagebox as mb

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.agregar_menu()
        self.scrolledtext=st.ScrolledText(self.ventana1, width=80,
height=20)
        self.scrolledtext.grid(column=0, row=0, padx=10, pady=10)
        self.ventana1.mainloop()
```

```
    def agregar_menu(self):
        menubar1=tk.Menu(self.ventana1)
        self.ventana1.config(men=menubar1)
        opciones1=tk.Menu(menubar1, tearoff=0)
        opciones1.add_command(label="Guardar archivo",
command=self.guardar)
        opciones1.add_command(label="Recuperar archivo",
command=self.recuperar)
        opciones1.add_separator()
        opciones1.add_command(label="Salir", command=self.salir)
        menubar1.add_cascade(label="Archivo", menu=opciones1)


    def guardar(self):
        nombrearch=fd.asksaveasfilename(initialdir="D:/Curso completo de
Python", title="Guardar como", filetypes=(("txt files", "*.txt"), ("todos
los archivos", "*.*")))
        if nombrearch!='':
            archi1=open(nombrearch, "w", encoding="utf-8")
            archi1.write(self.scrolledtext.get("1.0", tk.END))
            archi1.close()
            mb.showinfo("Información", "Los datos fueron guardados en el
archivo.")

    def recuperar(self):
        nombrearch=fd.askopenfilename(initialdir="D:/Curso completo de
Python", title="Seleccione archivo", filetypes=(("txt files", "*.txt"),
("todos los archivos", "*.*")))
        if nombrearch!='':
            archi1=open(nombrearch, "r", encoding="utf-8")
            contenido=archi1.read()
            archi1.close()
            self.scrolledtext.delete("1.0", tk.END)
            self.scrolledtext.insert("1.0", contenido)

    def salir(self):
        sys.exit(0)

# Bloque principal
aplicacion1=Aplicacion()
```

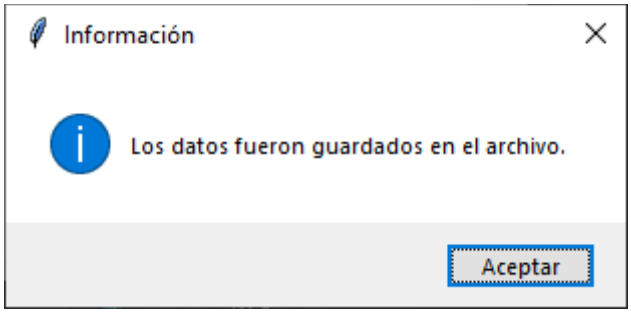
Vamos a ejecutar:

Vamos a escribir el siguiente texto:

Del menú Archivo seleccionaremos Guardar archivo:



Como nombre pondremos ejemplo1.txt seguido del botón guardar.

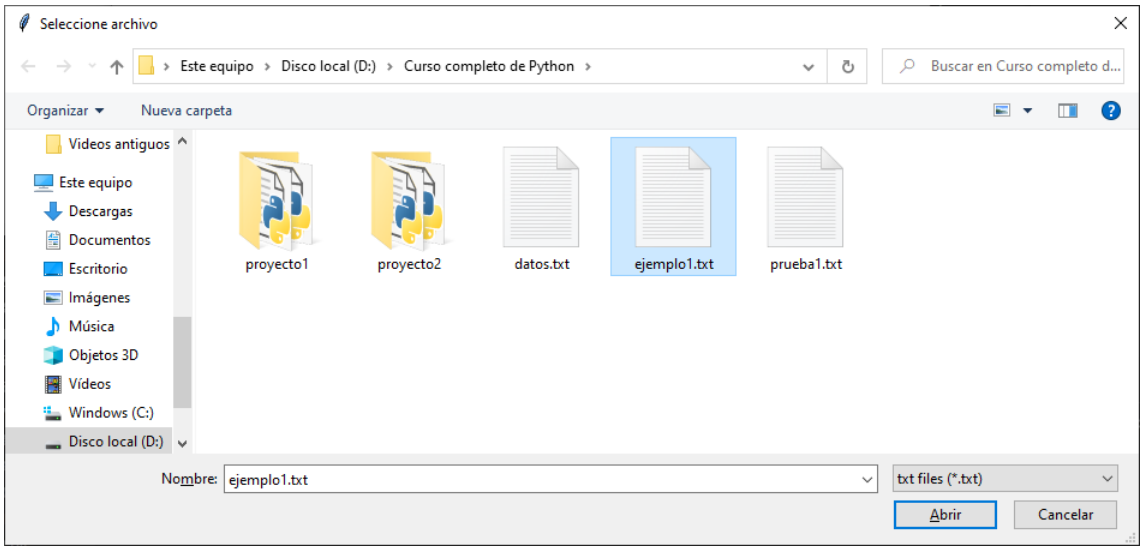


Obtenemos un mensaje de confirmación.

A continuación del menú Archivo seleccionaremos Salir.

Ahora lo volvemos a ejecutar de nuevo.

Del menú Archivo seleccionaremos Recuperar archivo.



Seleccionamos el documento que creamos con anterioridad ejemplo1.txt seguido del botón Abrir.

Ya hemos recuperado el contenido del archivo.

Capítulo 272.- MySQL: Base de datos desde Python – 1

Cuando tenemos que almacenar gran cantidad de datos y su posterior procesamiento es muy común utilizar un gestor de base de datos.

Con Python podemos comunicarnos con un gestor de base de datos y el primero que veremos para ver cual es la mecánica que conectarnos desde Python será el gestor de base de datos MySQL.

Debemos instalar primero si no lo tenemos a MySQL.

Instalación de MySQL

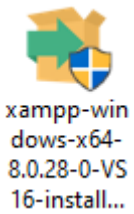
Para facilitar la administración de MySQL utilizaremos el programa XAMPP que entre otros instala:

- MySQL
- PHPMyAdmin (que nos permitirá administrar las bases de datos existentes en MsSQL)
- PHP (Lenguaje que nos permite ejecutar PHPMyAdmin)
- Apache (Servidor Web que nos permite ejecutar PHPMyAdmin y PHP en un servidor)

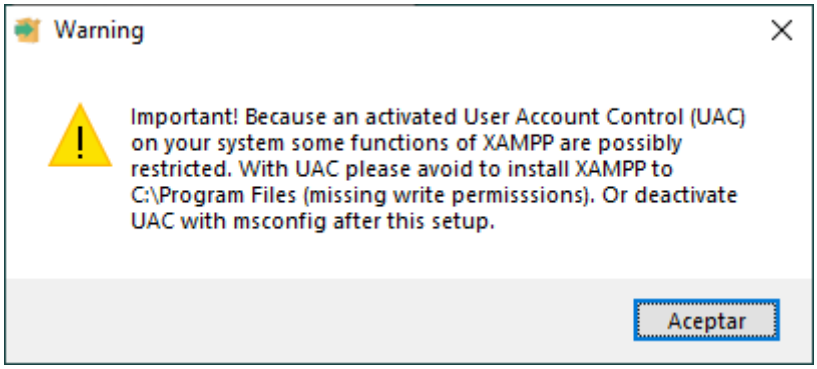
Descargaremos e instalamos:

https://www.apachefriends.org/es/download.html

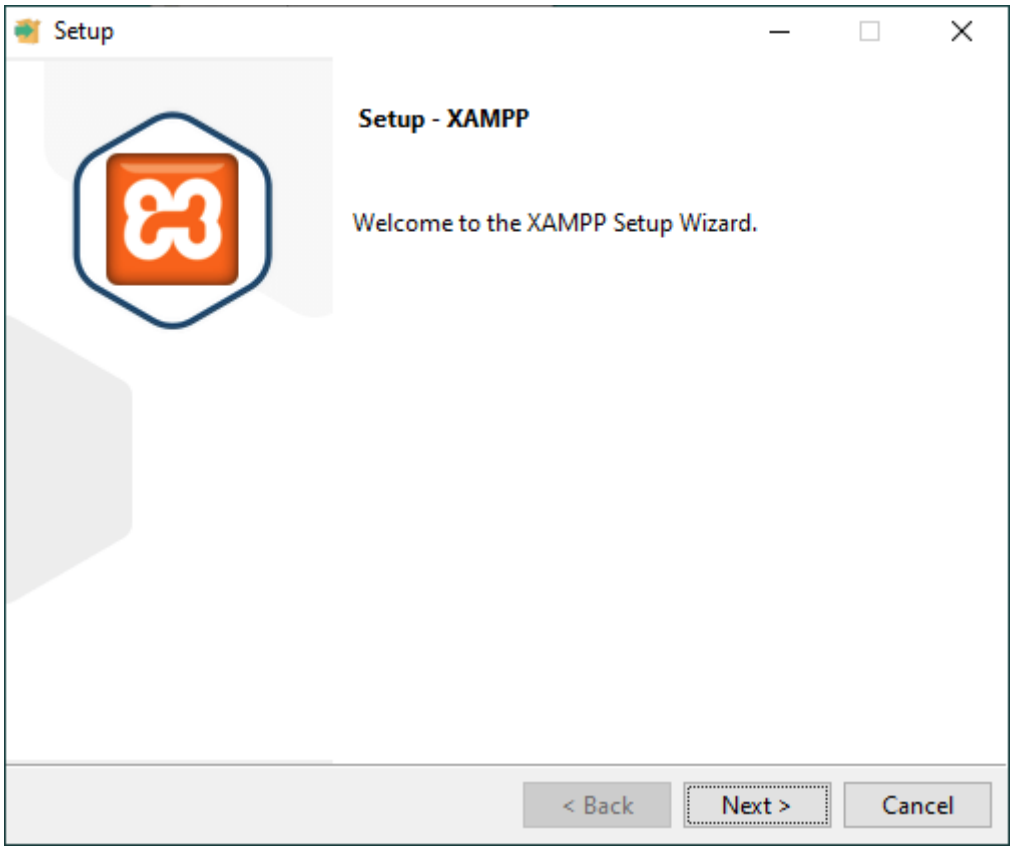
Los pasos para instalar son muy sencillos:



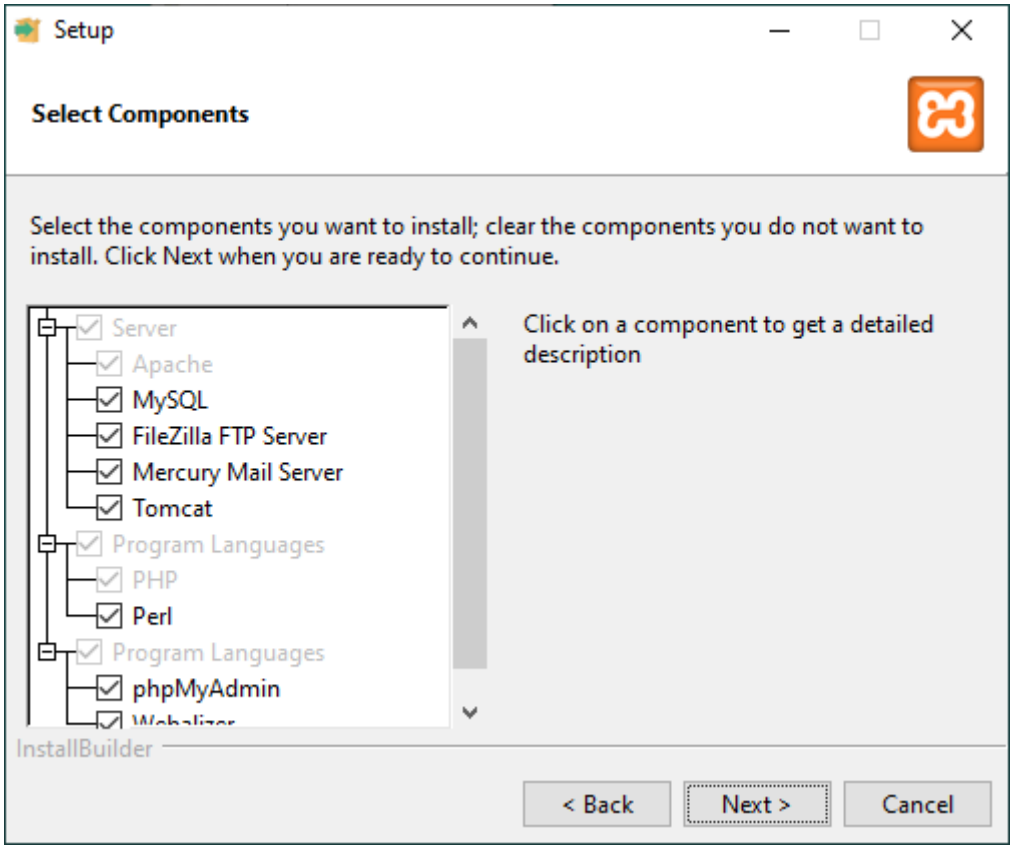
Una vez realizada la descarga procedemos a su instalación.



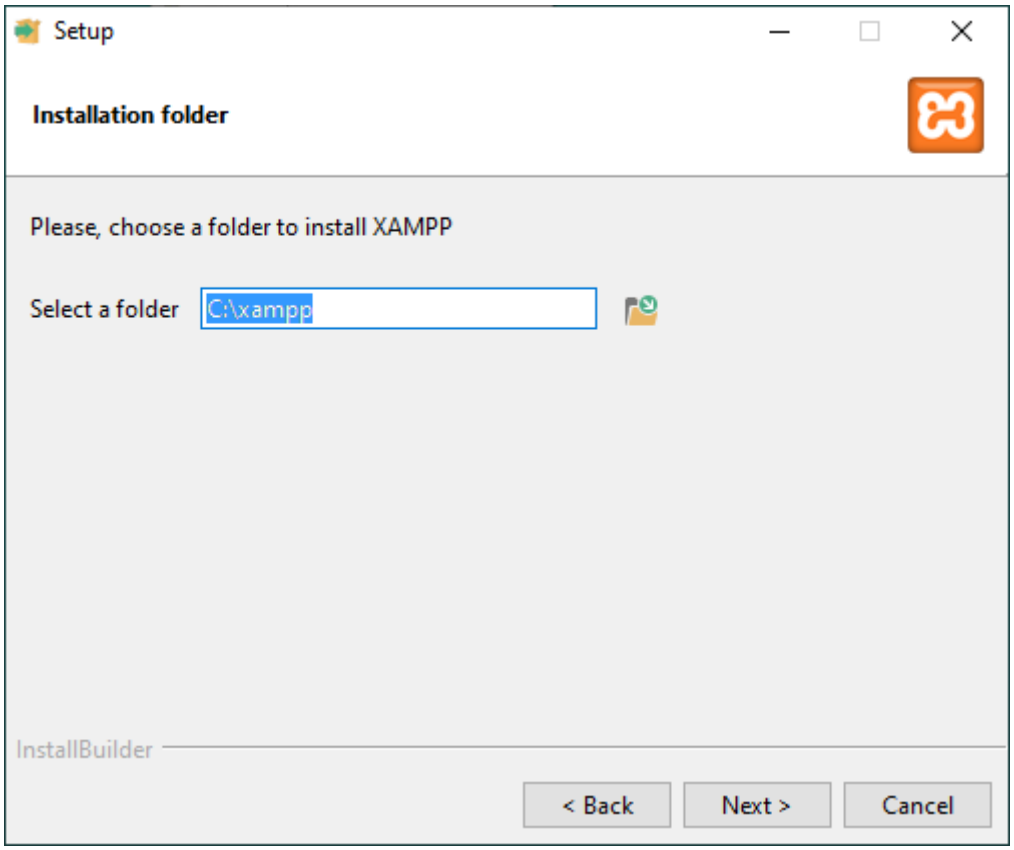
Le damos a aceptar.



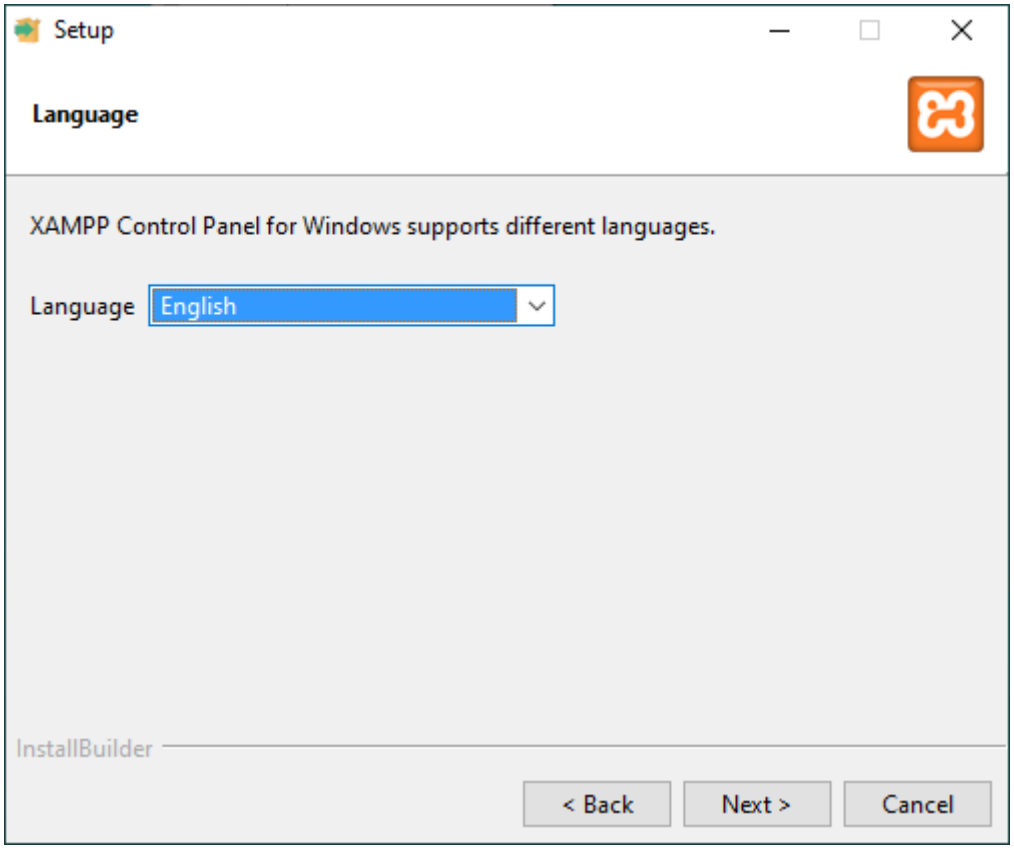
Le damos a Next.



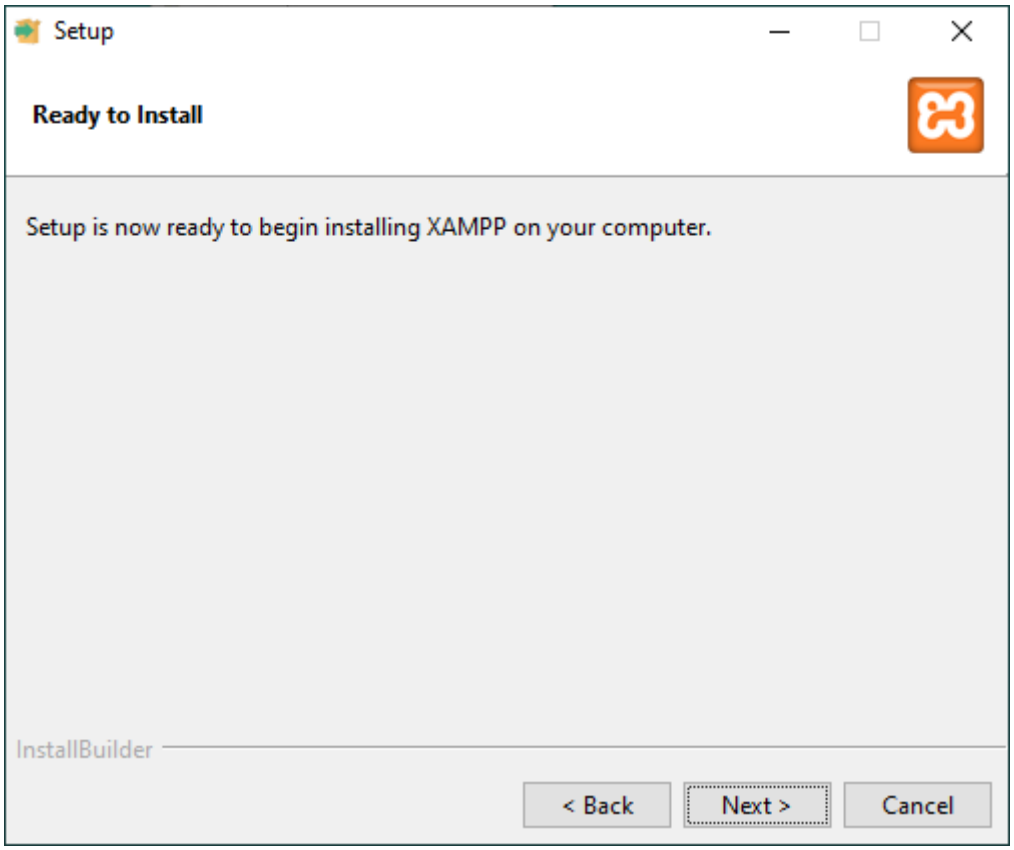
Le damos a Next.



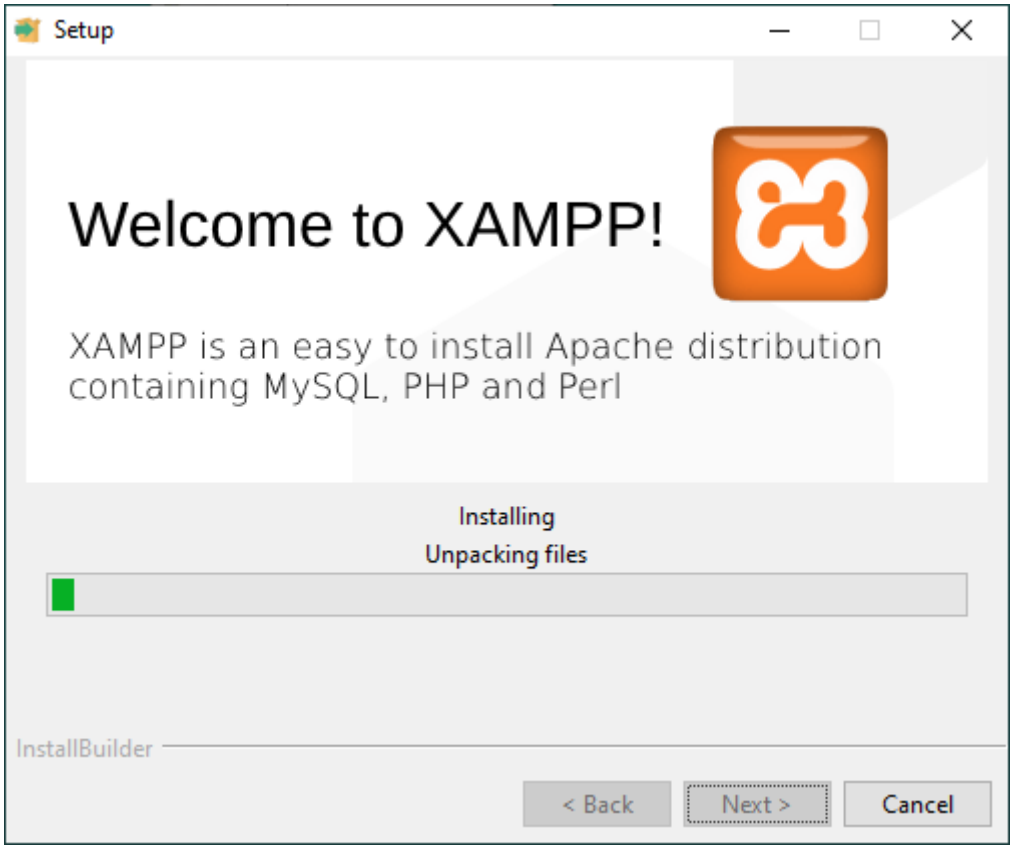
Le damos a Next.



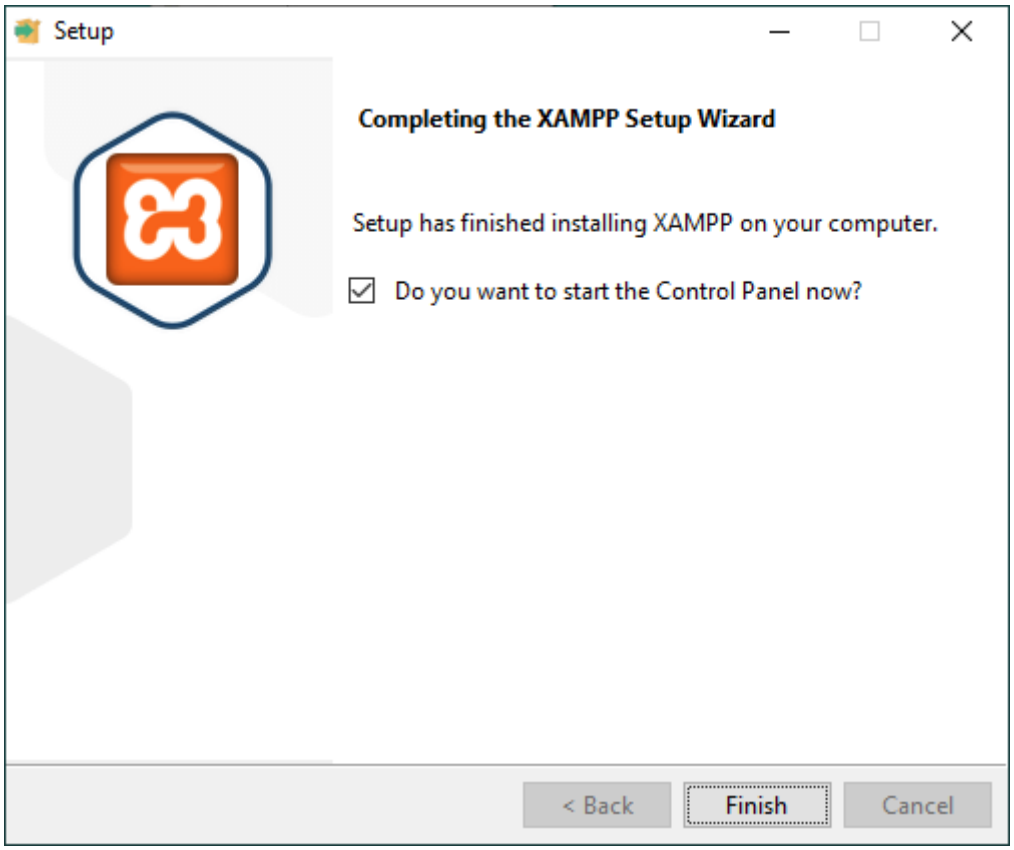
Le damos a Next.



Le damos a Next.



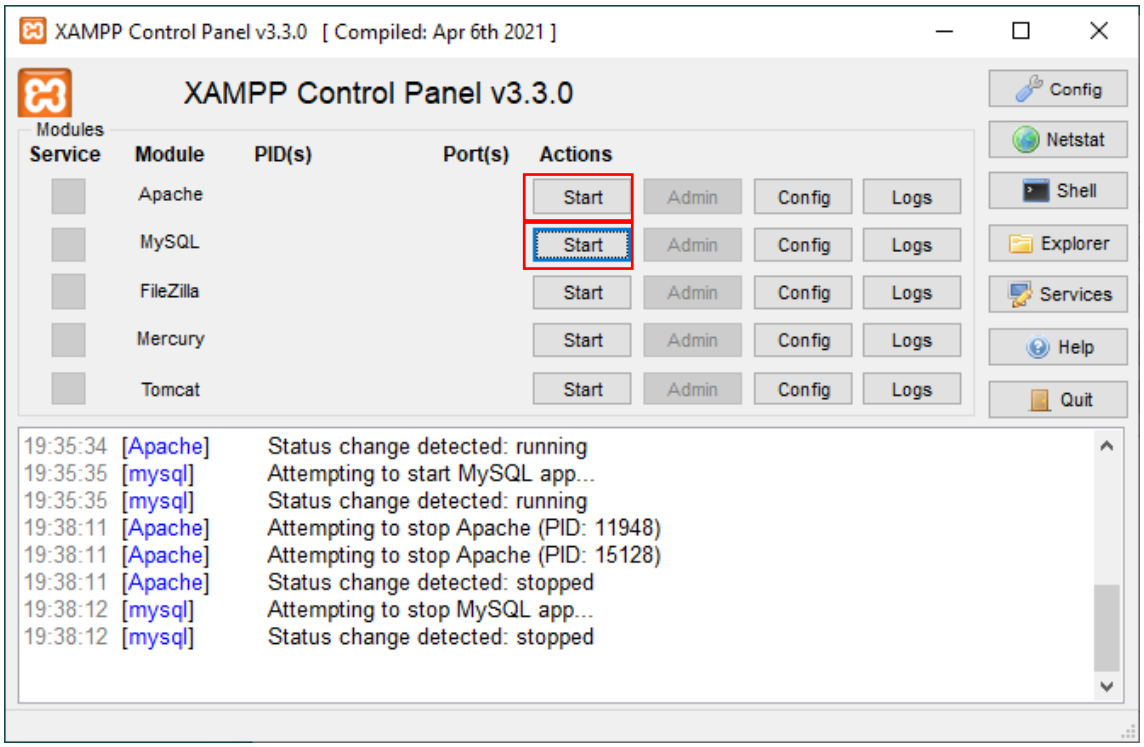
Empieza la instalación.



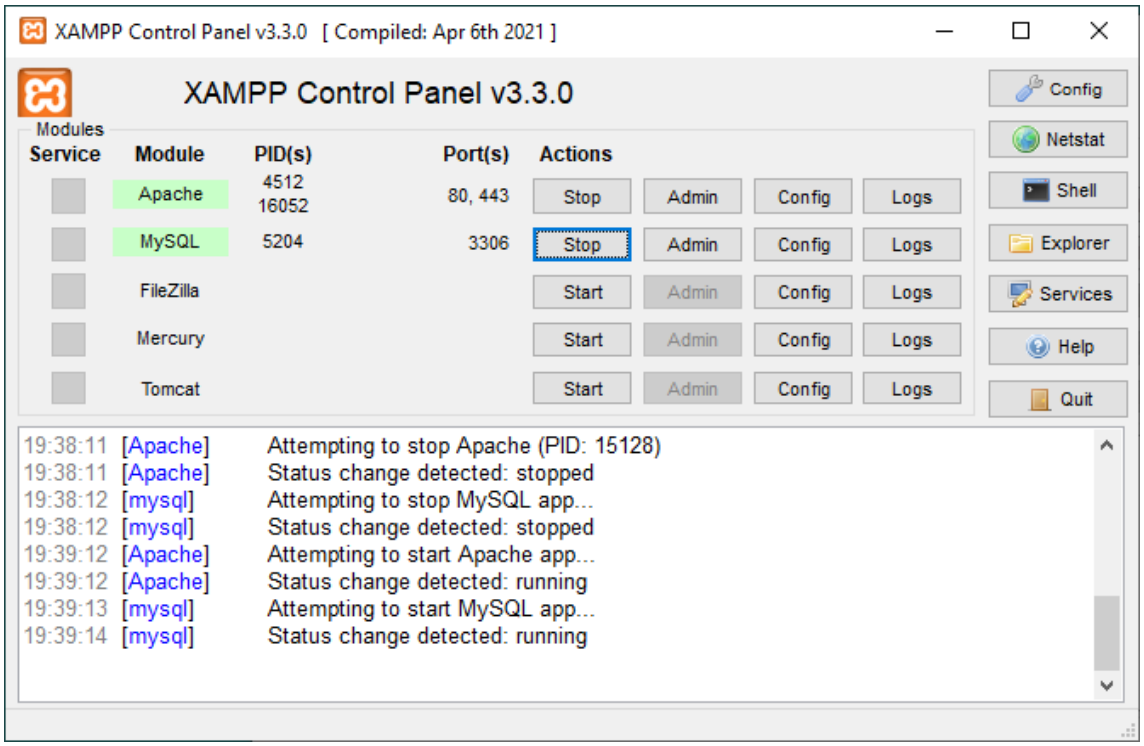
Le damos a Finish.

Una vez finalizada la instalación de todo el software debemos arrancar el programa "XAMPP".

Aparece la interfaz desde donde debemos inicializar el gestor de base de datos MySQL.

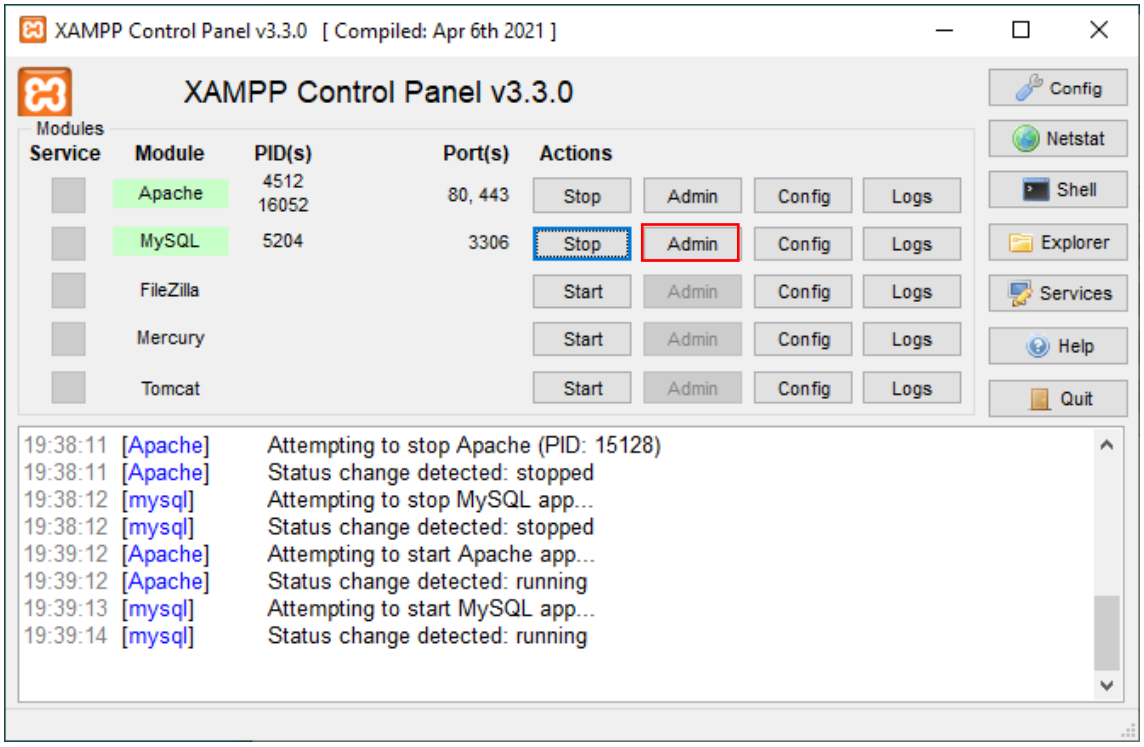


Presionamos el botón "Start" tanto para MySQL como para Apache.

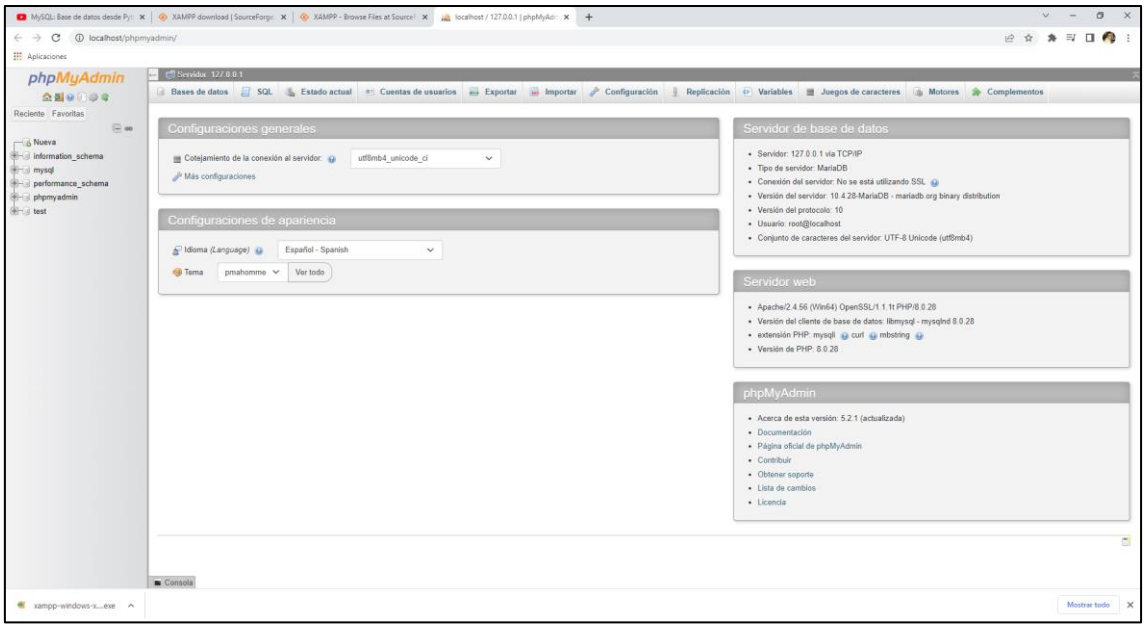


Creación de la base de datos MySQL.

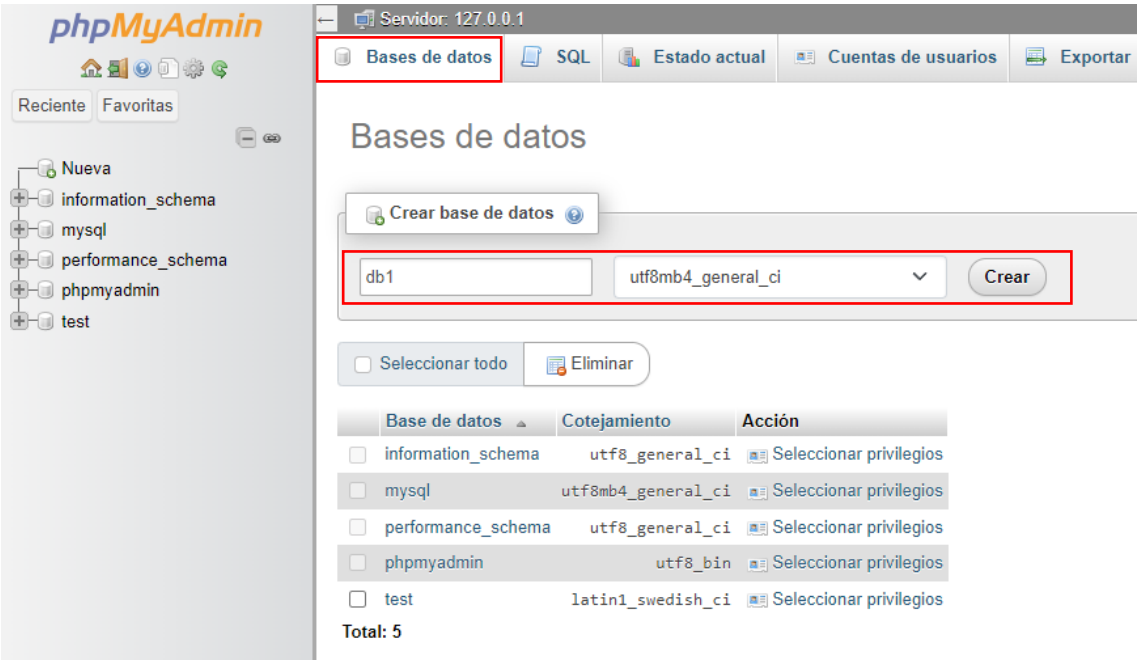
Para crear la base de datos utilizaremos el programa PHPMyAdmin que lo iniciamos presionando el botón "MySQL. – Admin" del "XAMPP Control Panel":



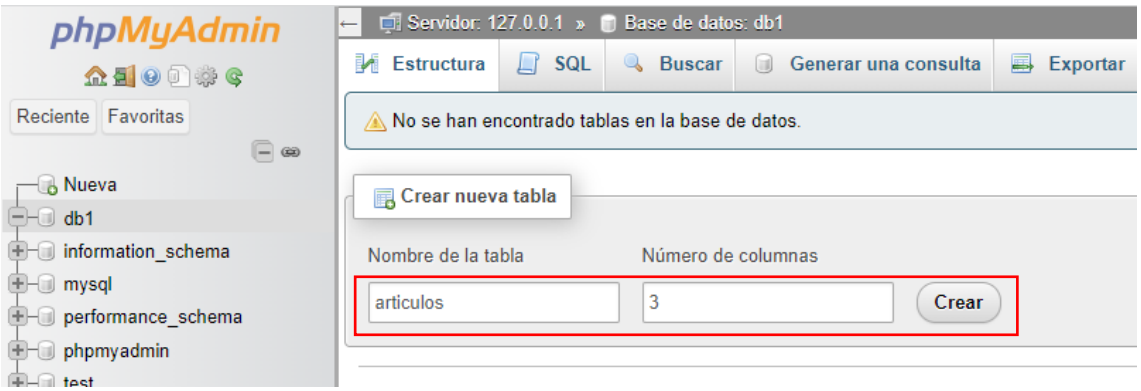
Nos muestra la siguiente página web:



Desde aquí vamos a crear una base de datos.



De la pestaña Base de datos creamos una base de datos llamada db1 seguido del botón Crear.



Como nombre le llamaremos artículos con 3 columnas seguido del botón Crear.

Nombre	Tipo	Longitud/Valores	Predeterminado	Cotejamiento	Atributos	Nulo	Índice	A_I
codigo Seleccionar desde las columnas centrales	INT		Ninguno			☐	PRIMARY PRIMARY	☑

Como nombre codigo de tipo INT y activamos la casilla A_I de autoincrementable.

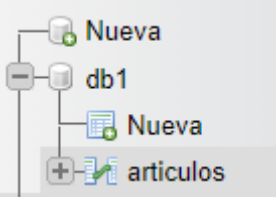
descripcion Seleccionar desde las columnas centrales	VARCHAR	50	Ninguno			☐	---	☐

El campo de descripcion de tipo VARCHAR de 50 caracteres.

precio Seleccionar desde las columnas centrales	FLOAT		Ninguno			☐	---	☐

El campo precio de tipo FLOAT.

Seguido del botón Guardar.



Ya hemos creado la base de datos db1 con la tabla artículos.

Paquete de Python necesario para conectarnos a MySQL.

Utilizaremos el programa 'pip' que vimos anteriormente para instalar el paquete necesario para interconectar 'Python' y 'MySQL'.

Desde la línea de comandos ejecutamos el programa pip con el siguiente paquete a instalar:

pip install mysql-connector

```
PROBLEMAS   SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN                2: Python

PS C:\pythonya> pip install mysql-connector
Collecting mysql-connector
  Using cached mysql-connector-2.2.9.tar.gz (11.9 MB)
Could not build wheels for mysql-connector, since package 'wheel' is not installed.
Installing collected packages: mysql-connector
    Running setup.py install for mysql-connector ... done
Successfully installed mysql-connector-2.2.9
PS C:\pythonya>
```

Conexión con el servidor de MySQL.

Controlar que el "XAMPP Control Panel" se encuentra en ejecución el servidor de MySQL:

El primer programa que implementaremos nos conectaremos con el servidor de MySQL y mostraremos todas las bases de datos existentes (una de esas debería ser db1)

Programa:

```
import mysql.connector

conexion1=mysql.connector.connect(host="localhost", user="root", passwd="")
cursor1=conexion1.cursor()
cursor1.execute("show databases")
for base in cursor1:
        print(base)
conexion1.close()
```

Vamos a escribirlo en Visual Studio Code en un nuevo proyecto.

```
ejercicio273.py > ...
1   import mysql.connector
2
3   conexion1=mysql.connector.connect(host="localhost", user="root", passwd="")
4   cursor1=conexion1.cursor()
5   cursor1.execute("show databases")
6   for base in cursor1:
7       print(base)
8   conexion1.close()
```

Ejecutamos:

```
PS D:\Curso completo de Python> & C:/Users/pmver/AppData/Local/Programs/Python/Pyth
on311/python.exe "d:/Curso completo de Python/ejercicio273.py"
('db1',) ←
('information_schema',)
('mysql',)
('performance_schema',)
('phpmyadmin',)
('test',)
PS D:\Curso completo de Python>
```

Nos muestra todas las bases de datos además de la que hemos creado nosotros.

Capítulo 273.- MySQL: Base de datos desde Python – 2

Ahora implementamos un programa que recupere todas las tablas contenidas en una base de datos. Trabajaremos con la base de datos que creamos desde phpMyAdmin llamada 'db1!.

Programa:

```
import mysql.connector

conexion1=mysql.connector.connect(host="localhost",
                                  user="root",
                                  passwd="",
                                  database="db1")
cursor1=conexion1.cursor()
cursor1.execute("show tables")
for tabla in cursor1:
        print(tabla)
conexion1.close()
```

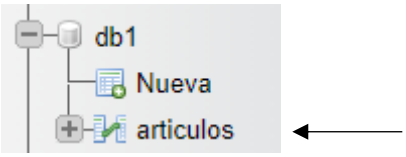
Vamos a escribirlo en Visual Studio Code en un nuevo proyecto.

```
ejercicio273.py > ...
1   import mysql.connector
2   conexion1=mysql.connector.connect(host="localhost",
3                                     user="root",
4                                     passwd="",
5                                     database="db1")
6   cursor1=conexion1.cursor()
7   cursor1.execute("show tables")
8   for tabla in cursor1:
9       print(tabla)
10  conexion1.close()
```

Ejecutamos:

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL        Python

PS D:\Curso completo de Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/
python.exe "d:/Curso completo de Python/ejercicio273.py"
('articulos',)
PS D:\Curso completo de Python>
```

Tenemos la tabla artículos que creamos con anterioridad.



Ahora queremos consultar las tablas que tiene la base de datos mysql

ejercicio273.py > ...

```python
import mysql.connector
conexion1=mysql.connector.connect(host="localhost",
                                  user="root",
                                  passwd="",
                                  database="mysql")
cursor1=conexion1.cursor()
cursor1.execute("show tables")
for tabla in cursor1:
    print(tabla)
conexion1.close()
```

Vamos a ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python

```
PS D:\Curso completo de Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/
python.exe "d:/Curso completo de Python/ejercicio273.py"
('column_stats',)
('columns_priv',)
('db',)
('event',)
('func',)
('general_log',)
('global_priv',)
('gtid_slave_pos',)
('help_category',)
('help_keyword',)
('help_relation',)
('help_topic',)
('index_stats',)
('innodb_index_stats',)
('innodb_table_stats',)
('plugin',)
('proc',)
('procs_priv',)
('proxies_priv',)
('roles_mapping',)
('servers',)
('slow_log',)
('table_stats',)
('tables_priv',)
('time_zone',)
('time_zone_leap_second',)
('time_zone_name',)
('time_zone_transition',)
('time_zone_transition_type',)
('transaction_registry',)
('user',)
```

Así podemos recuperar en el resto de las base de datos.

Capítulo 274.- MySQL: Base de datos desde Python – 3

Insertar filas en una tabla

Programa:

```
import mysql.connector

conexion1=mysql.connector.connect(host="localhost",
                                  user="root",
                                  passwd="",
                                  database="db1")
cursor1=conexion1.cursor()
sql="insert into articulos(descripcion, precio) values (%s, %s)"
datos=("naranjas", 23.50)
cursor1.execute(sql, datos)
datos=("peras", 34)
cursor1.execute(sql, datos)
datos=("bananas", 25)
cursor1.execute(sql, datos)
conexion1.commit()
for tabla in cursor1:
        print(tabla)
conexion1.close()
```

Definimos un string con el comando SQL insert disponiendo las máscaras %s donde queremos que se sustituya por un valor que le pasaremos al método execute:

sql="insert into artículos(descripcion, precio) values (%s,%s)"

La variable datos se una tupla que contiene los datos que se utilizarán en la sustitución %s:

datos=("narnaja", 32.50)

Finalmente llamamos al método 'execute' y le pasamos las dos variables que acabamos de crear:

cursor1.execute(sql, datos)

Es fundamental llamar al final al método 'commit' para que queden firmes los comandos SQL.

'insert':

conesion1.commit()

Lo vamos a escribir en VS Code.

ejercicio274.py > ...

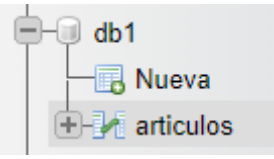
```
1  import mysql.connector
2  conexion1=mysql.connector.connect(host="localhost",
3                                    user="root",
4                                    passwd="",
5                                    database="db1")
```

```
6   cursor1=conexion1.cursor()
7   sql="insert into articulos(descripcion, precio) values (%s, %s)"
8   datos=("naranjas", 23.50)
9   cursor1.execute(sql, datos)
10  datos=("peras", 34)
11  cursor1.execute(sql, datos)
12  datos=("bananas", 25)
13  cursor1.execute(sql, datos)
14  conexion1.commit()
15  for tabla in cursor1:
16      print(tabla)
17  conexion1.close()
```

A continuación lo vamos a ejecutar:

En la consola no se muestra nada.

Entramos a PHPAdmin.



Seleccionamos artículos:

			codigo	descripcion	precio
Editar	Copiar	Borrar	1	naranjas	23.5
Editar	Copiar	Borrar	2	peras	34
Editar	Copiar	Borrar	3	bananas	25

Encontramos los tres registros que hemos insertado.

Capítulo 275.- MySQL: Base de datos desde Python – 4

Recuperar todas las filas de una tabla.

Implementaremos un programa que solicite ejecutar un 'select' en la tabla 'articulos' y nos retorne sus filas-

Programa:

```
import mysql.connector

conexion1=mysql.connector.connect(host="localhost",
                                  user="root",
                                  passwd="",
                                  database="db1")
cursor1=conexion1.cursor()
cursor1.execute("select codigo, descripcion, precio from articulos")
for fila in cursor1:
    print(fila)
conexion1.close()
```

Cuando ejecutamos el programa podemos ver que se recupera todas las filas de la tabla 'articulos':

ejercicio275.py > ...

```
1   import mysql.connector
2   conexion1=mysql.connector.connect(host="localhost",
3                                     user="root",
4                                     passwd="",
5                                     database="db1")
6   cursor1=conexion1.cursor()
7   cursor1.execute("select codigo, descripcion, precio from articulos")
8   for fila in cursor1:
9       print(fila)
10  conexion1.close()
11
12
13
14
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python

```
PS D:\Curso completo de Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/
python.exe "d:/Curso completo de Python/ejercicio275.py"
(1, 'naranjas', 23.5)
(2, 'peras', 34.0)
(3, 'bananas', 25.0)
PS D:\Curso completo de Python>
```

Capítulo 276.- MySQL: Base de datos desde Python – 5

Borrado y modificado de filas.

Las otras dos actividades fundamentales que podemos hacer con una tabla es borrar filas y modificar datos.

Desarrollaremos un pequeño programa que borre el artículo cuyo código sea el 1 y modifique el precio del artículo cuyo código sea 3.

Programa:

```
import mysql.connector

conexion1=mysql.connector.connect(host="localhost",
                                  user="root",
                                  passwd="",
                                  database="db1")
cursor1=conexion1.cursor()
cursor1.execute("select codigo, descripcion, precio from articulos")
for fila in cursor1:
    print(fila)
print("-----------------------------------------------------------------------------------")

cursor1.execute("delete from articulos where codigo=1")
cursor1.execute("update articulos set precio=50 where codigo=3")
conexion1.commit()

cursor1.execute("select codigo, descripcion, precio from articulos")
for fila in cursor1:
    print(fila)
conexion1.close()
```

Luego de crear el cursor podemos llamar al método 'execute' varias veces y pasar distintos comandos SQL:

```
cursor1=conexion1.cursor()
cursor1.execute("delete from articulos where codigo=1")
cursor1.execute("update articulos set precio=50 where codigo=3")
```

Siempre que pasemos un commando SQL: insert, delete o update debemos llamar al método commit para que quede firme los cambios en la base de datos:

```
conexion1.commit()
```

Ejecutamos finalmente un 'select' para comprobar los cambios efectuados en la tabla 'articulos':

```
cursor1.execute("select codigo, descripcion, precio from articulos")
for fila in cursor1:
    print(fila)
```

Vamos a escribir el Código en VS Code y a continuación lo vamos a ejecutar:

ejercicio276.py > ...

```python
import mysql.connector
conexion1=mysql.connector.connect(host="localhost",
                                  user="root",
                                  passwd="",
                                  database="db1")
cursor1=conexion1.cursor()
cursor1.execute("select codigo, descripcion, precio from articulos")
for fila in cursor1:
    print(fila)
print("------------------------------------------------------------------------
cursor1.execute("delete from articulos where codigo=1")
cursor1.execute("update articulos set precio=50 where codigo=3")
conexion1.commit()
cursor1.execute("select codigo, descripcion, precio from articulos")
for fila in cursor1:
    print(fila)
conexion1.close()
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python

```
PS D:\Curso completo de Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/
python.exe "d:/Curso completo de Python/ejercicio276.py"
(1, 'naranjas', 23.5)
(2, 'peras', 34.0)
(3, 'bananas', 25.0)
------------------------------------------------------------------------
(2, 'peras', 34.0)
(3, 'bananas', 50.0)
PS D:\Curso completo de Python>
```

Antes y después de los cambios

Capítulo 277.- MySQL: Interfaz visual con tkinter y acceso a la base de datos – 1

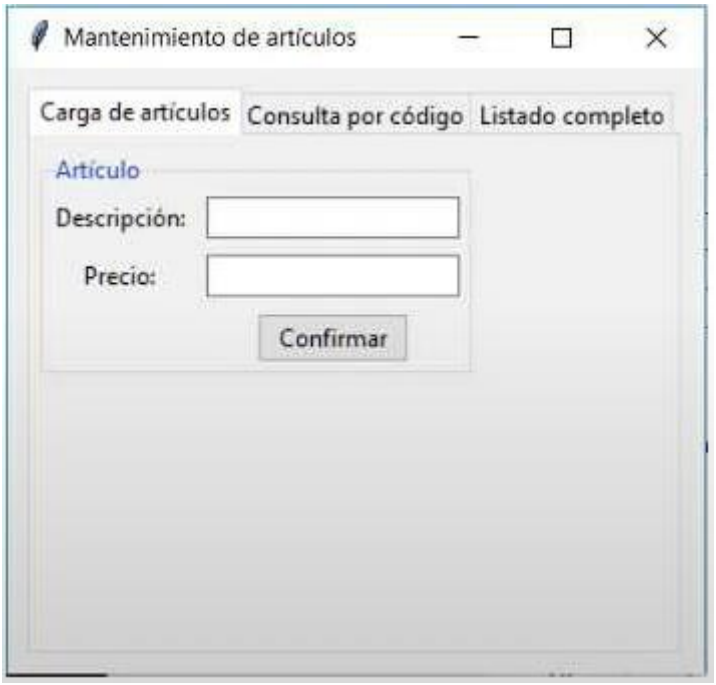
Es muy posible que necesitemos en algunas situaciones acceder a una base de datos de MySQL desde una aplicación con una interfaz visual.

Problema:

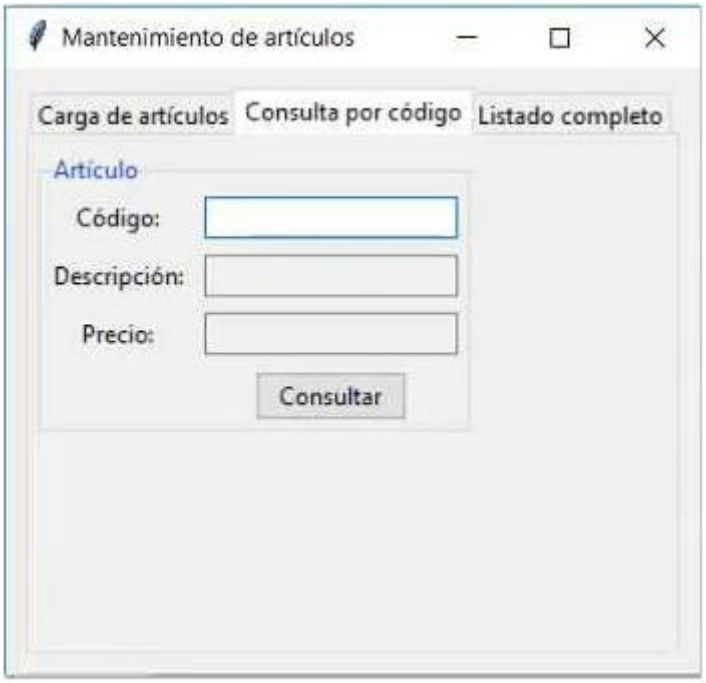
Desarrollar una aplicación visual con la librería tkinter que permita implementar los algoritmos de carta de artículos, consulta por código y listado completo.

Seguiremos trabajando con la tabla 'articulos' que creamos en el concepto anterior.

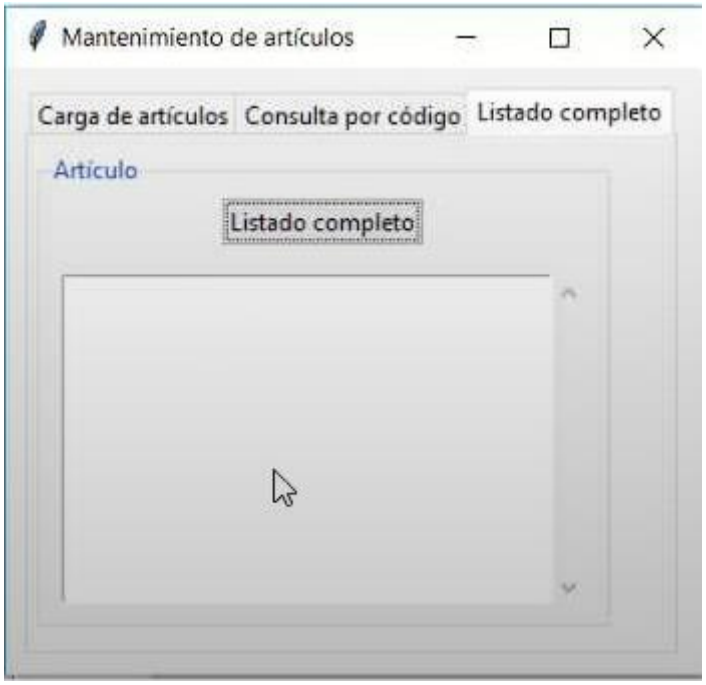
La interfaz visual para la carga debe ser:



La interfaz visual para la consulta:



Y finalmente la interfaz para el listado completo:



Para trabajar un poco más ordenado en la resolución de este problema lo dividiremos en dos módulos 'formularioarticulos.py' y 'articulos.py'.

El primer paso será crear una carpeta llamada 'proyecto3' y dentro de esta crearemos los dos módulos.

El módulo 'formularioarticulos.py' contiene toda la lógica de presentación de datos y hace uso del otro módulo para el acceso a la base de datos MySQL:

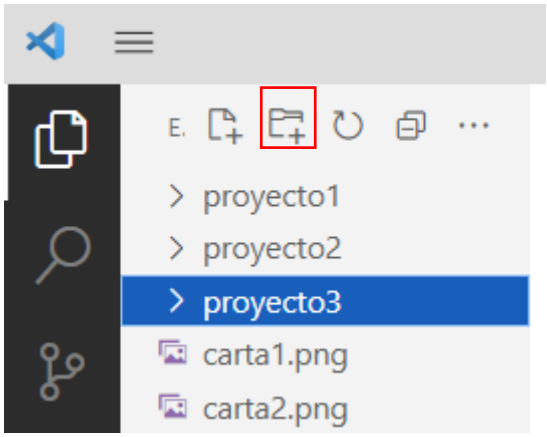
módulo: formularioarticulos.py

El módulo 'articulos' contiene toda la lógica de acceso a datos:

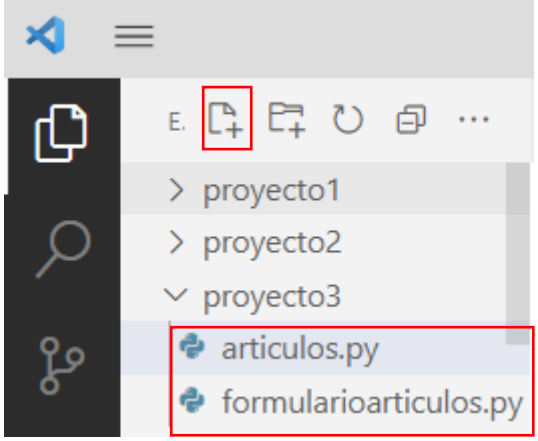
módulo: articulos.py

Tener en cuenta que el módulo principal se encuentra en el archivo 'formularioarticulos.py' y es el que debemos ejecutar.

Vamos a programar:
Vamos a crear una carpeta llamada 'proyecto3'



Dentro de esta carpeta crearemos los proyectos 'formularioarticulos.py' y 'articulos.py'.



Vamos a escribir el código del programa 'articulos.py':

```
import mysql.connector

class Articulos:

    def abrir(self):
        conexion=mysql.connector.connect(host="localhost",
                                         user="root",
                                         passwd="",
                                         database="db1")
        return conexion

    def alta(self,datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="insert into articulos(descripcion, precio) values (%s,%s)"
        cursor.execute(sql, datos)
        cone.commit()
        cone.close()

    def consulta(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="select descripcion, precio from articulos where codigo=%s"
        cursor.execute(sql, datos)
        cone.close()
        # fetchall() Recupera todas las filas del resultado
        # de una consulta como una lista de tuplas.
        return cursor.fetchall()

    def recuperar_todos(self):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="select codigo, descripcion, precio from articulos"
```

```
        cursor.execute(sql)
        cone.close()
        # fetchall() Recupera todas las filas del resultado
        # de una consulta como una lista de tuplas.
        return cursor.fetchall()
```

Ahora vamos a programar 'formularioarticulos.py'.

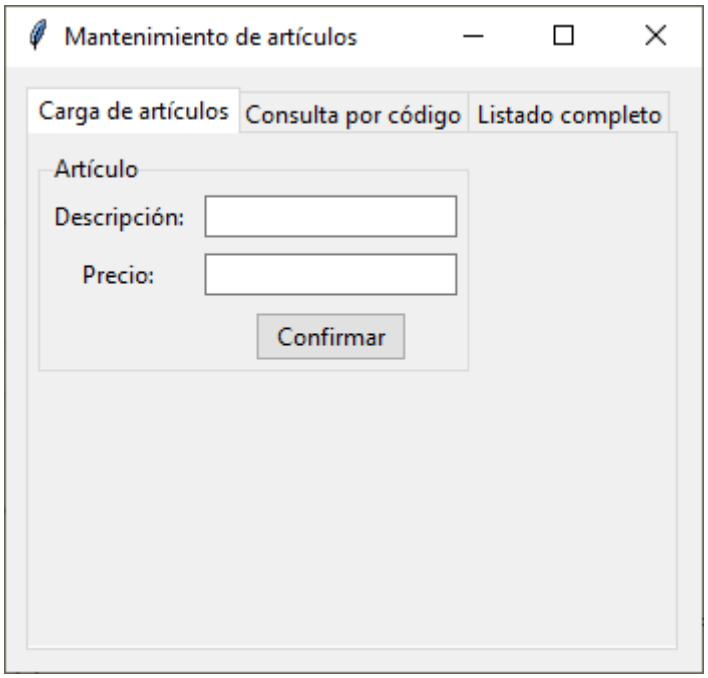
```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mb
from tkinter import scrolledtext as st
# importamos la clase Articulos del fichero articulos.py
import articulos

class FormularioArticulos:
    def __init__(self):
        self.articulo1=articulos.Articulos()
        self.ventana1=tk.Tk()
        self.ventana1.title("Mantenimiento de artículos")
        self.cuaderno1=ttk.Notebook(self.ventana1)
        self.carga_articulos()
        self.consulta_por_codigo()
        self.listado_completo()
        self.cuaderno1.grid(column=0, row=0, padx=10, pady=10)
        self.ventana1.mainloop()

    def carga_articulos(self):
        self.pagina1=ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina1, text="Carga de artículos")
        self.labelframe1=ttk.LabelFrame(self.pagina1, text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe1, text="Descripción:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.descripcioncarga=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe1,
textvariable=self.descripcioncarga)
        self.entrydescripcion.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1, text="Precio:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.preciocarga=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe1,
textvariable=self.preciocarga)
        self.entryprecio.grid(column=1, row=1, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Confirmar",
command=self.agregar)
        self.boton1.grid(column=1, row=2, padx=4, pady=4)
```

```
    def consulta_por_codigo(self):
        self.pagina2=ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina2, text="Consulta por código")
        self.labelframe2=ttk.LabelFrame(self.pagina2, text="Artículo")
        self.labelframe2.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe2, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigo=tk.StringVar()
        self.entrycodigo=ttk.Entry(self.labelframe2,
textvariable=self.codigo)
        self.entrycodigo.grid(column=1, row=0, padx=4, pady=5)
        self.label2=ttk.Label(self.labelframe2, text="Código:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.descripcion=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe2,
textvariable=self.descripcion, state="readonly")
        self.entrydescripcion.grid(column=1, row=1, padx=4, pady=5)
        self.label3=ttk.Label(self.labelframe2, text="Código:")
        self.label3.grid(column=0, row=2, padx=4, pady=4)
        self.precio=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe2,
textvariable=self.precio, state="readonly")
        self.entryprecio.grid(column=1, row=2, padx=4, pady=5)
        self.boton1=ttk.Button(self.labelframe2, text="Consultar",
command=self.consultar)
        self.boton1.grid(column=1, row=3, padx=4, pady=4)

    def listado_completo(self):
        self.pagina3=ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina3, text="Listado completo")
        self.labelframe3=ttk.LabelFrame(self.pagina3, text="Artículo")
        self.labelframe3.grid(column=0, row=0, padx=5, pady=10)
        self.boton1=ttk.Button(self.labelframe3, text="Listado completo",
command=self.listar)
        self.boton1.grid(column=0, row=0, padx=4, pady=4)
        self.scrolledtext1=st.ScrolledText(self.labelframe3, width=30,
height=10)
        self.scrolledtext1.grid(column=0, row=1, padx=10, pady=10)

    def agregar(self):
        datos=(self.descripcioncarga.get(), self.preciocarga.get())
        self.articulo1.alta(datos)
        mb.showinfo("Información", "Los datos fueron cargados")
        self.descripcioncarga.set("")
        self.preciocarga.set("")

    def consultar(self):
        datos=(self.codigo.get(),)
        respuesta=self.articulo1.consulta(datos)
```

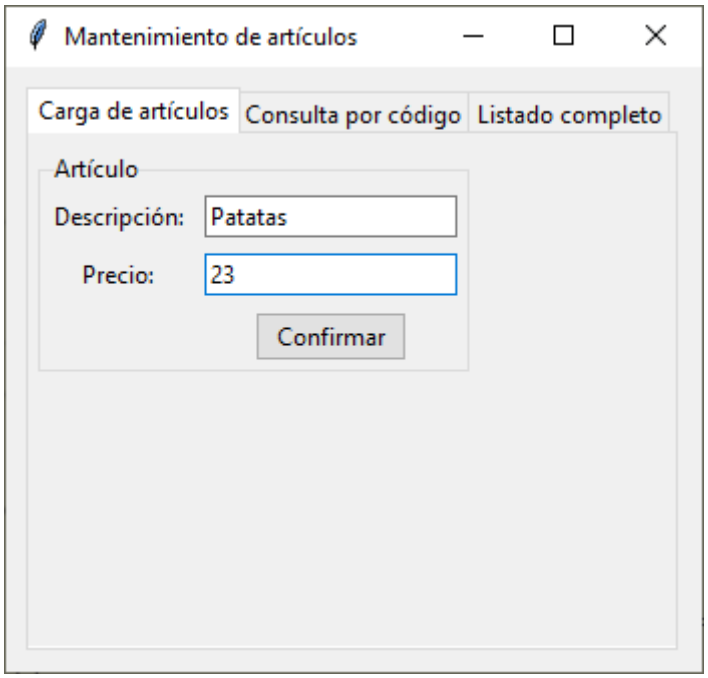
```
        if len(respuesta)>0:
            self.descripcion.set(respuesta[0][0])
            self.precio.set(respuesta[0][1])
        else:
            self.descripcion.set("")
            self.precio.set("")
            mb.showinfo("Información", "No existe un artículo con dicho
código")

    def listar(self):
        respuesta=self.articulo1.recuperar_todos()
        self.scrolledtext1.delete("1.0", tk.END)
        for fila in respuesta:
            self.scrolledtext1.insert(tk.END,
"Código:"+str(fila[0])+"\nDescripción:"+fila[1]+"\nPrecio:"+str(fila[2])+
"\n\n")

# Bloque principal
formularioarticulos=FormularioArticulos()
```

Vamos a ejecutar:

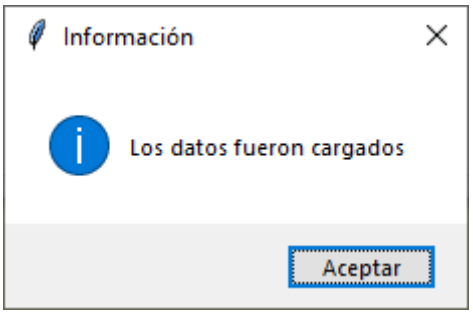
Recuerda ejecutar XAMMP Control Panel antes de ejecutar tu programa.



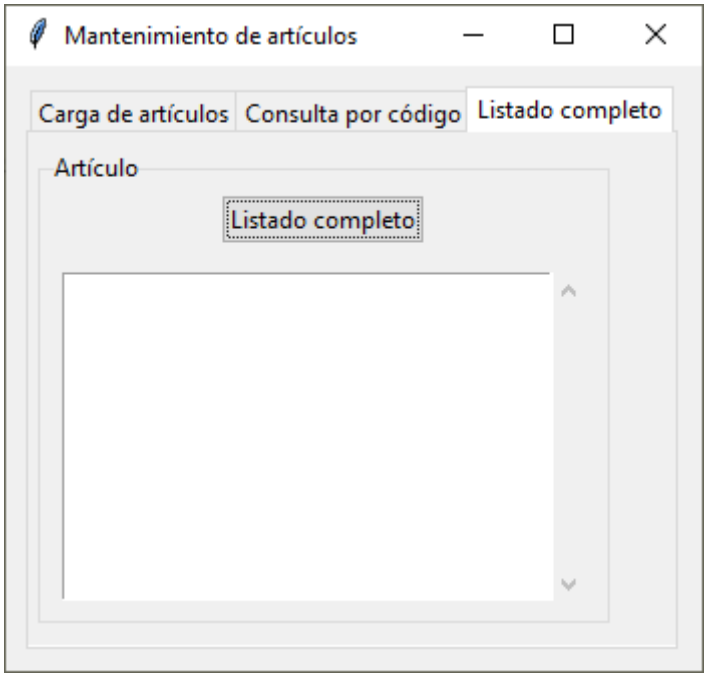
Vamos a añadir un artículo:



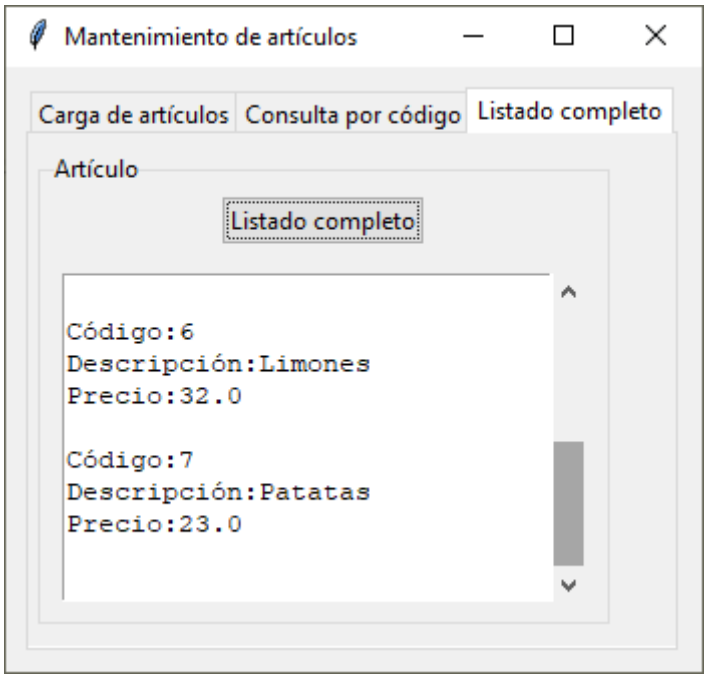
Le damos a confirmar.



Le damos a aceptar.
Vamos a la pestaña listado completo.

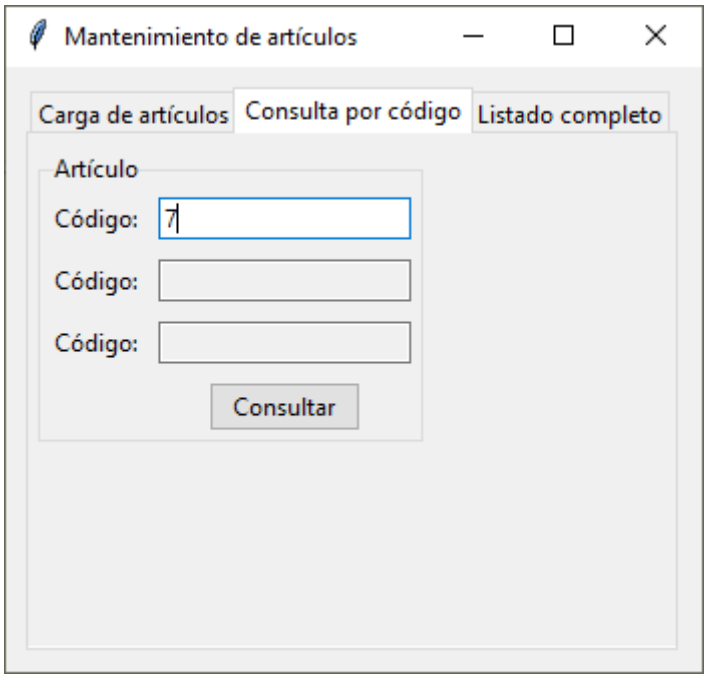


Seleccionamos el botón Listado completo.

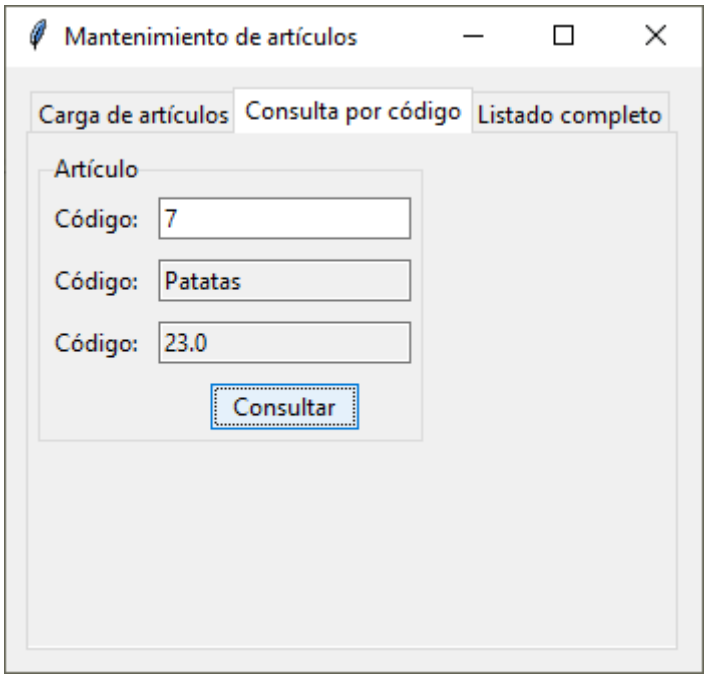


Vamos a memorizar el código del artículo que queremos consultar, en mi caso elegiré el 7 que son Patatas a un precio de 23.

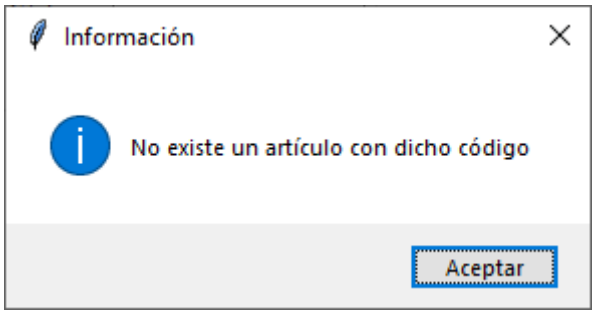
Nos vamos a la pestaña Consulta por código:



Introduzco el código del artículo que deseo consultar y a continuación presionamos el botón Consultar.



Ahora voy a introducir un codigo inexistente:



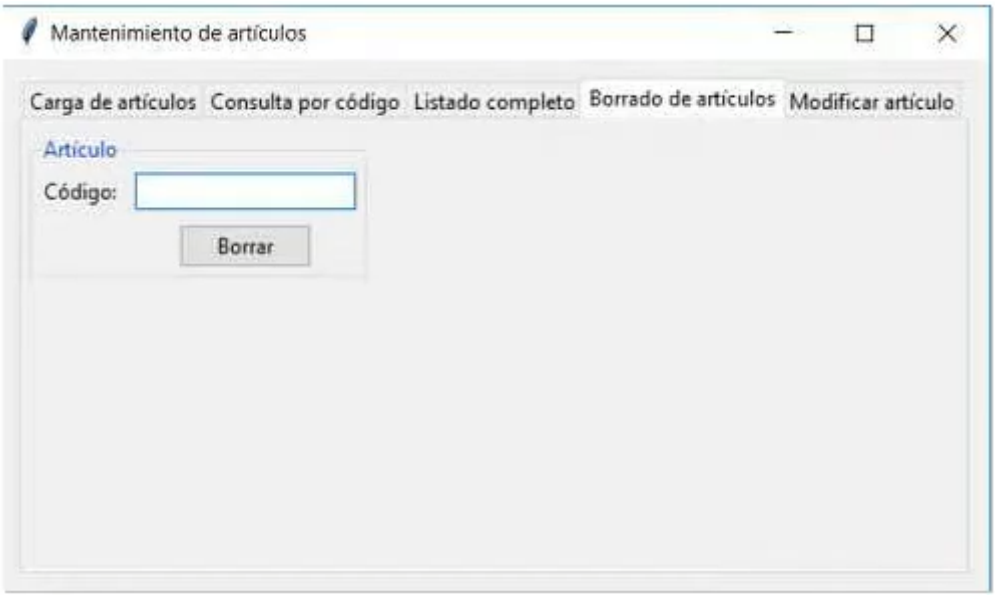
Me informa que no existe un artículo con dicho código.

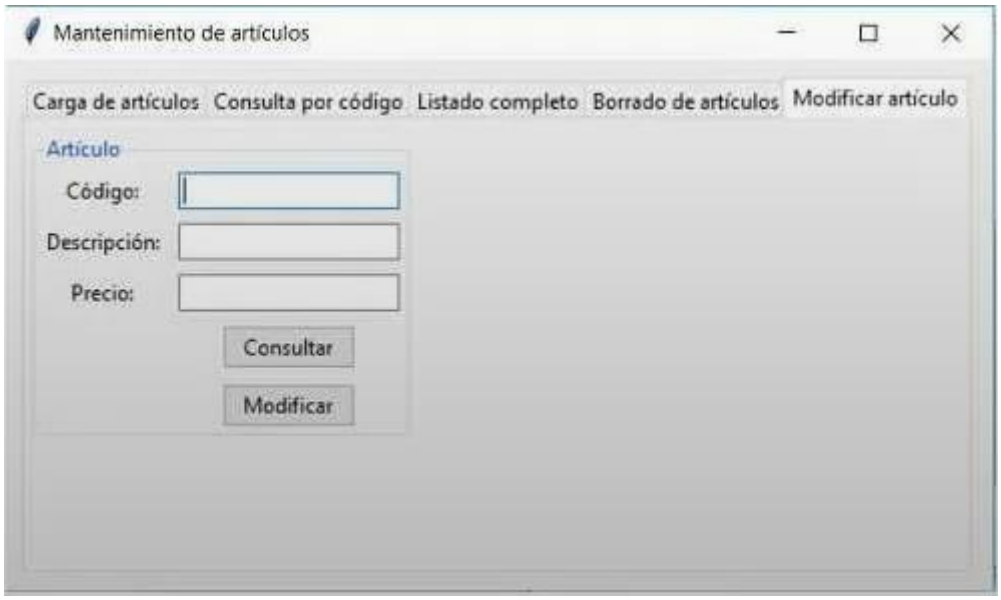
Capítulo 278.- MySQL: Interfaz visual con tkinter y acceso a la base de datos – 2

Problema propuesto

Agregar dos pestañas al programa de administración de artículos que permitan borrar un artículo ingresando su código y otra opción que permita consultar y modificar la descripción y precio de un artículo.

Las interfaces visuales a implementar son:





Vamos a programar:

El código "artículos.py"

```
import mysql.connector

class Articulos:

    def abrir(self):
        conexion=mysql.connector.connect(host="localhost",
                                         user="root",
                                         passwd="",
                                         database="db1")
        return conexion

    def alta(self,datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="insert into articulos(descripcion, precio) values (%s,%s)"
        cursor.execute(sql, datos)
        cone.commit()
        cone.close()

    def consulta(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="select descripcion, precio from articulos where codigo=%s"
        cursor.execute(sql, datos)
        cone.close()
        # fetchall() Recupera todas las filas del resultado
        # de una consulta como una lista de tuplas.
        return cursor.fetchall()

    def recuperar_todos(self):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="select codigo, descripcion, precio from articulos"
        cursor.execute(sql)
        cone.close()
        # fetchall() Recupera todas las filas del resultado
        # de una consulta como una lista de tuplas.
        return cursor.fetchall()

    def baja(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="delete from articulos where codigo=%s"
        cursor.execute(sql, datos)
        cone.commit()
        cone.close()
        return cursor.rowcount # retornamos la cantidad de filas borradas
```

```
    def modificacion(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="update articulos set descripcion=%s, precio=%s where
codigo=%s"
        cursor.execute(sql, datos)
        cone.commit()
        cone.close()
        return cursor.rowcount # retornamos la cantidad de filas
modificadas
```

Código "formulariosarticulos.py"

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mb
from tkinter import scrolledtext as st
# importamos la clase Articulos del fichero articulos.py
import articulos

class FormularioArticulos:
    def __init__(self):
        self.articulo1=articulos.Articulos()
        self.ventana1=tk.Tk()
        self.ventana1.title("Mantenimiento de artículos")
        self.cuaderno1=ttk.Notebook(self.ventana1)
        self.carga_articulos()
        self.consulta_por_codigo()
        self.listado_completo()
        self.borrado()
        self.modificar()
        self.cuaderno1.grid(column=0, row=0, padx=10, pady=10)
        self.ventana1.mainloop()

    def carga_articulos(self):
        self.pagina1=ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina1, text="Carga de artículos")
        self.labelframe1=ttk.LabelFrame(self.pagina1, text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe1, text="Descripción:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.descripcioncarga=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe1,
textvariable=self.descripcioncarga)
        self.entrydescripcion.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1, text="Precio:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
```

```
        self.preciocarga=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe1,
textvariable=self.preciocarga)
        self.entryprecio.grid(column=1, row=1, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Confirmar",
command=self.agregar)
        self.boton1.grid(column=1, row=2, padx=4, pady=4)

    def consulta_por_codigo(self):
        self.pagina2=ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina2, text="Consulta por código")
        self.labelframe2=ttk.LabelFrame(self.pagina2, text="Artículo")
        self.labelframe2.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe2, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigo=tk.StringVar()
        self.entrycodigo=ttk.Entry(self.labelframe2,
textvariable=self.codigo)
        self.entrycodigo.grid(column=1, row=0, padx=4, pady=5)
        self.label2=ttk.Label(self.labelframe2, text="Código:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.descripcion=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe2,
textvariable=self.descripcion, state="readonly")
        self.entrydescripcion.grid(column=1, row=1, padx=4, pady=5)
        self.label3=ttk.Label(self.labelframe2, text="Código:")
        self.label3.grid(column=0, row=2, padx=4, pady=4)
        self.precio=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe2,
textvariable=self.precio, state="readonly")
        self.entryprecio.grid(column=1, row=2, padx=4, pady=5)
        self.boton1=ttk.Button(self.labelframe2, text="Consultar",
command=self.consultar)
        self.boton1.grid(column=1, row=3, padx=4, pady=4)

    def listado_completo(self):
        self.pagina3=ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina3, text="Listado completo")
        self.labelframe3=ttk.LabelFrame(self.pagina3, text="Artículo")
        self.labelframe3.grid(column=0, row=0, padx=5, pady=10)
        self.boton1=ttk.Button(self.labelframe3, text="Listado completo",
command=self.listar)
        self.boton1.grid(column=0, row=0, padx=4, pady=4)
        self.scrolledtext1=st.ScrolledText(self.labelframe3, width=30,
height=10)
        self.scrolledtext1.grid(column=0, row=1, padx=10, pady=10)
```

```
    def agregar(self):
        datos=(self.descripcioncarga.get(), self.preciocarga.get())
        self.articulo1.alta(datos)
        mb.showinfo("Información", "Los datos fueron cargados")
        self.descripcioncarga.set("")
        self.preciocarga.set("")

    def consultar(self):
        datos=(self.codigo.get(),)
        respuesta=self.articulo1.consulta(datos)
        if len(respuesta)>0:
            self.descripcion.set(respuesta[0][0])
            self.precio.set(respuesta[0][1])
        else:
            self.descripcion.set("")
            self.precio.set("")
            mb.showinfo("Información", "No existe un artículo con dicho
código")

    def listar(self):
        respuesta=self.articulo1.recuperar_todos()
        self.scrolledtext1.delete("1.0", tk.END)
        for fila in respuesta:
            self.scrolledtext1.insert(tk.END,
"Código:"+str(fila[0])+"\nDescripción:"+fila[1]+"\nPrecio:"+str(fila[2])+
"\n\n")
```

```
    def borrado(self):
        self.pagina4 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina4, text="Borrado de artículos")
        self.labelframe4=ttk.LabelFrame(self.pagina4,
text="Artículo")
        self.labelframe4.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe4, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigoborra=tk.StringVar()
        self.entryborra=ttk.Entry(self.labelframe4,
textvariable=self.codigoborra)
        self.entryborra.grid(column=1, row=0, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe4, text="Borrar",
command=self.borrar)
        self.boton1.grid(column=1, row=1, padx=4, pady=4)
```

```
def modificar(self):
        self.pagina5 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina5, text="Modificar artículo")
        self.labelframe5=ttk.LabelFrame(self.pagina5, text="Artículo")
        self.labelframe5.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe5, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigomod=tk.StringVar()
        self.entrycodigo=ttk.Entry(self.labelframe5,
textvariable=self.codigomod)
        self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe5,
text="Descripción:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.descripcionmod=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe5,
textvariable=self.descripcionmod)
        self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)
        self.label3=ttk.Label(self.labelframe5, text="Precio:")
        self.label3.grid(column=0, row=2, padx=4, pady=4)
        self.preciomod=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe5,
textvariable=self.preciomod)
        self.entryprecio.grid(column=1, row=2, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe5, text="Consultar",
command=self.consultar_mod)
        self.boton1.grid(column=1, row=3, padx=4, pady=4)
        self.boton2=ttk.Button(self.labelframe5, text="Modificar",
command=self.modifica)
        self.boton2.grid(column=1, row=4, padx=4, pady=4)
```

```
    def borrar(self):
        datos=(self.codigoborra.get(), )
        cantidad=self.articulo1.baja(datos)
        if cantidad==1:
            mb.showinfo("Información", "Se borró el artículo con dicho
código")
        else:
            mb.showinfo("Información", "No existe un artículo con dicho
código")
```

```
    def modifica(self):
        datos=(self.descripcionmod.get(), self.preciomod.get(),
self.codigomod.get())
        cantidad=self.articulo1.modificacion(datos)
        if cantidad==1:
            mb.showinfo("Información", "Se modificó el artículo")
        else:
```

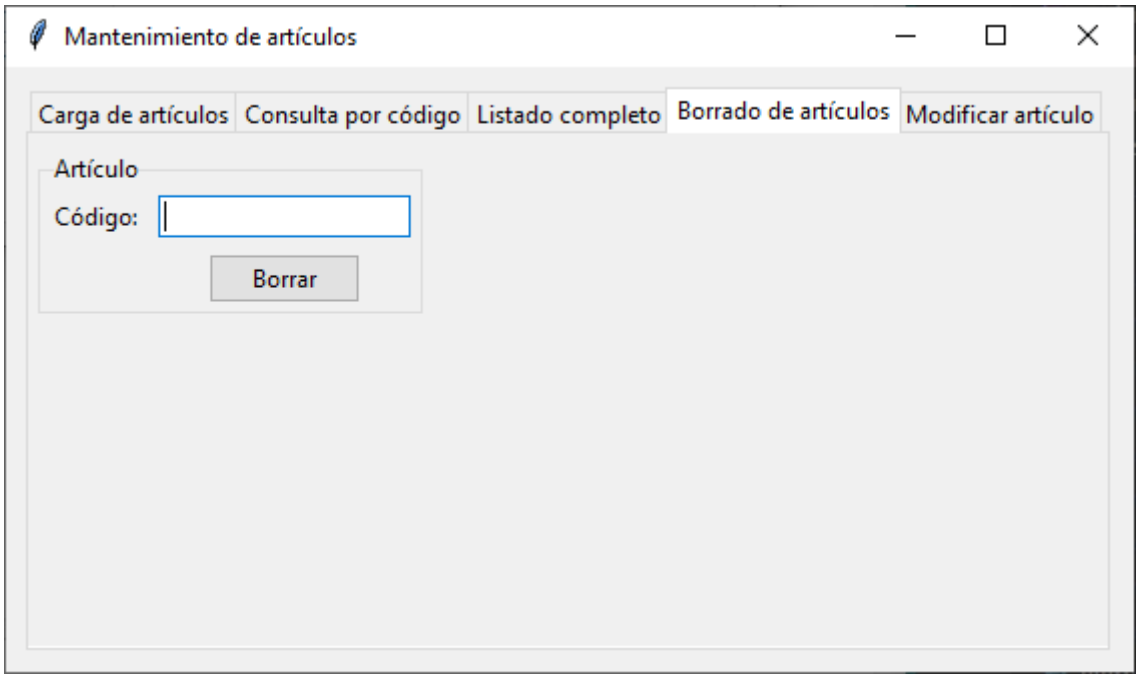
```
            mb.showinfo("Información", "No existe un artículo con dicho
código")
```

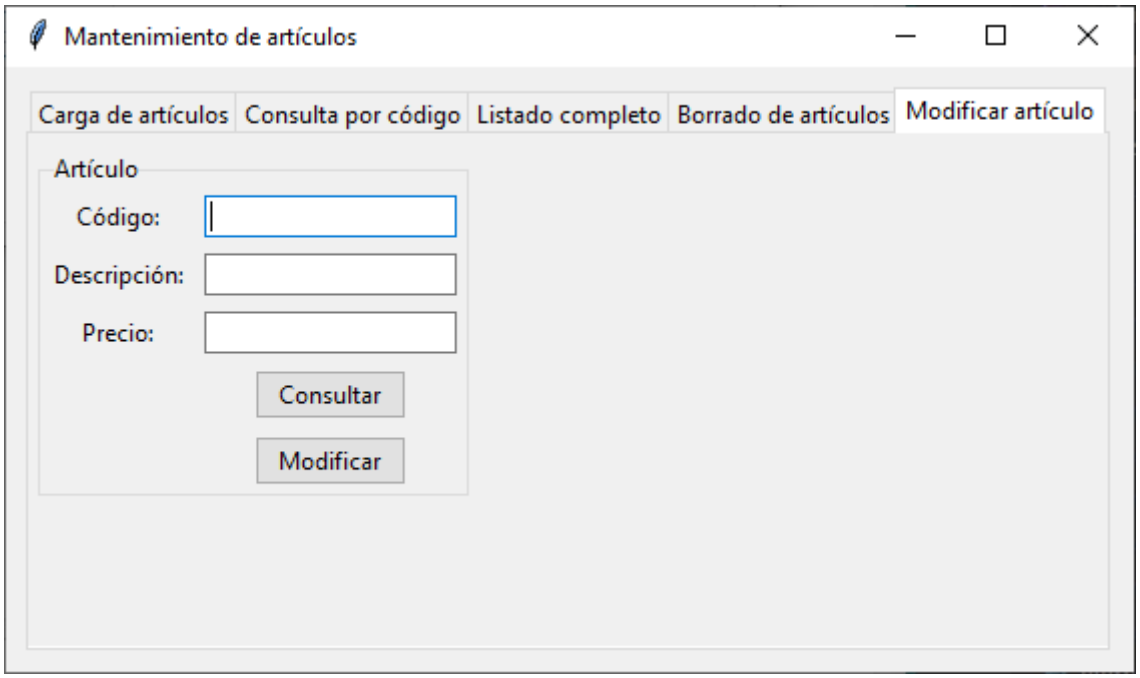
```
def consultar_mod(self):
        datos=(self.codigomod.get(), )
        respuesta=self.articulo1.consulta(datos)
        if len(respuesta)>0:
            self.descripcionmod.set(respuesta[0][0])
            self.preciomod.set(respuesta[0][1])
        else:
            self.descripcionmod.set('')
            self.preciomod.set('')
            mb.showinfo("Información", "No existe un artículo con dicho
código")
```

```
# Bloque principal
formularioarticulos=FormularioArticulos()
```

Vamos a ejecutar:

Para observar las dos pestañas nuevas que hemos añadido al proyecto.




Mantenimiento de artículos
Carga de artículos
Consulta por código
Listado completo
Borrado de artículos
Modificar artículo
Artículo
Código:
Descripción:
Precio:
Consultar
Modificar

Capítulo 279.- MySQL: Otras funciones del paquete mysql connector – 1

Recuperar el valor generado para un campo auto_increment

Cuando insertamos una fila en una tabla que contiene un campo que se auto incrementa podemos recuperar dicho valor en el mismo momento que efectuamos la inserción.

Luego de efectuar la inserción de una fila en la tabla procedemos a acceder al atributo 'lastrowid' que almacena el código que se acaba de generar.

Vamos a programar:

Ejecutar XAMPP Control Panel.

ejercicio279.py > ...

```
1   import mysql.connector
2
3   conexion1=mysql.connector.connect(host="localhost",
4                                     user="root",
5                                     passwd="",
6                                     database="db1")
7   cursor1=conexion1.cursor()
8   sql="insert into articulos(descripcion, precio) values (%s, %s)"
9   datos=("kiwi", 45.56)
10  cursor1.execute(sql, datos)
11  conexion1.commit()
12  print("Valor del último código del artículo ingresado:", cursor1.lastrowid)
13  conexion1.close()
```

Si ejecutamos este será el resultado:

```
Valor del último código del artículo ingresado: 11
```

Capítulo 280.- MySQL: Otra funcionalidad el paquete mysql connector – 2

Inserción de múltiples filas en una tabla.

La clase cursor a parte del método 'execute' cuenta con otro método llamada 'executemany' que tiene el objetivo de insertar múltiples filas de una tabla.

Si accedemos luego a la propiedad 'lastrowid' nos devuelve el código del último código de artículo almacenado.

Vamos a progamar.

ejercicio280.py > ...

```
1   import mysql.connector
2
3   conexion1=mysql.connector.connect(host="localhost",
4                                     user="root",
5                                     passwd="",
6                                     database="db1")
7   cursor1=conexion1.cursor()
8   sql="insert into articulos(descripcion, precio) values (%s, %s)"
9   filas=[("naranjas", 24),
10         ("peras", 12),
11         ("sandia", 54),
12         ("bananas", 56)]
13  cursor1.executemany(sql, filas)
14  conexion1.commit()
15  conexion1.close()
```

Ejecutamos y consultamos la base de datos:

←T→			codigo	descripcion	precio
Editar	Copiar	Borrar	8	Pomelo amarillo	35
Editar	Copiar	Borrar	9	Patatas	21
Editar	Copiar	Borrar	10	kiwi	45.56
Editar	Copiar	Borrar	11	kiwi	45.56
Editar	Copiar	Borrar	12	naranjas	24
Editar	Copiar	Borrar	13	peras	12
Editar	Copiar	Borrar	14	sandia	54
Editar	Copiar	Borrar	15	bananas	56

Los cuatro últimos registros los hemos añadido de una sola vez.

Capítulo 281.- MySQL: Otras funcionalidades del paquete mysql connector – 3

Creación de una base de datos y tablas.

En conceptos anteriores vimos como crear una base de datos de MySQL utilizando la aplicación PHPMyAdmin, en algunas situaciones podemos necesitar crear una base de datos desde el mismo programa de Python. La misma metodología será si queremos crear tablas.

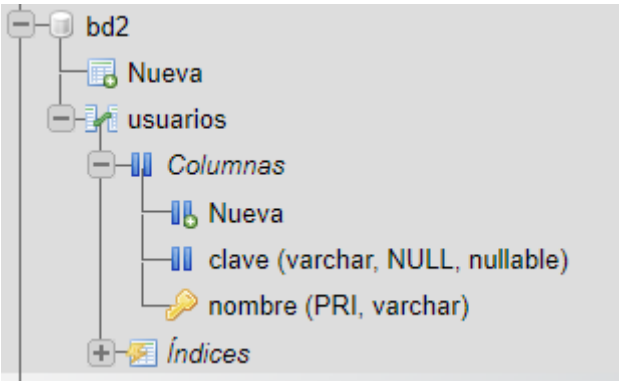
Vamos a programar:

ejercicio281.py > ...

```
1   import mysql.connector
2
3   conexion1=mysql.connector.connect(host="localhost",
4                                     user="root",
5                                     passwd="")
6
7   cursor1=conexion1.cursor()
8   sql="create database bd2"
9   cursor1.execute(sql)
10  sql="use bd2"
11  cursor1.execute(sql)
12  sql="""create table usuarios(
13          nombre varchar(30) primary key,
14          clave varchar(30)
15  )"""
16  cursor1.execute(sql)
17  conexion1.commit()
18  conexion1.close()
```

Vamos a ejecutar:

Vamos a consultar desde PHPMyAdmin.



Capítulo 282.- MySQL: Otras funcionalidades del paquete mysql connector – 4

Borrar una base de datos.

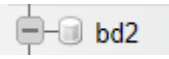
Cuando queremos crear una base de datos que ya existe se genera un error, podemos primero borrarla y luego ya si crearla sin problemas.

Vamos a programar:

ejercicio282.py > ...

```
1   import mysql.connector
2
3   conexion1=mysql.connector.connect(host="localhost",
4                                     user="root",
5                                     passwd="")
6
7   cursor1=conexion1.cursor()
8   sql="drop database if exists bd2"
9   cursor1.execute(sql)
10
11  sql="create database bd2"
12  cursor1.execute(sql)
13  conexion1.close()
```

Ejecutamos:



Desde PHPMyAdmin podemos observar que la ha borrado y la ha creado de nuevo, porque en la base de datos anterior tenía una tabla que ahora no la tiene.

Capítulo 283.- MySQL: Otras funcionalidades del paquete mysql connector -5

Borrar una tabla y crear otra con el mismo nombre.

No se puede crear una tabla con el mismo nombre de una existente, en esos casos debemos borrar la actual y crear una nueva.

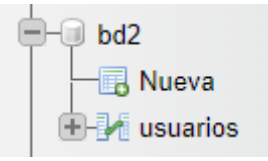
Vamos a programar:

ejercicio283.py > ...

```
1   import mysql.connector
2
3   conexion1=mysql.connector.connect(host="localhost",
4                                     user="root",
5                                     passwd="",
6                                     database="bd2")
7
8   cursor1=conexion1.cursor()
9   sql="drop table if exists usuarios"
10  cursor1.execute(sql)
11  sql="""
12          create table usuarios(
13              nombre varchar(30) primary key,
14              clave varchar(30),
15              mail varchar(70)
16          )
17  """
18  cursor1.execute(sql)
19  conexion1.commit()
20  conexion1.close()
```

Ejecutamos:

Desde PHPMyAdmin podremos observar como la base de datos bd2 tiene una tabla:



Capítulo 284.- Manejo de excepciones en Python

Las excepciones son errores que se disparan durante la ejecución de un programa. En Python podemos dejar que dichas excepciones detengan el programa o en caso contrario escribir un algoritmo para reaccionar a dicha situación.

Los ejemplos más comunes que podemos nombrar de excepciones:

- Tratar de convertir a entero un string que no tienen valores numéricos.
- Tratar de dividir por cero.
- Abrir un archivo de texto inexistente o que se encuentra bloqueado por otra aplicación.
- Conectar con un servidor de bases de datos que no se encuentra activo.
- Acceder a subíndices de listas o tuplas inexistentes.
- Llamar a un método o función inexistente.
- Importar módulos que no existen.

La captura de excepciones nos permite crear programas mucho más robustos y tolerantes a fallas que ocurren en escasas situaciones, pero en caso que se presenten disponemos de un algoritmo alternativo para reaccionar a dicha situación evitando que el programa finalice su ejecución.

Problema:

Realizar la carga de dos números enteros por teclado e imprima su suma, luego preguntar si quiere seguir sumando valores.

Codificar dos programas en uno que capture la excepción de ingreso de datos no numéricos y el otro que no tenga en cuenta el tipo de entrada de datos.

Como podemos comprobar el programa finaliza si ingresamos la cadena 'juan' en lugar de un número. En un tipo de error que según el tipo de programa a que estamos desarrollando capturaremos ésta excepción, por ejemplo si es un pequeño script que utilizaremos nosotros mismos es posible que este algoritmo nos alcance y tengamos cuidado de cargar siempre valores enteros por teclado.

Pero que pasa si el programa lo utilizará otro usuario que no tenga cuidado en la entrada de datos, podemos modificar este programa para que no se interrumpa al ingresar valores no numéricos y le informe con un mensaje que valores están permitidos ingresar por teclado.

Vamos a programar:

ejercicio284.py > ...

```
1  while True:
2      valor1=int(input("Ingrese un primer número:"))
3      valor2=int(input("Ingrese un segundo número: "))
4      suma=valor1+valor2
5      print("La suma es", suma)
6      respuesta=input("Desea realizar la carga de otros dos valores? [s/n]")
7      if respuesta=="n":
8          break
```

Cuando ejecutamos el programa y cuando nos piden por ingresar unos valores, estamos obligados a contestar por valores numéricos, de lo contrario se cometerá un error que interrumpirá la ejecución del programa.

```
Ingrese un primer número:7
Ingrese un segundo número: 5
La suma es 12
Desea realizar la carga de otros dos valores? [s/n]s
Ingrese un primer número:Juan
Traceback (most recent call last):
  File "d:\Curso completo de Python\ejercicio284.py", line 2, in <module>
    valor1=int(input("Ingrese un primer número:"))
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'Juan'
```

Mientras introduzcamos valores numéricos el programa funcionará correctamente, pero si por error en lugar de introducir un número introducimos un texto, en este caso se genera un error de tipo ValueError.

Como podemos evitar que el programa nos indique que debemos ingresar un número pero queremos que el programa continúe con la ejecución del mismo.

ejercicio284.py > ...

```
1  while True:
2      try:
3          valor1=int(input("Ingrese un primer número:"))
4          valor2=int(input("Ingrese un segundo número: "))
5          suma=valor1+valor2
6      except ValueError:
7          print("Debe ingresar números.")
8      print("La suma es", suma)
9      respuesta=input("Desea realizar la carga de otros dos valores? [s/n]")
10     if respuesta=="n":
11         break
```

Cuando ejecutemos:

```
Ingrese un primer número:5
Ingrese un segundo número: 7
La suma es 12
Desea realizar la carga de otros dos valores? [s/n]s
Ingrese un primer número:Juan
Debe ingresar números.
La suma es 12
Desea realizar la carga de otros dos valores? [s/n]s
Ingrese un primer número:3
Ingrese un segundo número: 9
La suma es 12
Desea realizar la carga de otros dos valores? [s/n]
```

El programa controla el error y si ese sucede nos mostrará un mensaje y además seguirá la ejecución del programa.

Capítulo 285.- Excepciones estándar de Python

Python provee un conjunto de excepciones por defecto que podemos hacer uso en nuestros programas.

Algunas de las excepciones más comunes que podemos utilizar son:

- **ValueError**: Se genera cuando una operación o función recibe un argumento que tiene un valor inapropiado como vimos en el concepto anterior al tratar de convertir un string a entero siendo que el string no contiene valores numéricos.
- **IndexError**: Se genera cuando un subíndice de una secuencia (tupla, lista, string) está fuera de rango. Por ejemplo en la lista de 5 elementos queremos acceder al elemento con subíndice 70.

Excepciones.py - D:/Curso completo de Python/Excepciones.py (3.11.1)

File Edit Format Run Options Window Help

```
1 lista=[1,3,5,7,9]
2 print(lista[70])
```

```
Traceback (most recent call last):
  File "D:/Curso completo de Python/Excepciones.py", line 2, in <module>
    print(lista[70])
IndexError: list index out of range
```

- **NameError**: Se genera cuando llamamos a una función, método o tratamos de acceder a una variable inexistente.

Excepciones.py - D:/Curso completo de Python/Excepciones.py (3.11.1)

File Edit Format Run Options Window Help

```
1 print(x)
2
```

```
Traceback (most recent call last):
  File "D:/Curso completo de Python/Excepciones.py", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
```

- **TypeError**: Se genera cuando una operación o función se aplica a un tipo de dato inapropiado. Por ejemplo si queremos sumar un entero con un string:

Excepciones.py - D:/Curso completo de Python/Excepciones.py (3.11.1)

File Edit Format Run Options Window Help

```
1 suma=10 + 'Juan'
2
```

```
Traceback (most recent call last):
  File "D:/Curso completo de Python/Excepciones.py", line 1, in <module>
    suma=10 + 'Juan'
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- **ModuleNotFoundError**: Se genera cuando la declaración de un import tiene problemas al intentar cargar el módulo inexistente:

```
Excepciones.py - D:/Curso completo de Python/Excepciones.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 import randomx
2
```

```
Traceback (most recent call last):
  File "D:/Curso completo de Python/Excepciones.py", line 1, in <module>
    import randomx
ModuleNotFoundError: No module named 'randomx'
```

- **KeyError**: Se genera cuando no se encuentra una clave en un diccionario:

```
Excepciones.py - D:/Curso completo de Python/Excepciones.py (3.11.1)
File  Edit  Format  Run  Options  Window  Help
1 productos={"manzanas":39, "peras":32, "lechuga":17}
2 print(productos["sandia"])
```

```
Traceback (most recent call last):
  File "D:/Curso completo de Python/Excepciones.py", line 2, in <module>
    print(productos["sandia"])
KeyError: 'sandia'
```

Un listado completo de las excepciones estándar de Python las puedes consultar aquí.

Problema:

Realizar la carga de dos números por teclado e imprimir la división del primero respecto al segundo, capturar la excepción ZeroDivisionError.

```
ejercicio285.py > ...
1  try:
2      valor1=int(input("Ingrese dividendo:"))
3      valor2=int(input("Ingrese divisor:"))
4      division=valor1/valor2
5  except ZeroDivisionError:
6      print("No se puede dividir por cero.")
```

```
Ingrese dividendo:7
Ingrese divisor:0
No se puede dividir por cero.
```

Problema:

Almacenar en una tupla los nombres de meses del año. Solicitar el ingreso del número de mes y mostrar seguidamente el nombre de dicho mes. Capturar la excepción IndexError.

```
ejercicio285.py > ...
1 meses=("enero", "febrero", "marzo", "abril", "mayo", "junio",
2          "julio", "agosto", "septiembre", "octubre", "noviembre", "diciembre")
3 try:
4     numero_mes=int(input("Ingrese un número de mes [1-12:]"))
5     print(meses[numero_mes-1])
```

```
6 except IndexError:
7     print("El número de mes debe ir entre 1 y 12")
```

Cuando introducimos un valor mayor de 12 nos avisa con un mensaje.

```
Ingrese un número de mes [1-12:]7
julio
```

```
Ingrese un número de mes [1-12:]15
El número de mes debe ir entre 1 y 12
```

Capítulo 286.- Instrucción finally en un bloque try

Hemos visto que la sintaxis para la captura de excepciones es:

try:
  [algoritmo principal]
except [nombre de la excepción 1]:
  [algoritmo alternativo 1]
except [nombre de la excepción 2]:
  [algoritmo alternativo 2]
except [nombre de la excepción 3]
  [algoritmo alternativo 3]

Hay otra sección opcional de la cláusula 'try' llamada 'finally'.

try:
  [algoritmo principal]
except [nombre de la excepción 1]:
  [algoritmo alternativo 1]
except [nombre de la excepción 2]:
  [algoritmo alternativo 2]
except [nombre de la excepción 3]
  [algoritmo alternativo 3]

filally:
  [algoritmo que siempre se ejecuta]

Cuando en el algoritmo principal se genera una excepción las instrucciones que continúan nunca se ejecutarán ya que pesar a ejecutar el bloque 'except'. Hay situaciones donde queremos que haya un algoritmo que siempre se ejecute ya sea que se dispare una excepción o no.

Veamos algunos ejemplos donde es conveniente implementar el bloque finally.

Problema:

Almacenar una serie de string en un archivo de texto. Tratar de llamar al método 'write' pasando un entero.

Capturar la excepción TypeError.

ejercicio286.py > ...

```
1   try:
2       archivo1=open("datos.txt", "w")
3       archivo1.write("Linea 1\n")
4       archivo1.write("Linea 2\n")
5       archivo1.write(100)
6   except TypeError:
7       print("No se puede grabar un entero en el write")
8   finally:
9       archivo1.close()
10      print("Se cerró el archivo")
```

Este será el resultado cuando lo ejecutemos:

```
No se puede grabar un entero en el write
Se cerró el archivo
```

ejercicio286.py > ...

```
1  try:
2      archivo1=open("datos.txt", "w")
3      archivo1.write("Linea 1\n")
4      archivo1.write("Linea 2\n")
5      # archivo1.write(100)
6  except TypeError:
7      print("No se puede grabar un entero en el write")
8  finally:
9      archivo1.close()
10     print("Se cerró el archivo")
```

Comentamos la línea 5 que es la encargada de producir el error, este será el resultado:

```
Se cerró el archivo
```

Problema:

Conectarse a una base de datos de MySQL y ejecutar un comando SQL incorrecto.

ejercicio286.py > ...

```
1  import mysql.connector
2
3  try:
4      conexion1=mysql.connector.connect(host="localhost",
5                                        user="root",
6                                        passwd="")
7      cursor1=conexion1.cursor()
8      cursor1.execute("whow databasesxxxxx")
9      for base in cursor1:
10         print(base)
11 except mysql.connector.errors.ProgrammingError:
12     print("Error en el comando SQL")
13 finally:
14     conexion1.close()
15     print("Se cerro la conexión con la base de datos")
```

Vamos a ejecutar:

```
Error en el comando SQL
Se cerro la conexión con la base de datos
```

Vamos a modificar el código para evitar el error.

```
ejercicio286.py > ...
1   import mysql.connector
2
3   try:
4       conexion1=mysql.connector.connect(host="localhost",
5                                         user="root",
6                                         passwd="")
7       cursor1=conexion1.cursor()
8       cursor1.execute("show databases")
9       for base in cursor1:
10          print(base)
11  except mysql.connector.errors.ProgrammingError:
12      print("Error en el comando SQL")
13  finally:
14      conexion1.close()
15      print("Se cerro la conexión con la base de datos")
```

Ejecutamos de nuevo:

```
('bd2',)
('db1',)
('information_schema',)
('mysql',)
('performance_schema',)
('phpmyadmin',)
('test',)
Se cerro la conexión con la base de datos
```

No hay error pero las líneas 14 y 15 también se ejecutan.

Capítulo 287.- Base de datos desde Python

La segunda base de datos que veremos se acceso desde Python es SQLite.

SQLite también es un gestor de base de datos relacional pero con objetivos muy diferentes a MySQL, SQLServer, Oracle, etc.

Este gestor de base de datos tiene por objetivo ser parte de la misma aplicación con la que colabora, es decir no cumple los conceptos de cliente y servidor.

Para entender sus usos podemos dar algunos ejemplos donde se utiliza el gestor SQLite.

- Firefox usa SQLite para almacenar los favoritos, el historial, los cookies, etc.
- También el navegador Opera usa SQLite.
- La aplicación de comunicaciones Skype de Microsoft utiliza SQLite.
- Los sistemas operativos Android y IOS adoptan SQLite para permitir el almacenamiento y recuperación de datos.

SQLite es Open Source y se ha instalado por defecto con Python, es decir forma parte de la biblioteca estándar, no tenemos que instalar ningún módulo con pip.

Si nuestra aplicación necesita almacenar gran cantidad de información local con cierta estructura el empleo de SQLite es nuestra principal opción.

Creación de una base de datos y tablas.

En principio no se requiere tener más que Python instalado para poder trabajar con SQLite. Podemos desde nuestra propia aplicación crear la base de datos y sus tablas.

Vamos a programar:

ejercicio287_1.py > ...

```
1   import sqlite3
2
3   conexion=sqlite3.connect("bd1.db")
4   try:
5       conexion.execute("""
6                                   create table articulos(
7                                       codigo integer primary key autoincrement,
8                                       descripcion teext,
9                                       precio real
10                                  )
11                              """)
12      print("Se creo la tabla artículos")
13  except sqlite3.OperationalError:
14      print("La tabla artículos ya existe")
15  conexion.close()
```

Ejecutamos:

```
Se creo la tabla artículos
```

Al ejecutar si la base de datos no existe, se crea.

Insertar filas en una tabla.

Ahora implementamos un programa que inserte un par de filas en la tabla 'articulos' de la base de datos 'db1' que acabamos de crear con el programa anterior.

Vamos a programar:

```
ejercicio287_2.py > ...
import sqlite3

conexion=sqlite3.connect("bd1.db")
conexion.execute("insert into articulos(descripcion, precio) values (?,?)",
                 ("naranjas", 23.5))
conexion.execute("insert into articulos(descripcion, precio) values (?,?)",
                 ("peras",34))
conexion.execute("insert into articulos(descripcion, precio) values (?,?)",
                 ("bananas",25))
conexion.commit()
conexion.close()
```

Ejecutamos.

Recuperar todas las filas de una tabla.

Implementamos un programa que solicite el ingreso del código de un producto y luego nos muestre su descripción y precio.

Vamos a programar:

```
ejercicio287_3.py > ...
import sqlite3

conexion=sqlite3.connect("bd1.db")
cursor=conexion.execute("select codigo, descripcion, precio from articulos")
for fila in cursor:
    print(fila)
conexion.close()
```

Ejecutamos:

```
(1, 'naranjas', 23.5)
(2, 'peras', 34.0)
(3, 'bananas', 25.0)
```

Recuperar una fila de una tabla.

Implementamos un programa que solicite el ingreso del código de un producto y luego nos muestre su descripción y precio.

Vamos a programar:

```
ejercicio287_4.py > ...
import sqlite3
conexion=sqlite3.connect("bd1.db")
codigo=int(input("Ingrese el código del artículo:"))
```

```
4  cursor=conexion.execute("select descripcion, precio from articulos where codigo=?",
5                          (codigo, ))
6  fila=cursor.fetchone()
7  if fila!=None:
8      print(fila)
9  else:
10     print("No existe un artículo con dicho código")
11     conexion.close()
```

Retorna un único elemento.

Vamos a ejecutar introduciendo un código de artículo que existe:

```
Ingrese el código del artículo:2
('peras', 34.0)
```

Ahora vamos a introducir un código de artículo que no existe:

```
Ingrese el código del artículo:5
No existe un artículo con dicho código
```

Recuperar varias filas de una tabla.

Implementamos un programa que solicite el ingreso de un precio y luego nos muestre la descripción de todos los artículos con un precio inferior al ingresado.

Vamos a programar:

ejercicio287_5.py > ...

```
1  import sqlite3
2  conexion=sqlite3.connect("bd1.db")
3  precio=float(input("Ingrese un precio:"))
4  cursor=conexion.execute("select descripcion, precio from articulos where precio<?",
5                          (precio, ))
6  filas=cursor.fetchall()
7  if len(filas)>0:
8      for fila in filas:
9          print(fila)
10 else:
11     print("No existe artículos con un precio menor al ingresado")
12 conexion.close()
```

Retorna varios elementos.

Vamos a ejecutar con un precio de 58.

```
Ingrese un precio:58
('naranjas', 23.5)
('peras', 34.0)
('bananas', 25.0)
```

Ahora con un precio de 30.

```
Ingrese un precio:30
('naranjas', 23.5)
('bananas', 25.0)
```

Capítulo 288.- SQLite: Interfaz visual con tkinter y acceso a la base de datos – 1

Es muy posible que necesitemos en algunas situaciones acceder a una base de datos de SQLite desde una aplicación con una interfaz visual.

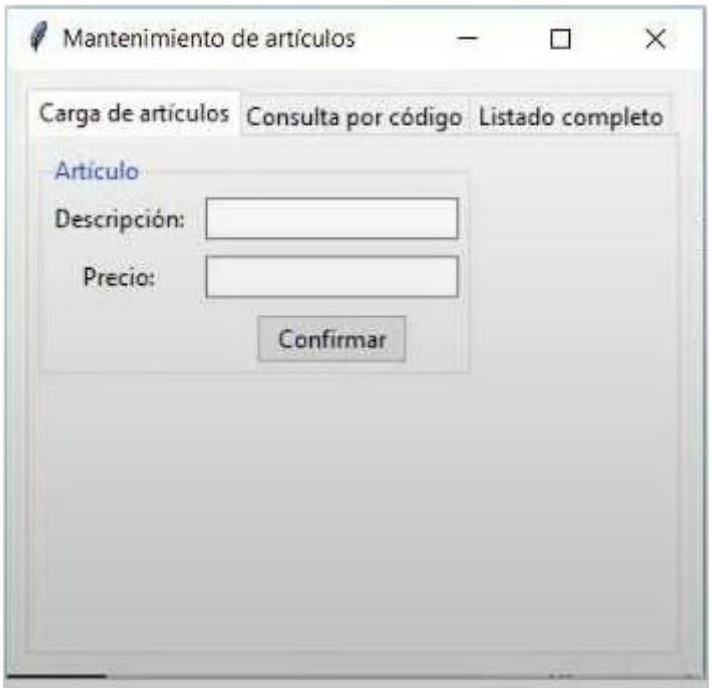
Implementaremos el mismo problema que resolvimos cuando trabajamos con el gestor de base de datos MySQL.

Problema:

Desarrollar una aplicación visual con la librería tkinter que permita implementar los algoritmos de carta de artículos, consulta por código y listado completo.

Trabajaremos con la base de datos 'bd1.db' que creamos en el concepto anterior.

La interfaz visual para la carga debe ser:



Vamos a programar:

Para ello vamos a crear una nueva carpeta llamada proyecto4, donde copiaremos la base de datos que realizamos en el capítulo anterior mas los dos proyectos nuevos con el nombre de artículos.py y formularioarticulos.py.

Código del programa artículos.py

```
import sqlite3

class Articulos:

    def abrir(self):
        conexion=sqlite3.connect("d:/CursocompletodePython/bd1.db")
        return conexion
```

```
    def alta(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="insert into articulos(descripcion, precio) values (?,?)"
        cursor.execute(sql, datos)
        cone.commit()
        cone.close()

    def consulta(self, datos):
        try:
            cone=self.abrir()
            cursor=cone.cursor()
            sql="select descripcion, precio from articulos where
codigo=?"
            cursor.execute(sql, datos)
            return cursor.fetchall()
        finally:
            cone.close()

    def recuperar_todos(self):
        try:
            cone=self.abrir()
            cursor=cone.cursor()
            sql="select codigo, descripcion, precio from articulos"
            cursor.execute(sql)
            return cursor.fetchall()
        finally:
            cone.close()
```

Código del programa formularioarticulos.py

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mb
from tkinter import scrolledtext as st
import articulos

class FormularioArticulos:
    def __init__(self):
        self.articulo1=articulos.Articulos()
        self.ventana1=tk.Tk()
        self.ventana1.title("Mantenimiento de artículos")
        self.cuaderno1 = ttk.Notebook(self.ventana1)
        self.carga_articulos()
        self.consulta_por_codigo()
        self.listado_completo()
        self.cuaderno1.grid(column=0, row=0, padx=10, pady=10)
        self.ventana1.mainloop()
```

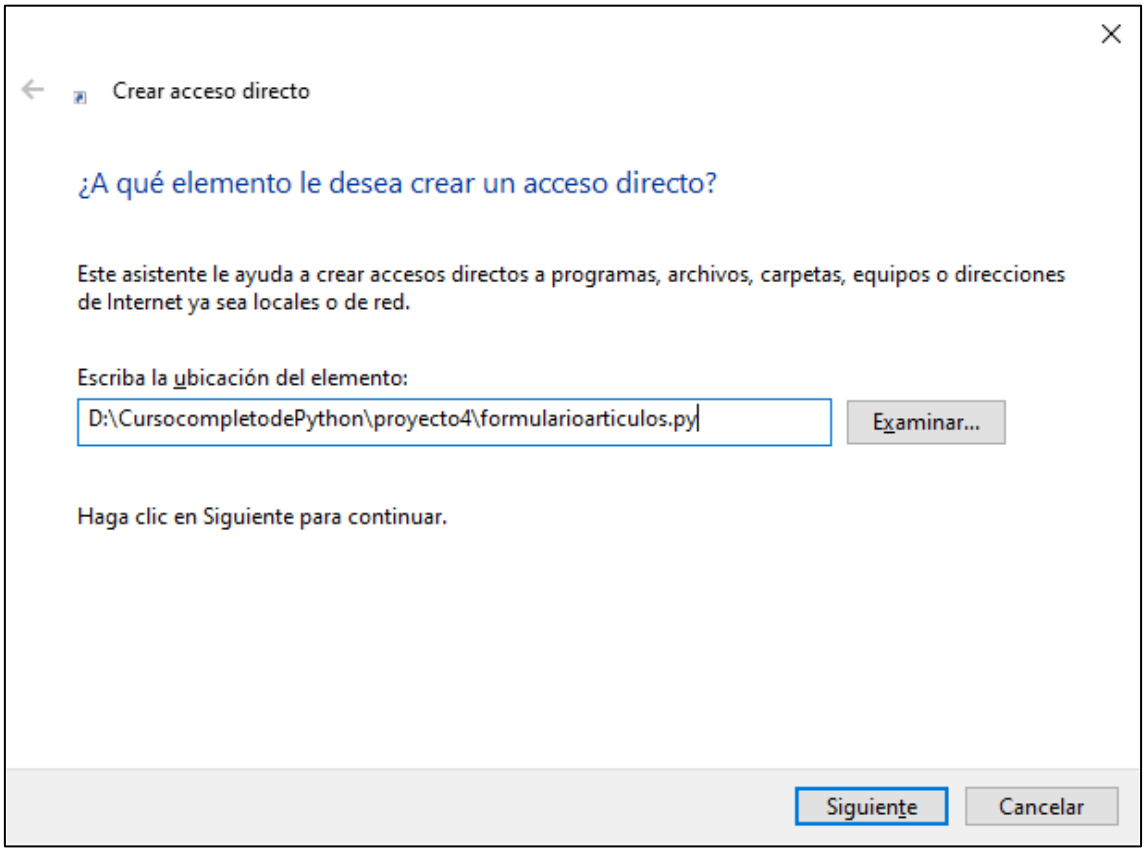
```
    def carga_articulos(self):
        self.pagina1 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina1, text="Carga de artículos")
        self.labelframe1=ttk.LabelFrame(self.pagina1,
text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe1, text="Descripción:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.descripcioncarga=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe1,
textvariable=self.descripcioncarga)
        self.entrydescripcion.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1, text="Precio:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.preciocarga=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe1,
textvariable=self.preciocarga)
        self.entryprecio.grid(column=1, row=1, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Confirmar",
command=self.agregar)
        self.boton1.grid(column=1, row=2, padx=4, pady=4)

    def agregar(self):
        datos=(self.descripcioncarga.get(), self.preciocarga.get())
        self.articulo1.alta(datos)
        mb.showinfo("Información", "Los datos fueron cargados")
        self.descripcioncarga.set("")
        self.preciocarga.set("")

    def consulta_por_codigo(self):
        self.pagina2 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina2, text="Consulta por código")
        self.labelframe2=ttk.LabelFrame(self.pagina2, text="Artículo")
        self.labelframe2.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe2, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigo=tk.StringVar()
        self.entrycodigo=ttk.Entry(self.labelframe2,
textvariable=self.codigo)
        self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe2,
text="Descripción:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.descripcion=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe2,
textvariable=self.descripcion, state="readonly")
        self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)
        self.label3=ttk.Label(self.labelframe2, text="Precio:")
```

```
        self.label3.grid(column=0, row=2, padx=4, pady=4)
        self.precio=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe2,
textvariable=self.precio, state="readonly")
        self.entryprecio.grid(column=1, row=2, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe2, text="Consultar",
command=self.consultar)
        self.boton1.grid(column=1, row=3, padx=4, pady=4)

    def consultar(self):
        datos=(self.codigo.get(), )
        respuesta=self.articulo1.consulta(datos)
        if len(respuesta)>0:
            self.descripcion.set(respuesta[0][0])
            self.precio.set(respuesta[0][1])
        else:
            self.descripcion.set('')
            self.precio.set('')
            mb.showinfo("Información", "No existe un artículo con dicho
código")

    def listado_completo(self):
        self.pagina3 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina3, text="Listado completo")
        self.labelframe3=ttk.LabelFrame(self.pagina3, text="Artículo")
        self.labelframe3.grid(column=0, row=0, padx=5, pady=10)
        self.boton1=ttk.Button(self.labelframe3, text="Listado completo",
command=self.listar)
        self.boton1.grid(column=0, row=0, padx=4, pady=4)
        self.scrolledtext1=st.ScrolledText(self.labelframe3, width=30,
height=10)
        self.scrolledtext1.grid(column=0,row=1, padx=10, pady=10)

    def listar(self):
        respuesta=self.articulo1.recuperar_todos()
        self.scrolledtext1.delete("1.0", tk.END)
        for fila in respuesta:
            self.scrolledtext1.insert(tk.END,
"código:"+str(fila[0])+"\ndescripción:"+fila[1]+"\nprecio:"+str(fila[2])+
"\n\n")

# Bloque principal
aplicacion1=FormularioArticulos()
```

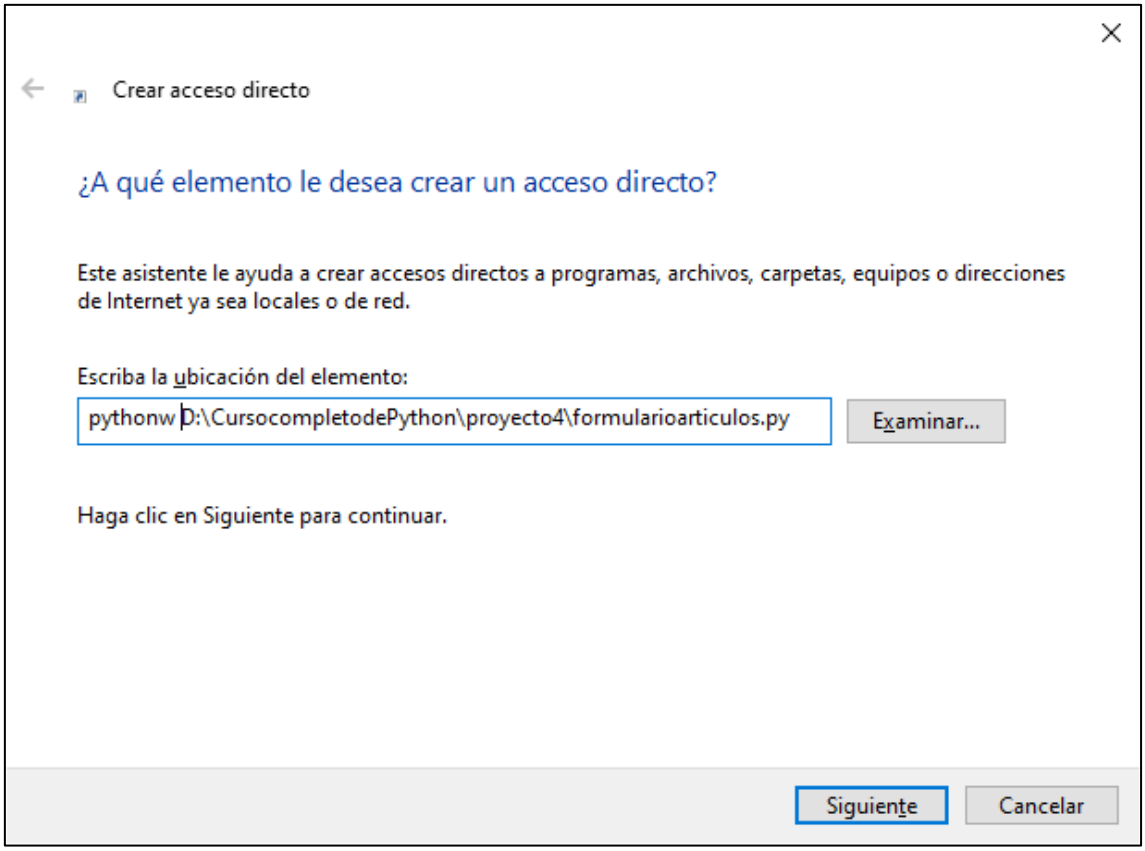
Vamos a ejecutar y probar la aplicación.

Podemos crear un acceso directo de la aplicación formularioarticulos.py en nuestro escritorio para poderlo ejecutar con mayor facilidad.

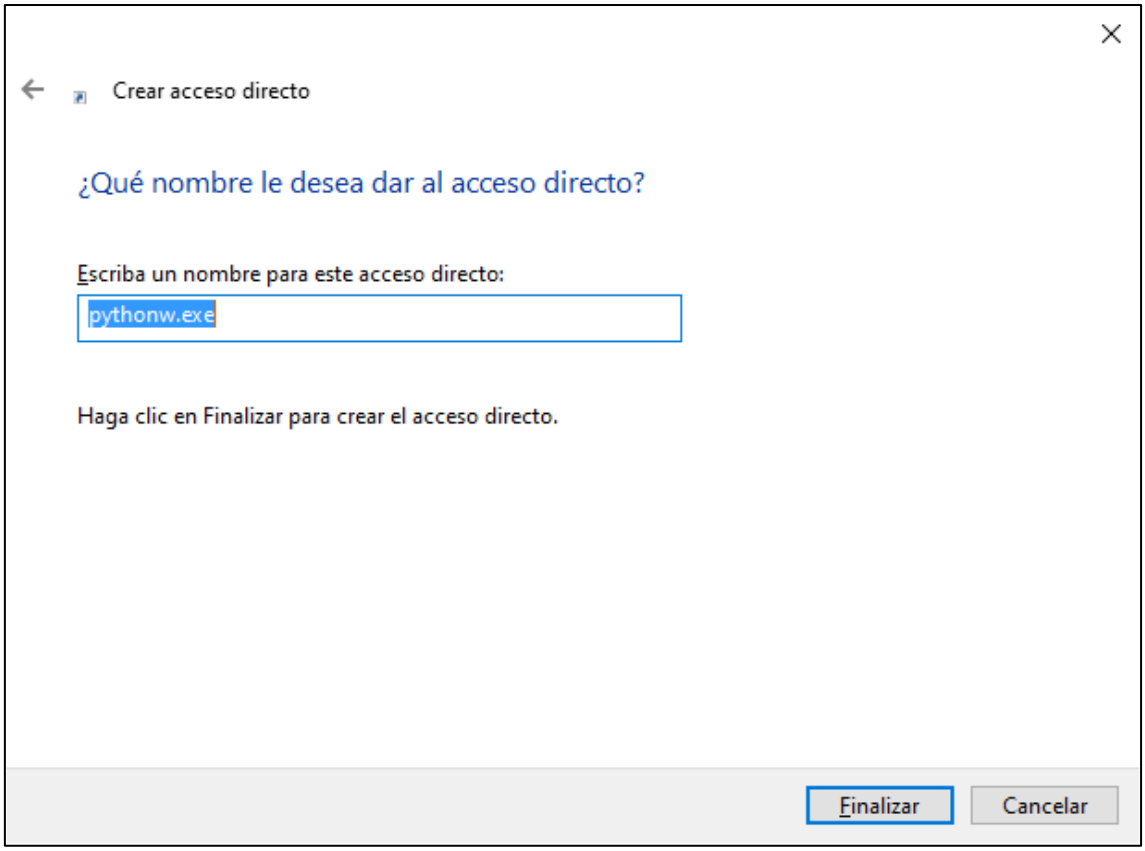
Botón derecho del ratón sobre el escritorio y seleccionaremos acceso directo.



Al principio de la línea vamos a agregar pytonw.

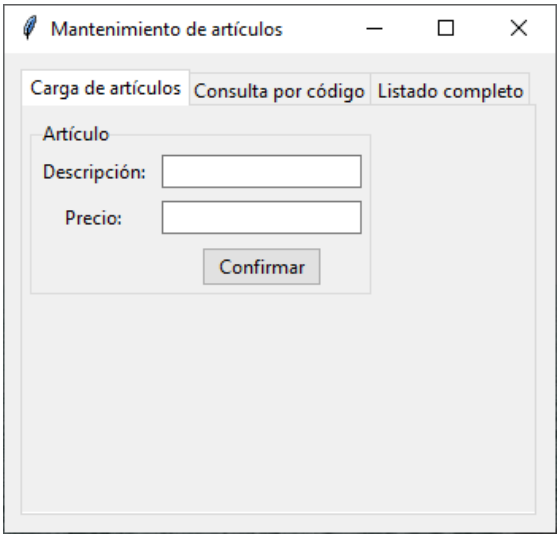


Le damos a siguiente.



Seguido de finalizar.

Ya podemos ejecutarlo haciendo doble clic.

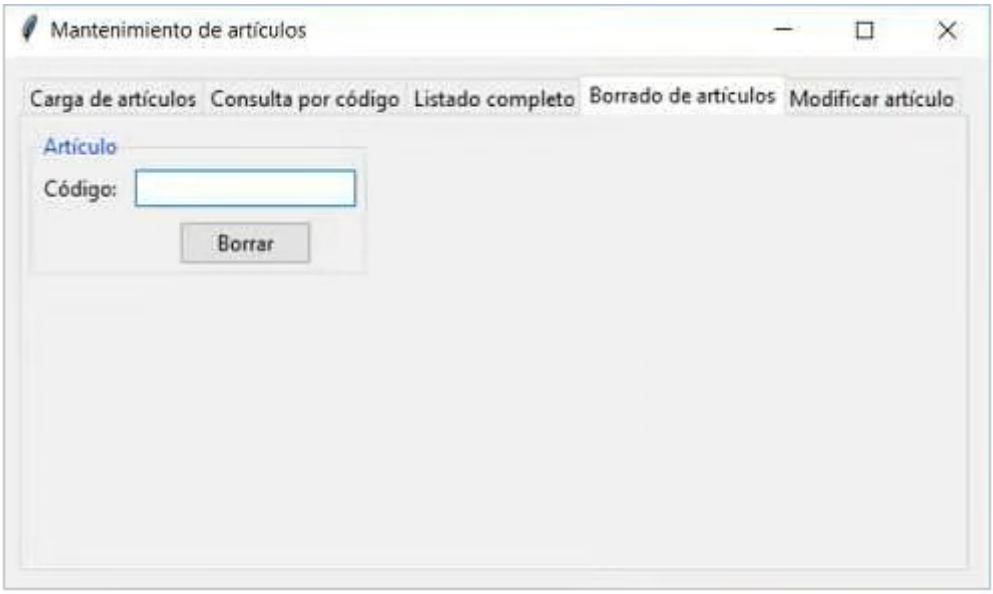


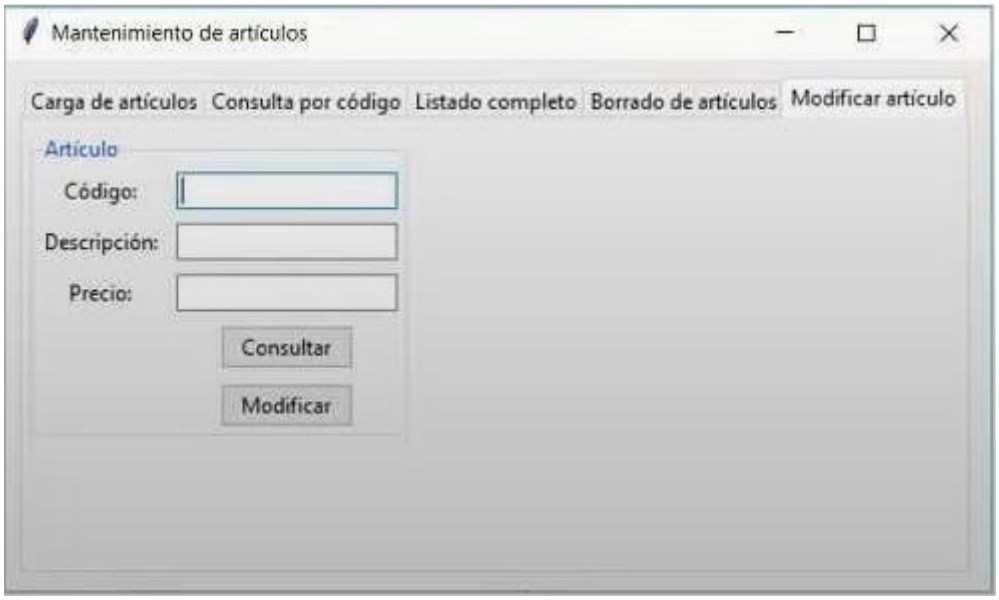
Capítulo 289.- SQLite: Interfaz visual con tkinter y acceso a la base de datos – 2

Problema propuesto

Agregar dos pestañas al programa de administración de artículos que permita borrar un artículo ingresando su código y otra opción que permita consultar y modificar las descripción y precio de un artículo.

La interfaces visuales a implementar son:





Vamos a programar:

Para este capítulo vamos a modificar el ejercicio que hicimos en el capítulo anterior.

Código del archivo 'formularioarticulos.py':

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mb
from tkinter import scrolledtext as st
import articulos

class FormularioArticulos:
    def __init__(self):
        self.articulo1=articulos.Articulos()
        self.ventana1=tk.Tk()
        self.ventana1.title("Mantenimiento de artículos")
        self.cuaderno1 = ttk.Notebook(self.ventana1)
        self.carga_articulos()
        self.consulta_por_codigo()
        self.listado_completo()
        self.borrado()
        self.modificar()
        self.cuaderno1.grid(column=0, row=0, padx=10, pady=10)
        self.ventana1.mainloop()

    def carga_articulos(self):
        self.pagina1 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina1, text="Carga de artículos")
        self.labelframe1=ttk.LabelFrame(self.pagina1,
text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe1, text="Descripción:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.descripcioncarga=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe1,
textvariable=self.descripcioncarga)
        self.entrydescripcion.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1, text="Precio:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.preciocarga=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe1,
textvariable=self.preciocarga)
        self.entryprecio.grid(column=1, row=1, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Confirmar",
command=self.agregar)
        self.boton1.grid(column=1, row=2, padx=4, pady=4)

    def agregar(self):
        datos=(self.descripcioncarga.get(), self.preciocarga.get())
        self.articulo1.alta(datos)
        mb.showinfo("Información", "Los datos fueron cargados")
        self.descripcioncarga.set("")
```

```
        self.preciocarga.set("")

    def consulta_por_codigo(self):
        self.pagina2 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina2, text="Consulta por código")
        self.labelframe2=ttk.LabelFrame(self.pagina2, text="Artículo")
        self.labelframe2.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe2, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigo=tk.StringVar()
        self.entrycodigo=ttk.Entry(self.labelframe2,
textvariable=self.codigo)
        self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe2,
text="Descripción:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.descripcion=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe2,
textvariable=self.descripcion, state="readonly")
        self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)
        self.label3=ttk.Label(self.labelframe2, text="Precio:")
        self.label3.grid(column=0, row=2, padx=4, pady=4)
        self.precio=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe2,
textvariable=self.precio, state="readonly")
        self.entryprecio.grid(column=1, row=2, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe2, text="Consultar",
command=self.consultar)
        self.boton1.grid(column=1, row=3, padx=4, pady=4)

    def consultar(self):
        datos=(self.codigo.get(), )
        respuesta=self.articulo1.consulta(datos)
        if len(respuesta)>0:
            self.descripcion.set(respuesta[0][0])
            self.precio.set(respuesta[0][1])
        else:
            self.descripcion.set('')
            self.precio.set('')
            mb.showinfo("Información", "No existe un artículo con dicho
código")

    def listado_completo(self):
        self.pagina3 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina3, text="Listado completo")
        self.labelframe3=ttk.LabelFrame(self.pagina3, text="Artículo")
        self.labelframe3.grid(column=0, row=0, padx=5, pady=10)
        self.boton1=ttk.Button(self.labelframe3, text="Listado completo",
command=self.listar)
```

```
        self.boton1.grid(column=0, row=0, padx=4, pady=4)
        self.scrolledtext1=st.ScrolledText(self.labelframe3, width=30,
height=10)
        self.scrolledtext1.grid(column=0,row=1, padx=10, pady=10)

    def listar(self):
        respuesta=self.articulo1.recuperar_todos()
        self.scrolledtext1.delete("1.0", tk.END)
        for fila in respuesta:
            self.scrolledtext1.insert(tk.END, "código:"+str(fila[0])+
                                              "\ndescripción:"+fila[1]+
                                              "\nprecio:"+str(fila[2])+"\
n\n")

    def borrado(self):
        self.pagina4 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina4, text="Borrado de artículos")
        self.labelframe4=ttk.LabelFrame(self.pagina4,
text="Artículo")
        self.labelframe4.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe4, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigoborra=tk.StringVar()
        self.entryborra=ttk.Entry(self.labelframe4,
textvariable=self.codigoborra)
        self.entryborra.grid(column=1, row=0, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe4, text="Borrar",
command=self.borrar)
        self.boton1.grid(column=1, row=1, padx=4, pady=4)

    def borrar(self):
        datos=(self.codigoborra.get(), )
        cantidad=self.articulo1.baja(datos)
        if cantidad==1:
            mb.showinfo("Información", "Se borró el artículo con dicho
código")
        else:
            mb.showinfo("Información", "No existe un artículo con dicho
código")

    def modificar(self):
        self.pagina5 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina5, text="Modificar artículo")
        self.labelframe5=ttk.LabelFrame(self.pagina5, text="Artículo")
        self.labelframe5.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe5, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigomod=tk.StringVar()
```

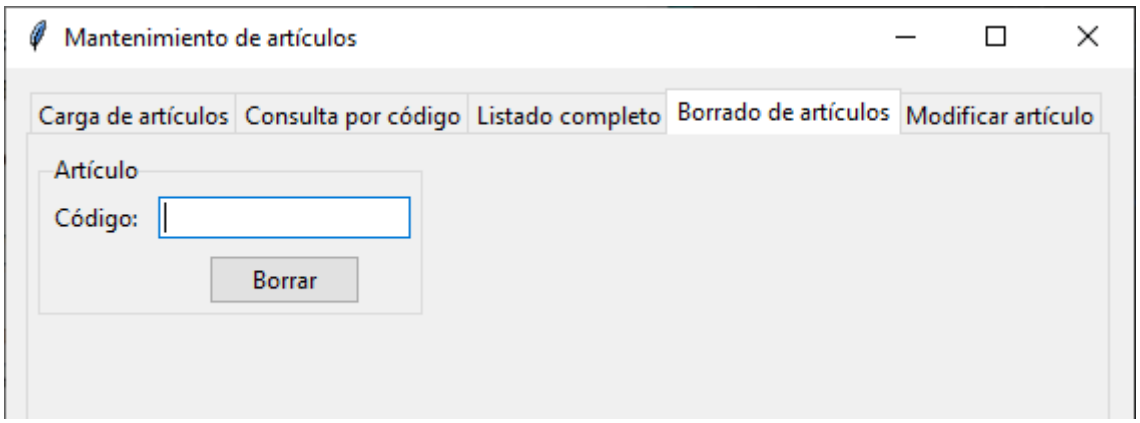
```
        self.entrycodigo=ttk.Entry(self.labelframe5,
textvariable=self.codigomod)
        self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe5,
text="Descripción:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.descripcionmod=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe5,
textvariable=self.descripcionmod)
        self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)
        self.label3=ttk.Label(self.labelframe5, text="Precio:")
        self.label3.grid(column=0, row=2, padx=4, pady=4)
        self.preciomod=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe5,
textvariable=self.preciomod)
        self.entryprecio.grid(column=1, row=2, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe5, text="Consultar",
command=self.consultar_mod)
        self.boton1.grid(column=1, row=3, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe5, text="Modificar",
command=self.modifica)
        self.boton1.grid(column=1, row=4, padx=4, pady=4)

    def modifica(self):
        datos=(self.descripcionmod.get(), self.preciomod.get(),
self.codigomod.get())
        cantidad=self.articulo1.modificacion(datos)
        if cantidad==1:
            mb.showinfo("Información", "Se modificó el artículo")
        else:
            mb.showinfo("Información", "No existe un artículo con dicho
código")

    def consultar_mod(self):
        datos=(self.codigomod.get(), )
        respuesta=self.articulo1.consulta(datos)
        if len(respuesta)>0:
            self.descripcionmod.set(respuesta[0][0])
            self.preciomod.set(respuesta[0][1])
        else:
            self.descripcionmod.set('')
            self.preciomod.set('')
            mb.showinfo("Información", "No existe un artículo con dicho
código")


aplicacion1=FormularioArticulos()
```

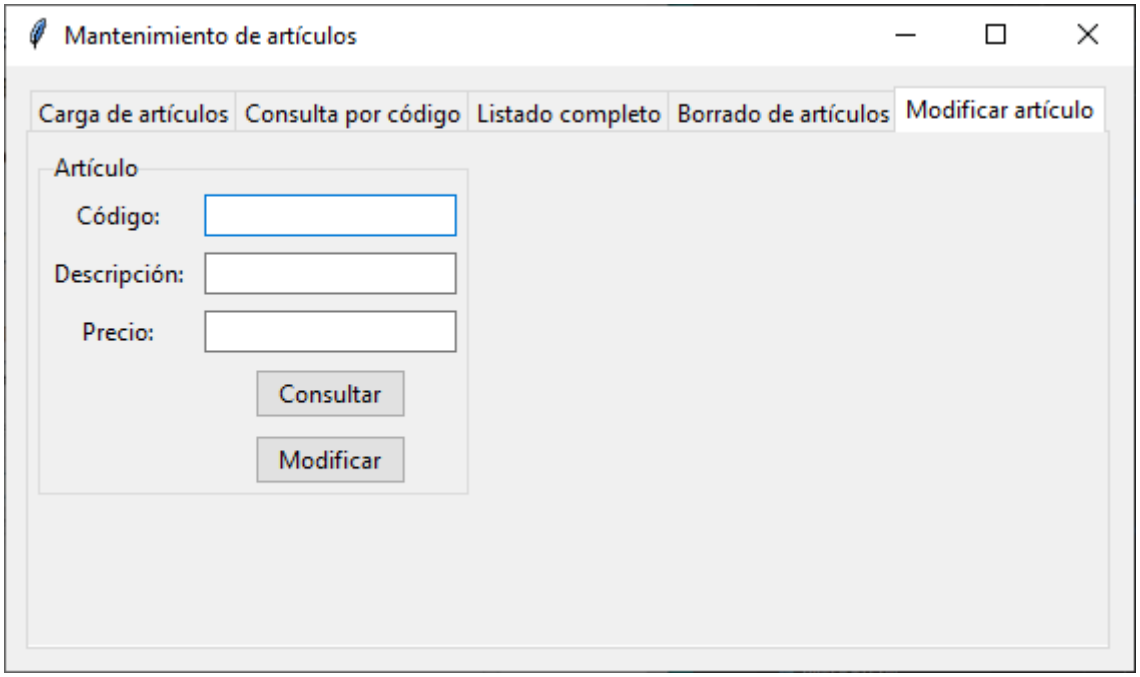
Código 'articulos.py':

```
import sqlite3

class Articulos:

    def abrir(self):
        conexion=sqlite3.connect("d:/CursocompletodePython/bd1.db")
        return conexion


    def alta(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="insert into articulos(descripcion, precio) values (?,?)"
        cursor.execute(sql, datos)
        cone.commit()
        cone.close()

    def consulta(self, datos):
        try:
            cone=self.abrir()
            cursor=cone.cursor()
            sql="select descripcion, precio from articulos where
codigo=?"
            cursor.execute(sql, datos)
            return cursor.fetchall()
        finally:
            cone.close()

    def recuperar_todos(self):
        try:
            cone=self.abrir()
            cursor=cone.cursor()
            sql="select codigo, descripcion, precio from articulos"
            cursor.execute(sql)
            return cursor.fetchall()
        finally:
            cone.close()

    def baja(self, datos):
        try:
            cone=self.abrir()
            cursor=cone.cursor()
            sql="delete from articulos where codigo=?"
            cursor.execute(sql, datos)
            cone.commit()
            return cursor.rowcount # retornamos la cantidad de filas
borradas
```

```
        except:
            cone.close()

    def modificacion(self, datos):
        try:
            cone=self.abrir()
            cursor=cone.cursor()
            sql="update articulos set descripcion=?, precio=? where
codigo=?"
            cursor.execute(sql, datos)
            cone.commit()
            return cursor.rowcount # retornamos la cantidad de filas
modificadas
        except:
            cone.close()
```

Cuando ejecutemos el archivo 'formularioarticulos.py' este serla el resultado:





Capítulo 290.- Python: f-strings para formateo de cadenas de caracteres

En las últimas versiones de ha agregado al lenguaje Python la posibilidad de definir string y dentro hacer la sustitución de variables, también llamado en otros lenguajes: interpolación de variables.

Veamos con un ejemplo cual es la sintaxis para implementar los f-strings en Python.

Problema:

Ingresar por teclado dos enteros, calcular su suma y luego mostrar un mensaje de los valores ingresados y su suma.

ejercicio290.py > ...

```
1   valor1=int(input("Ingrese primer valor:"))
2   valor2=int(input("Ingrese segundo valor:"))
3   suma=valor1+valor2
4   print(f"La suma de {valor1} y {valor2} es {suma}")
```

Vamos a ejecutar:

```
Ingrese primer valor:7
Ingrese segundo valor:10
La suma de 7 y 10 es 17
```

Para indicar que un string en Python contendrá variables a ser sustituidas debemos anteceder el carácter "f":

print(f"La suma de {valor1} y {valor2} es {suma}")

Las variables a ser sustituidas por su valor deben estar encerradas entre llaves.

Problema:

Definir una lista con 5 valores enteros. Mostrar los 5 valores formateados a derecha junto a su suma.

ejercicio290.py > ...

```
1   lista=[2000, 500, 17000, 24, 7]
2   for elemento in lista:
3       print(f"{elemento:10}")
4   print("----------")
5   print(f"{sum(lista):10}")
```

Si ejecutamos:

```
      2000
       500
     17000
        24
         7
----------
     19531
```

Cuando disponemos entre llaves la variable a sustituir podemos agregar luego dos puntos y la cantidad de espacios que ocupará la variable:

print(f"{elemento:10}")

También otra posibilidad es indicar entre llaves la llamada a una función (la función sum retorna la suma de toda la lista).

print(f"{sum(lista):10}")

Problema:

Definir una lista con 5 valores flotantes, con distintas cantidades de decimales. Mostrar solo con dos decimales.

```
ejercicio290.py > ...
1   lista=[30.195, 400.2, 20.5555, 18.34, 67.0]
2   for elemento in lista:
3       print(f"{elemento:10.2f}")
```

Si ejecutamos:

```
  30.20
 400.20
  20.56
  18.34
  67.00
```

Anotaciones

- Si queremos insertar los caracteres de apertura o cerrado de llaves en un f-string debemos duplicarlos para que lo interprete Python:

valor1=10
print(f"{{valor1}} tiene el valor {valor1}")

Luego el string que se emprime en pantalla es:

{valor1} tiene el valor 10

- Podemos utilizar tanto el carácter "f" minúsculas como el carácter "F" mayúsculas para indicar que se trata de un f-string.

valor1=10
print(F"{{valor1}} tiene el valor {valor1}")

- Se puede utilizar los f-strings para cadenas de múltiples líneas:

```
ejercicio290.py > ...
1   valor1=1000
2   valor2=340
3   suma=valor1+valor2
4   cadena=f"""
5   {valor1:5}
6   {valor2:5}
7   -----
8   {suma:5}
9   """
10  print(cadena)
```

Este será el resultado:

```
 1000
  340
-----
 1340
```

- Anteriormente sin la sintaxis de los f-strings debíamos convertir todos los datos a string y concatenarlos.

```
ejercicio290.py > ...
1   valor1=int(input("Ingrese primer valor:"))
2   valor2=int(input("Ingrese segundo valor:"))
3   suma=valor1+valor2
4   print("La suma de "+str(valor1)+" y "+str(valor2)+" es "+str(suma))
```

Este será el resultado:

```
Ingrese primer valor:7
Ingrese segundo valor:5
La suma de 7 y 5 es 12
```

También podemos utilizar el método format:

```
ejercicio290.py > ...
1   valor1=int(input("Ingrese primer valor:"))
2   valor2=int(input("Ingrese segundo valor:"))
3   suma=valor1+valor2
4   print("La suma de {} y {} es {}".format(valor1, valor2, suma))
```

Este será el resultado:

```
Ingrese primer valor:7
Ingrese segundo valor:5
La suma de 7 y 5 es 12
```

Con la llegada de los f-string el código fuente de nuestro programa es mucho más legible.

Capítulo 291.- Recursividad: Conceptos básicos – 1

La recursividad es una técnica de programación que nos permite que un bloque de instrucciones se ejecuten n veces. Remplaza en ocasiones a estructuras repetitivas.

La recursividad es un concepto difícil de entender en principio, pero luego de analizar diferentes problemas aparecen puntos comunes.

En Python las funciones o métodos pueden llamarse a sí mismos. Si dentro de una función o método existe la llamada a sí mismo decimos que la función o método es recursivo.

Cuando una función o método se llama a sí mismo, se asigna espacio en la pila par las nuevas variables locales y parámetros.

Al volver de una llamada recursiva, se recuperan de la pila las variables locales y los parámetros antiguos y la ejecución se reanuda en el punto de la llamada al método.

Problema 1:

Implementación de una función recursiva.

```
def repetir()
    repetir()

repetir()
```

La función repetir es recursiva porque dentro de la función se llama a sí misma.

Cuando ejecuta este programa se bloqueará y generará una excepción: "RecursionError: maximun recursión Depth exceded".

Analicemos como funciona:

Se llama a la función repetir.

Hay que tener en cuenta que cada vez que se llama a una función se reservan un conjunto de bytes de memoria que se liberarán cuando finalice la ejecución.

La primera línea de la función llama a la función repetir, es decir que se reservan más bytes nuevamente. Se ejecuta nuevamente una instancia de la función repetir y así sucesivamente hasta que la pila estática se colme y se cuelgue el programa.

Problema 2:

Implementación de una función recursiva que reciba un parámetro de tipo entero y luego llame en forma recursiva con el valor del parámetro menos 1.

ejercicio291.py > ...

```
1  def imprimir(x):
2      print(x)
3      imprimir(x-1)
4
5  imprimir(5)
```

Desde el bloque principal del programa se llama a la función imprimir y se la envía el valor 5. El parámetro x recibe el valor 5. Se ejecuta el algoritmo de la función, imprime el contenido del parámetro (5) y seguidamente se llama a una función, en este caso a sí misma (por eso decimos que es una función recursiva), enviándole el valor 4.

El parámetro x recibe el valor 4 y se imprime en pantalla el cuatro, llamando nuevamente a la función imprimir enviándole el valor 3.

Si continuamos este algoritmo podemos observar que en pantalla se emprime:

5 4 3 2 1 0 -1 -2 -3 . . . . . .

Hasta que se bloquee el programa generándose una excepción.

Tener en cuenta que cada llamada a una función consume varios bytes por la llamada y en este caso varios bytes por el parámetro x. Como nunca finaliza la ejecución completa de las funciones se desborda la pila estática por las sucesivas llamadas.

Problema 3:

Implementar una función recursiva que imprima en forma descendente de 5 a 1 de uno en uno.

ejercicio291.py > ...

```
1   def imprimir(x):
2       if x>0:
3           print(x)
4           imprimir(x-1)
5
6   imprimir(5)
```

Ahora si podemos ejecutar este programa y observar los resultados en pantalla. Se emprimen los números 5 4 3 2 1 y no se bloquea el programa.

Analice lo que sucede cuando el if x>0: se evalúa como falso, ¿a qué línea del programa retorna?

Problema 4:

Imprimir los números de 1 a 5 en pantalla utilizando recursividad.

ejercicio291.py > ...

```
1   def imprimir(x):
2       if x<=5:
3           print(x)
4           imprimir(x+1)
5
6   imprimir(1)
```

Con este ejemplo se presenta una situación donde debe analizarse línea a línea la ejecución del programa y el porqué de estos resultados.

¿Por qué se imprime en pantalla 1 2 3 4 5?

Veamos cómo se apilan las llamadas recursivas:

En la primera llamada desde el bloque principal del programa el parámetro x recibe el valor 5.

5	x

Cuando llamamos desde la misma función le enviamos el valor d x menos 1 y la memoria queda de la siguiente forma:

5	x
4	x`

Debemos entender que el parámetro x en la nueva llamada está en otra parte de la memoria y que almacena un 4, nosotros le llamaremos x prima.

Comienza a ejecutarse la función, la condición del if se evalúa como verdadero por lo que entra al bloque y llama recursivamente a la función imprimir pasándole el valor 3 al parámetro.

5	x
4	x`
3	x``

Nuevamente la condición se evalúa como verdadero y llama a la función enviándole un 2, lo mismo ocurre cuando le envía un 1 y un 0.

5	x
4	x`
3	x``
2	x```
1	x````
0	x``````

ejercicio291.py > ...

```
1   def imprimir(x):
2       if x>0:
3           print(x)
4           imprimir(x-1)
5
6   imprimir(5)
```

Cuando x vale 0 la condición if se valúa como falsa y sale de la función imprimir.

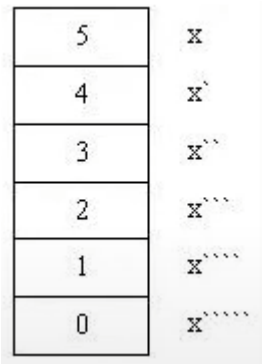
¿Qué línea ahora se ejecuta?

¿Vuelve al bloque principal y finaliza el programa? NO.

Recordemos que la última llamada de la función imprimir se había hecho desde la misma función imprimir por lo que vuelve a la línea:

```
imprimir(x-1)
print(x)
```

Ahora si analicemos que valor tiene el parámetro x. Observamos la pila de llamadas del gráfico:



x cuarta tiene el valor 1. Por lo que se imprime dicho valor en pantalla.

Luego de imprimir el 1 finaliza la ejecución de la función, se libera el espacio ocupado por el parámetro x y pasa a ejecutarse la siguiente línea donde se había llamado la función:

```
print(x)
```

Ahora x en esta instancia de la función tiene el valor 2.
Así sucesivamente hasta liberar todas las llamadas recursivas.

Es importante tener en cuenta que siempre en una función recursiva debe haber un if para finalizar la recursividad (en caso contrario la función recursiva será infinita y provocará que el programa se bloquee generando una excepción)

Es preferible utilizar estructuras repetitivas si es posible antes de la recursividad.

Capítulo 292.- Recursividad: Conceptos básicos – 2

Problema 5:

Otro problema típico que se presenta para analizar la recursividad es el obtener el factorial de un número.

Recordar que el factorial de un número es el resultado que se obtiene de multiplicar dicho número por el anterior y así sucesivamente hasta llegar a uno.

Ej. el factorial de 4 es 4*3*2*1 es decir 24.

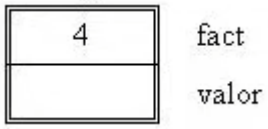
ejercicio292.py > ...

```
1   def factorial(fact):
2       if fact>0:
3           valor=fact*factorial(fact-1)
4           return valor
5       else:
6           return 1
7
8   print(f"el factorial de 4 es {factorial(4)}")
```

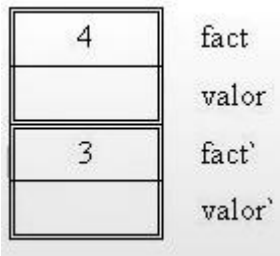
Este será el resultado:

```
el factorial de 4 es 24
```

La función factorial es recursiva porque desde la misma función llamamos a la función factorial. Debemos hacer el seguimiento del problema para analizar cómo se calcula.
La memoria en la primera llamada:



fact recibe el valor 4 y valor se cargará con el valor que se obtenga con el producto de fact por el valor devuelto por la función factorial (llamada recursiva)



Nuevamente se llama recursivamente hasta que el parámetro fact reciba el valor 0.

4	fact
	valor
3	fact`
	valor `
2	fact``
	valor ``
1	fact```
	valor ```
0	fact````

Cuando fact recibe un cero la condición del if se valúa como falsa y ejecuta el else retornando un 1, la variable local de la llamada anterior a la función queda de la siguiente manera:

4	fact	4	fact	4	fact	4	fact
	valor		valor		valor	24	valor
3	fact`	3	fact`	3	fact`		
	valor `		valor `	6	valor `		
2	fact``	2	fact``				
	valor ``	2	valor ``				
1	fact```						
1	valor ```						

Es importantísimo en tener la liberación del espacio de las variables locales y los parámetros en las sucesivas llamadas recursivas.

Por último el bloque principal recibe "valor", en este caso el valor 24.

Capítulo 293.- Recursividad: Conceptos básicos – 3

Problema 6:

Implementar un método recursivo para ordenar los elementos de una lista.

ejercicio293.py > ...

```
1   def ordenar(lista, cant):
2       for f in range(0, cant-1):
3           if lista[f]>lista[f+1]:
4               aux=lista[f]
5               lista[f]=lista[f+1]
6               lista[f+1]=aux
7           ordenar(lista, cant-1)
8
9   datos=[60, 44, 22, 33, 2]
10  print(datos)
11  ordenar(datos, len(datos))
12  print(datos)
```

Este será el resultado:

```
[60, 44, 22, 33, 2]
[2, 22, 33, 44, 60]
```

Hasta ahora hemos visto problemas que se pueden resolver tanto con recursividad como con una estructuras repetitivas.

Es muy importante tener en cuenta que siempre que podamos emplear algoritmos no recursivos será mejor (ocupa menos memoria de ram y se ejecuta más rápidamente).

Pero hay casos donde el empleo de recursividad hace mucho más sencillo el algoritmo (tener en cuenta que no es el aso de los problemas vistos previamente).

Capítulo 294.- Recursividad: Problemas donde conviene aplicar la recursividad - 1

En el concepto anterior se vieron pequeños problemas para entender cómo funciona la recursividad, pero no se desarrollaron problemas donde conviene utilizar la recursividad.

Problema:

Recorre un árbol de directorios en forma recursiva. Mostrar de cada directorio los archivos y directorios, luego descender a cada directorio y hacer la misma actividad.

ejercicio294.py > ...

```
1   import os
2
3   def leer(directorio):
4       lista=os.listdir(directorio)
5       for elemento in lista:
6           if os.path.isfile(directorio+elemento):
7               print(elemento+" [archivo]")
8           if os.path.isdir(directorio+elemento):
9               print(elemento+" [directorio]")
10              leer(directorio+elemento+"/")
11
12  leer("d:/CursocompletodePython/")
```

La función leer es recursiva ya que dentro de la misma función se llama leer.

Importamos el módulo 'os':

import os

Lo primero que hacemos dentro de la función leer es llamar a la función 'listdir' del módulo 'os' que lo que hace es retornar una lista con todos los nombres de archivos y directorios contenidos en la carpeta que le pasamos como parámetro (la primera vez que se llama la función leer recibe la carpeta 'd:/CursocompletodePython/'):

```
def leer(directorio):
    lista=os.listdir(directorio)
```

Recorremos la lista en forma completa y por cada entrada en la lista verificamos si se trata de un archivo:

```
for elemento in lista:
    if os.path.isfile(directorio+element):
        print(element+" [archive]")
```

Capítulo 295.- Recursividad: Problemas donde conviene aplicar la recursividad - 2

Problema:

Desarrollar un programa que permita recorrer un laberinto e indique si tiene salida o no empleando la librería tkinter.

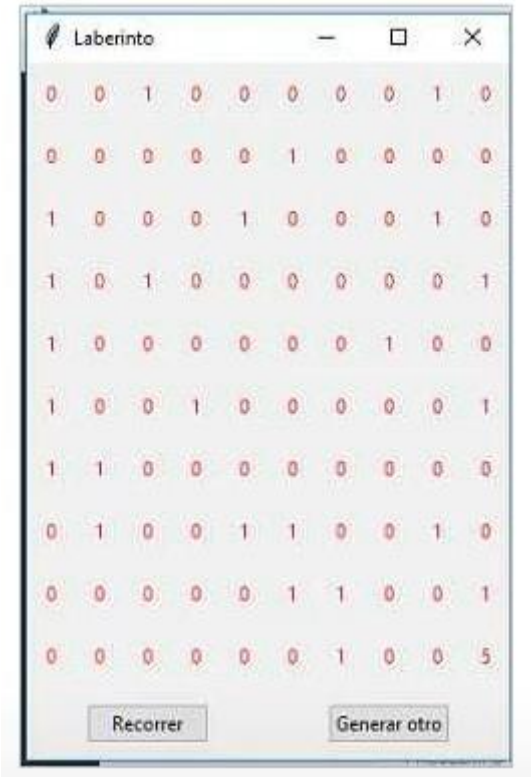
Para resolver este problema al laberinto lo representamos con una lista de 10 elementos de tipo lista que en cada una hay 10 Label.

El valor:

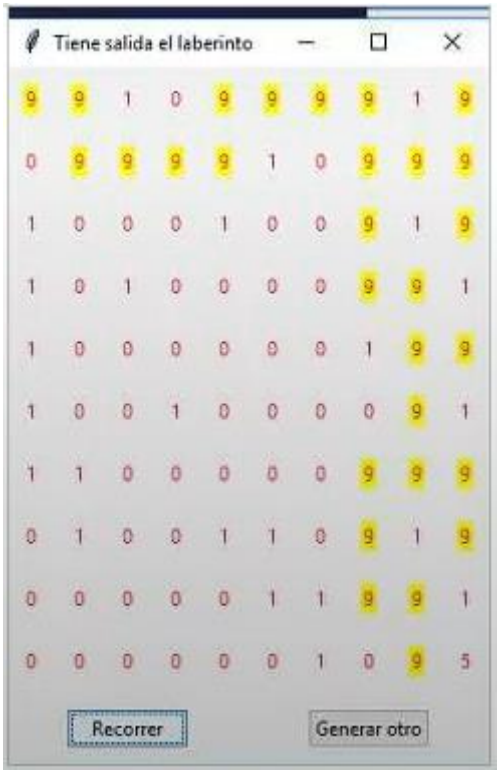
0 Representa pasillo.
1 Representa pared
9 Persona
5 Salida

A la salida ubicada en la componente de la fila 9 columna 9.
La persona comienza a recorrer el laberinto en la fila 0 y columna 0.
Los ceros y unos disponerlos en forma aleatoria.

Cuando ejecutamos el programa se genera un laberinto en forma aleatoria:



Cuando presionamos el botón "Recorrer" comenzamos a avanzar desde la fila 0 y columna 0:

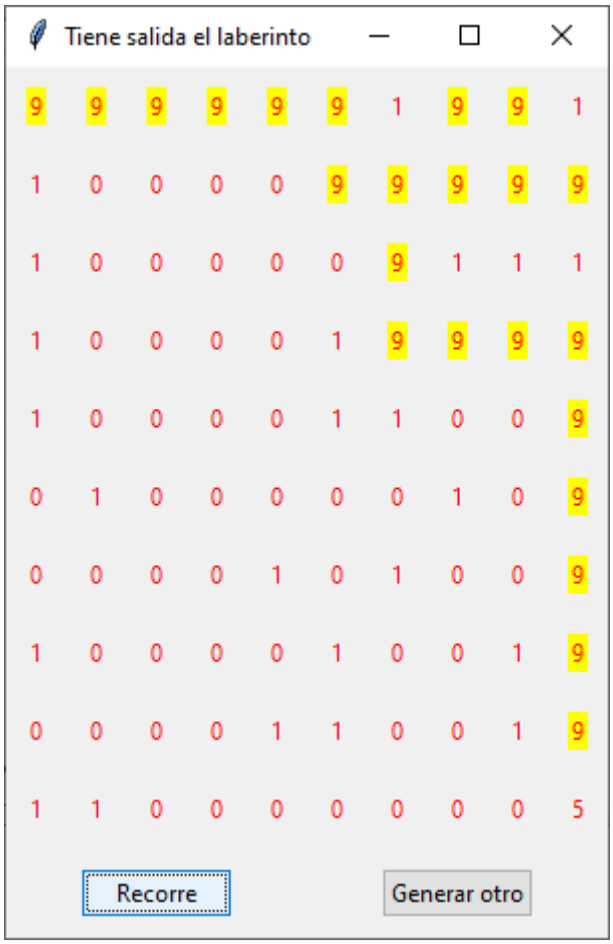


Vamos a programar:

```
import tkinter as tk
from tkinter import ttk
import random

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Laberinto")
        self.salida=False
        self.generar_laberinto()
        self.boton1=ttk.Button(self.ventana1, text="Recorre",
command=self.analizar)
        self.boton1.grid(column=0, row=10, columnspan=5, padx=10,
pady=10)
        self.boton2=ttk.Button(self.ventana1, text="Generar otro",
command=self.generar_otro)
        self.boton2.grid(column=5, row=10, columnspan=5, padx=10,
pady=10)
        self.ventana1.mainloop()

    def generar_laberinto(self):
        self.tablero=[]
        listafila=[]
        for fila in range(0,10):
            for columna in range(0,10):
```

```
                label=ttk.Label(self.ventana1,
text=self.retornar_aleatorio(), foreground="red")
                label.grid(column=columna, row=fila, padx=10, pady=10)
                listafila.append(label)
            self.tablero.append(listafila)
            listafila=[]
        self.tablero[0][0].configure(text=0)
        self.tablero[9][9].configure(text=5)

    def analizar(self):
        self.salida=False
        self.recorrer(0,0)
        if self.salida:
            self.ventana1.title("Tiene salida el laberinto")
        else:
            self.ventana1.title("No tiene salida el laberinto")

    def generar_otro(self):
        for fila in range(0,10):
            for columna in range(0,10):
                self.tablero[fila][columna].configure(text=self.retornar_
aleatorio(), background="#f0f0f0")
        self.tablero[0][0].configure(text=0)
        self.tablero[9][9].configure(text=5)

    def retornar_aleatorio(self):
        valor=random.randint(1,4)
        if valor==1:
            return 1
        else:
            return 0

    def recorrer(self, fil, col):
        if fil>=0 and fil<=10 and col>=0 and col<10 and
self.salida==False:
            if self.tablero[fil][col].cget("text")==5:
                self.salida=True
            else:
                if self.tablero[fil][col].cget("text")==0:
                    self.tablero[fil][col].configure(text=9)
                    self.tablero[fil][col].configure(background="yellow")
                    self.recorrer(fil,col+1)
                    self.recorrer(fil+1,col)
                    self.recorrer(fil-1, col)
                    self.recorrer(fil, col-1)

# Bloque principal
aplicacion1=Aplicacion()
```

Si ejecutamos este será el resultado:



Capítulo 296.- Recursividad: Problemas donde conviene aplicar la recursividad – 3

Problema:

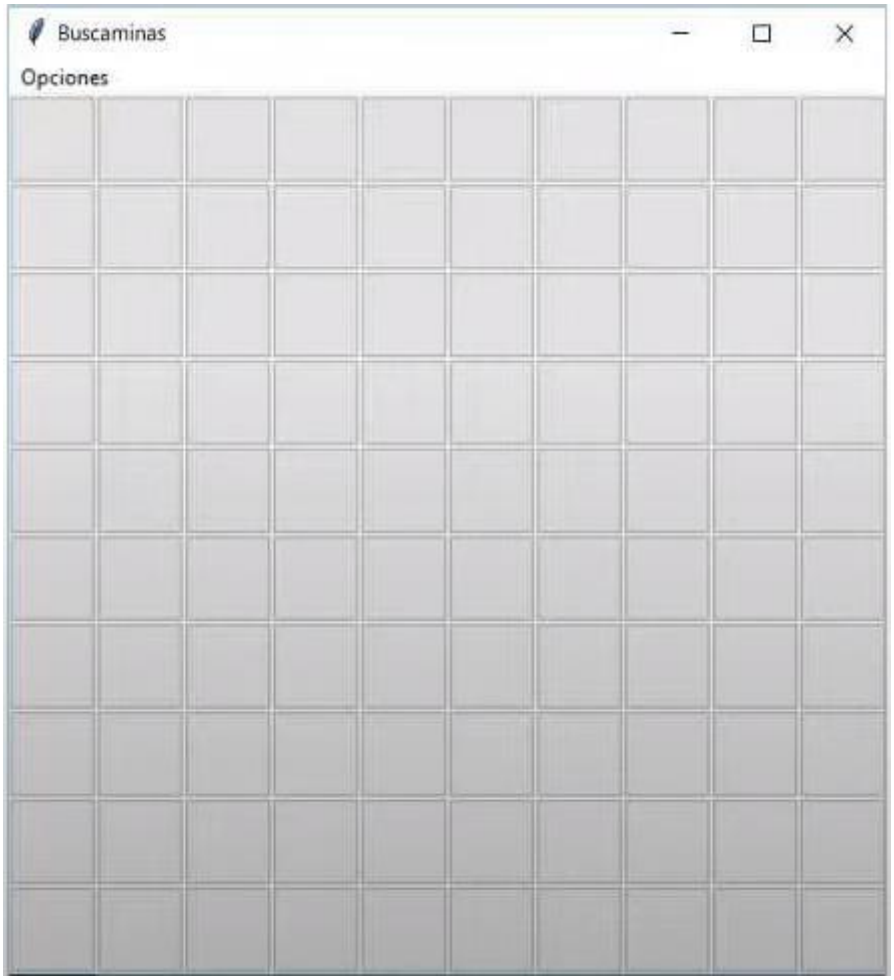
Desarrollar el juego del Buscaminas. Definir una lista de 10 elementos de tipo lista y en esta listas internas almacenar las referencias a botones.

El juego consiste en destapar todas las casillas que no tienen bombas. Si se presiona a la casilla que tiene bomba finaliza el juego. En las casillas que están en el perímetro de una bomba aparece un número que indica la cantidad de bombas a su alrededor. Por ejemplo si una casilla tiene el 2 significa que de las 8 casillas a su alrededor hay dos bombas.

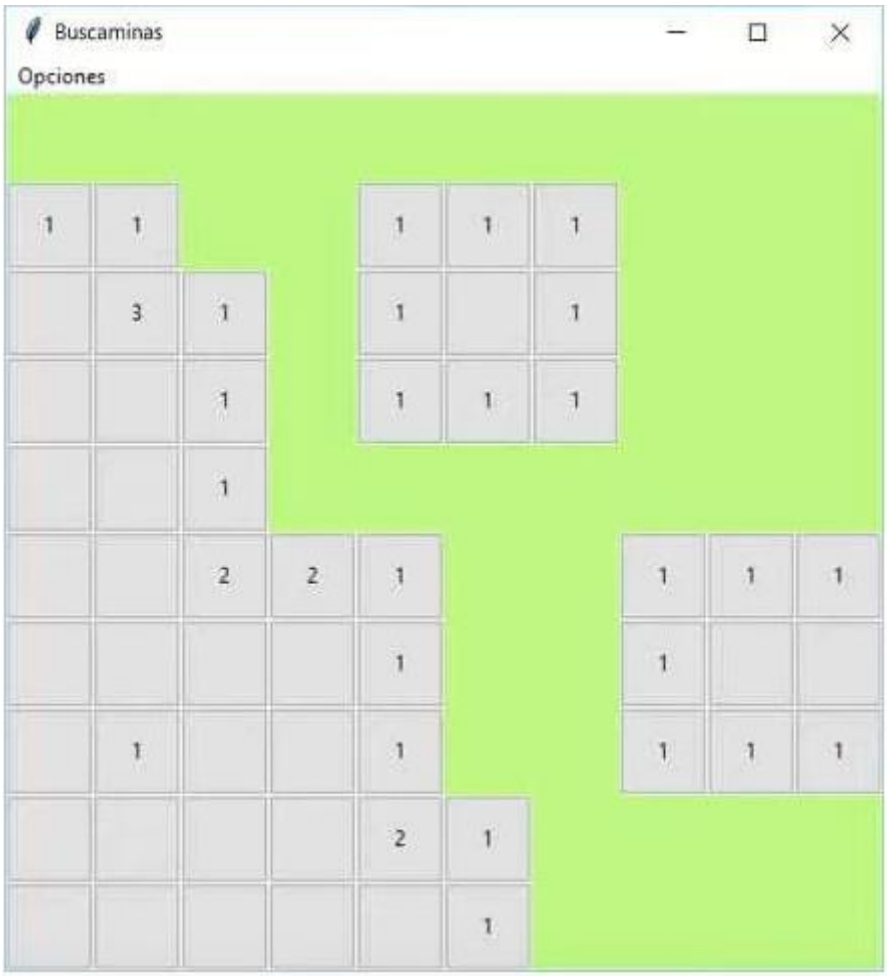
Si se selecciona una casilla que no tiene bomba a su alrededor se libera esta y todas las que se encuentran próximas a ella.

Permitir volver a jugar mediante un menú de opciones.

Cuando se inicia el juego debe aparecer el tablero con las 100 casillas:



Luego de destapar algunas casillas:



Vamos a programar:

```
import tkinter as tk
from tkinter import ttk
import random
from tkinter import messagebox as mb

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        # Si destapadas llega a 90 hemos ganado
        self.destapadas=0
        # Controlamos que el juego está activo
        self.enjuego=True
        self.ventana1.geometry("500x500")
        self.ventana1.title("Buscaminas")
        self.ventana1.configure(background="#BEF781")
        # Llamamos al método generar_tablero()
        self.generar_tablero()
        # Llamamos al método generar_bombas()
        self.generar_bombas()
        # Llamamos al método generar_bombas_proximas()
        self.generar_bombas_proximas()
```

```
        menubar1=tk.Menu(self.ventana1)
        self.ventana1.config(menu=menubar1)
        opciones1=tk.Menu(menubar1)
        opciones1.add_command(label="Reiniciar", command=self.reiniciar)
        opciones1.add_command(label="Salir",
command=self.ventana1.destroy)
        menubar1.add_cascade(label="Opciones", menu=opciones1)
        self.ventana1.mainloop()


    def generar_tablero(self):
        # Creamos la lista tablero que contendrá 10 listas
        # de la lista listafila.
        self.tablero=[]
        # listafilas en una variable local.
        listafila=[]
        # Con un ciclo for anidado creamos los 100 botones
        for fila in range(0,10):
            for columna in range(0,10):
                # Cada botón puede llamar al método  o funciones anónimas
                # Las funciones anónimas no tienen nombre
                boton=ttk.Button(self.ventana1, text="", command=lambda
fi=fila, co=columna: self.presion(fi,co))
                # El método place las medidas son en pixeles por eso lo
                # multiplicamos por 50 que es el ancho y el alto de cada
botón.
                boton.place(x=columna*50, y=fila*50, width=50, height=50)
                # Agregamos el boton a la lista listafila
                listafila.append(boton)
                # cuando la listafilas la agregamos a la lista tablero
            self.tablero.append(listafila)
            # Vaciamos los elementos de la listafila
            listafila=[]

    def generar_bombas(self):
        self.bombas=[]
        listafila=[]
        # Con este bucle anidado le decimos:
        for fila in range(0,10):
            for columna in range(0,10):
                # Añade a la lista listafila  10 valores con el valor 0
                listafila.append("0")
                # Añade a la lista bombas 10 listas de listafila
            self.bombas.append(listafila)
            # Vaciamos los elementos de la lista listafila.
            listafila=[]
        # El contador cantidad controlará el número de bombas
        cantidad=10
        while cantidad!=0:
```

```
        fila=random.randint(0,9)
        columna=random.randint(0,9)
        # Para evitar poner dos bombas en el mismo lugar
        if self.bombas[fila][columna]!="b":
            self.bombas[fila][columna]="b"
            # Para ver donde ha puesto las bombas
            # self.tablero[fila][columna].configure(text="b")
            cantidad=cantidad-1

def generar_bombas_proximas(self):
    for f in range(0,10):
        for c in range(0,10):
            if self.bombas[f][c]=="0":
                # Llama el método contar_lado
                # y retorna el número de bombas que tiene en
                # su perimetro.
                cant=self.contar_lado(f,c)
                # Gurdamos el número de bombas que hay
                # en su perímetro.
                self.bombas[f][c]=str(cant)

def contar_lado(self, fila, columna):
    # Hay 8 if son el perímetro de cada cuadrado.
    # Con la variable total contamos las bombas que ha en
    # el perímetro de cada botón que no tiene bomba.
    total=0
    # Si estamos dentro de la matriz
    if fila-1>=0 and columna-1>=0:
        if self.bombas[fila-1][columna-1]=="b":
            total=total+1
    # Si estamos dentro de la matriz
    if fila-1>=0:
        if self.bombas[fila-1][columna]=="b":
            total=total+1
    # Si estamos dentro de la matriz
    if fila-1>=0 and columna+1<10:
        if self.bombas[fila-1][columna+1]=="b":
            total=total+1
    # Si estamos dentro de la matriz
    if columna+1<10:
        if self.bombas[fila][columna+1]=="b":
            total=total+1
    # Si estamos dentro de la matriz
    if fila+1<10 and columna+1<10:
        if self.bombas[fila+1][columna+1]=="b":
            total=total+1
    # Si estamos dentro de la matriz
    if fila+1<10:
        if self.bombas[fila+1][columna]=="b":
```

```
                total=total+1
        # Si estamos dentro de la matriz
        if fila+1<10 and columna-1>=0:
            if self.bombas[fila+1][columna-1]=="b":
                total=total+1
        # Si estamos dentro de la matriz
        if columna-1>=0:
            if self.bombas[fila][columna-1]=="b":
                total=total+1
        # Retorna el número de bombas que tiene en su perímetro.
        return total

    def presion(self, fila, columna):
        # Verifica si estamos en juego
        if self.enjuego:
            # Si presionamos un botón que contiene la letra b
            # La variagble enjuego pasa a ser False y llama a la función
destapar()
            if self.bombas[fila][columna]=="b":
                self.enjuego=False
                # Llamamos al método destapar.
                self.destapar()
                mb.showinfo("Información", "Perdiste hay una bomba")
            else:
                # Si le damos a un botón que no tiene bomba
                if int(self.bombas[fila][columna])==0:
                    # Llamamos al método recorrer
                    # método recursivo.
                    self.recorrer(fila,columna)
                else:
                    if int(self.bombas[fila][columna])>=1 and
int(self.bombas[fila][columna])<=8 and
self.tablero[fila][columna].cget("text")==="":
                        self.tablero[fila][columna].configure(text=self.b
ombas[fila][columna])
                        self.destapadas=self.destapadas+1
            if self.destapadas==90:
                self.enjuego=False
                mb.showinfo("Información", "Ganaste")

    def recorrer(self, fil, col):
        # Verificamos que estamos dentro de las coordenadas
        if fil>=0 and fil<10 and col>=0 and col<10:
            # Verificamos si el valor es igual a 0 y además distinto a
vacio.
            if self.bombas[fil][col]=="0" and
self.tablero[fil][col]!=None:
                # Vaciamos de la matriz bomba las coordenadas [fil][col]
                self.bombas[fil][col]=" "
```

```
                # Contamos los botones que hemos destapado
                self.destapadas=self.destapadas+1
                # Destruye el botón y desaparece
                self.tablero[fil][col].destroy()
                # Le pasamos el valor None
                self.tablero[fil][col]=None
                # Recorre por todo el perímetro del botón
                self.recorrer (fil, col + 1)
                self.recorrer (fil, col - 1)
                self.recorrer (fil + 1, col)
                self.recorrer (fil - 1, col)
                self.recorrer (fil - 1, col - 1)
                self.recorrer (fil - 1, col + 1)
                self.recorrer (fil + 1, col + 1)
                self.recorrer (fil + 1, col - 1)
            else:
                # Si botón es distinto a vacio
                if self.tablero[fil][col]!=None:
                    # Si tiene alguna bomba en su perimetro y el botón no
muestra nada.
                    if int(self.bombas[fil][col])>=1 and
int(self.bombas[fil][col])<=8 and
self.tablero[fil][col].cget("text")==="":
                        # Que nos diga la cantidad de bombas que hay en
su perímetro.
                        self.tablero[fil][col].configure(text=self.bombas
[fil][col])
                        # Contabiliza el número de botones que destapa
                        self.destapadas=self.destapadas+1

    def reiniciar(self):
        self.destapadas=0
        self.eliminar_botones()
        self.generar_tablero()
        self.generar_bombas()
        self.generar_bombas_proximas()
        self.enjuego=True

    def eliminar_botones(self):
        for fila in range(0,10):
            for columna in range(0,10):
                if self.tablero[fila][columna]!=None:
                    self.tablero[fila][columna].destroy()
                    self.tablero[fila][columna]=None

    def destapar(self):
        # Recorre toda la matriz.
        for fila in range(0,10):
            for columna in range(0,10):
```

```
                # Si la matriz los valores distinto a vacio.
                if self.tablero[fila][columna]!=None:
                    # Si la matriz los valores son distintos a 0
                    if self.bombas[fila][columna]!="0":
                        # Muestrame donde están las bombas y los valores
                        # de bombas que hay en el perímetro.
                        self.tablero[fila][columna].configure(text=self.b
ombas[fila][columna])



# Bloque principal
aplicacion=Aplicacion()
```

Ya puedes ejecutar y jugar.

Capítulo 297.- Funciones de orden superior – 1

Python es un lenguaje multiparadigma: permite programación lineal, estructurada, orientada a objeto y también introduce características existentes en los lenguajes funcionales que nos dejan crear un código más claro y expresivo.

Una de las características del paradigma de la programación funcional son las funciones de orden superior.

Las funciones de orden superior son funciones que pueden recibir como parámetro otras funciones y/0 devolverlas como resultados.

Veremos una serie de ejemplos muy sencillos para ver como Python implementa el concepto de funciones de orden superior y a medida que avancemos en el curso podremos ver las ventajas de este paradigma.

Problema:

Definir una función de orden superior llamada operar. Llegan como parámetro dos enteros y una función. En el bloque de la función llamar a la función que llega como parámetro y enviar los dos primeros parámetros.

La función retorna un entero.

Vamos a programar.

```
def operar(v1, v2, fn):
    return fn(v1,v2)

def sumar(x1,x2):
    return x1+x2

def restar(x1,x2):
    return x1-x2

def multiplicar(x1,x2):
    return x1*x2

def dividir(x1,x2):
    return x1/x2

# Bloque principal
resu1=operar(10,4, sumar)
print(resu1)
print(sumar(10,4))

resu2=operar(10,4, restar)
print(resu2)
print(restar(10,4))

resu3=operar(10,4, multiplicar)
```

```
print(resu3)
print(multiplicar(10,4))

resu4=operar(10,4, dividir)
print(resu4)
print(dividir(10,4))
```

Este será el resultado:

```
14
14
6
6
40
40
2.5
2.5
```

Capítulo 298.- Funciones de orden superior – 2

Problema:

Declarar una clase que almacene el nombre y la edad de una persona. Definir un método que retorne True o False según si la persona es mayor de edad o no.

Este método debe recibir como parámetro una función que al llamarla pasando la edad de la persona retornará si es mayor o no de edad.

Tener en cuenta que una persona es mayor de edad en Estados Unidos si tiene 21 años o más años y en Argentina si tiene 18 o más años.

Vamos a programar:

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre=nombre
        self.edad=edad

    def es_mayor(self, fn):
        return fn(self.edad)

def mayor_estadosunidos(edad):
    if edad>=21:
        return True
    else:
        return False

def mayor_argentina(edad):
    if edad>=18:
        return True
    else:
        return False

persona1=Persona("Juan", 18)

if persona1.es_mayor(mayor_argentina):
    print(f"{persona1.nombre} es mayor si vive en Argentina")
else:
    print(f"{persona1.nombre} no es mayor si vive en Argentina")

if persona1.es_mayor(mayor_estadosunidos):
    print(f"{persona1.nombre} es mayor si vive en EEUU")
else:
    print(f"{persona1.nombre} no es mayor si vive en EEUU")
```

Este será el resultado:

```
Juan es mayor si vive en Argentina
Juan no es mayor si vive en EEUU
```

Capítulo 299.- Funciones lambda en Python – 1

Una función lambda es cuando enviamos a una función de orden superior directamente una función anónima.

Es más común enviar una expresión lambda en lugar de enviar la referencia de una función como vimos en el concepto anterior.

Problema

Definir una función de orden superior llamada operar. Llegan como parámetro dos enteros y una función. En el bloque de la función llamar la función que llega como parámetro y enviar los dos primeros parámetros.

Desde el bloque principal llamar a operar y enviar distintas expresiones lambda que permitan sumar, restar, multiplicar y dividir.

Vamos a programar.

ejercicio299.py > ...

```
1   def operar(v1,v2,fn):
2       return fn(v1,v2)
3
4   # Bloque principal
5   resu1=operar(10,4, lambda x1,x2:x1+x2)
6   print(resu1)
7
8   resu2=operar(10,20, lambda x1,x2:x1-x2)
9   print(resu2)
10
11  resu3=operar(5,4, lambda x1,x2:x1*x2)
12  print(resu3)
13
14  resu4=operar(10,2, lambda x1,x2:x1/x2)
15  print(resu4)
```

Este será el resultado:

```
14
-10
20
5.0
```

Capítulo 300.- Funciones lambda en Python – 2

Problema

Confeccionar una función de orden superior que reciba una lista que almacena valores enteros y una función con un parámetro entero y que retorne un bolean.

La función debe analizar cada elemento de la lista llamando a la función que recibe como parámetro, si retorna True se pasa a mostrar el elemento.

En el bloque principal definir una lista de enteros.

Imprimir de la lista:

- Los valore múltiplos de 2.
- Los valores múltiplos de 3 o 5.
- Los valores mayores o iguales a 50
- los valores comprendidos entre 1 y 50 o entre 70 y 100

Vamos a programar:

ejercicio300.py > ...

```
def imprimir_si(lista, fn):
    for elemento in lista:
        if fn(elemento):
            print(elemento)


# Bloque principal
lista1=[9,20,70,60,19]

print("Valores pares de la lista")
imprimir_si(lista1, lambda x:x%2==0)

print("Multiplos de 3 o de 5")
imprimir_si(lista1, lambda x:x%3==3 or x%5==0)

print("Valores mayores o iguales a 50")
imprimir_si(lista1, lambda x:x>=50)

print("Valores comprendidos entre 1 y 50 o 70 y 100")
imprimir_si(lista1, lambda x:x>=1 and x<=50 or x>=70 and x<=100)
```

Este será el resultado:

```
Valores pares de la lista
20
70
60
```

```
Multiplos de 3 o de 5
20
70
60
Valores mayores o iguales a 50
70
60
Valores comprendidos entre 1 y 50 o 70 y 100
9
20
70
19
```

Capítulo 301.- Funciones lambda en Python – 3

Problema

Confeccionar una función de orden superior que reciba un String y una función con un parámetro de tipo String que retorna un bolean.

La función debe analizar cada elemento del String llamando a la función que recibe como parámetro, retorna un True se agrega dicho carácter al String que se retornará.

En el bloque principal definir un String con una cadena cualquiera.

Llamar a la función de orden superior y pasar expresiones lambdas para filtrar y generar otro String con las siguientes instrucciones:

- Un string solo con las vocales.
- Un string solo con los caracteres en minúscula.
- Un string con todos los caracteres no alfabéticos.

Vamos a programar:

ejercicio301.py > ...

```
1   def filtrar(cadena,fn):
2       cad=""
3       for caracter in cadena:
4           if fn(caracter):
5               cad=cad+caracter
6       return cad
7
8   # Bloque principal
9   texto1="¿Esto es la prueba 1 o la prueba 2?"
10  print("Texto original")
11  print(texto1)
12
13  print("Un string con solo las vocales")
14  cadena1=filtrar(texto1, lambda c:c=='a' or c=='e' or c=='i' or c=='o' or c=='u'
15                  or c=='A' or c=='E' or c=='I' or c=='O' or c=='U')
16  print(cadena1)
17  print("Un string con los caracteres en minúsculas")
18  cadena2=filtrar(texto1, lambda c: c>='a' and c<='z')
19  print(cadena2)
20
21  print("Un string con todos los caracteres no alfabéticos")
22  cadena3=filtrar(texto1, lambda c: not(c>='a' and c<='z') and not(c>='A' and c<='Z'))
23  print(cadena3)
```

Este será el resultado:

```
Texto original
¿Esto es la prueba 1 o la prueba 2?
Un string con solo las vocales
Eoeaueaoauea
Un string con los caracteres en minúsculas
stoeslapruebaolaprueba
Un string con todos los caracteres no alfabéticos
¿    1    2?
```

El último resultado incluye los espacios en blanco.

Capítulo 302.- Comprensión de listas en Python – 1

La compresión de listas es una construcción en Python que nos permite crear listas a partir de otras listas, tuplas y cualquier iterable.

Nos permite escribir en forma más concisa un algoritmo. Veamos con un ejemplo como lo resolvemos con y sin comprensión de listas.

Esta instrucción para el recorrido de iterables se ejecutará más rápido que la forma tradicional. Requiere un poco de tiempo para adaptar nuestro razonamiento a la nueva sintaxis.

Problema 1

Definir una lista de 5 valores enteros, luego a partir de la primera lista generar una segunda lista con los valores elevados al cuadrado.

ejercicio302.py > ...

```
1   lista1=[2,3,4,5,6]
2   lista2=[]
3   for elemento in lista1:
4       lista2.append(elemento*elemento)
5   print(lista1)
6   print(lista2)
```

Este será el resultado:

```
[2, 3, 4, 5, 6]
[4, 9, 16, 25, 36]
```

Otra forma de hacerlo con el método compresión de listas:

ejercicio302.py > ...

```
1   lista1=[2,3,4,5,6]
2   lista2=[elemento*elemento for elemento in lista1]
3   print(lista1)
4   print(lista2)
```

El resultado será el mismo.

Problema 2

Se tiene una lista con un conjunto de tuplas con los nombres y edades de las personas:

personas=[('Pedro',33), ('Ana',3), ('Juan',13), ('Carla',45)]

Generar una lista con las personas mayores de edad.

ejercicio302.py > ...

```
1   personas1=[('Pedro',33), ('Ana',3), ('Juan',13), ('Carla',45)]
2   personas2=[]
3   for elemento in personas1:
4       if elemento[1]>=18:
```

```
5            personas2.append(elemento)
6    print(personas1)
7    print(personas2)
```

El resultado será:

```
[('Pedro', 33), ('Ana', 3), ('Juan', 13), ('Carla', 45)]
[('Pedro', 33), ('Carla', 45)]
```

Ahora con el método compresión de listas:

```
ejercicio302.py > ...
1    personas1=[('Pedro',33), ('Ana',3), ('Juan',13), ('Carla',45)]
2    personas2=[elemento for elemento in personas1 if elemento[1]>=18]
3    print(personas1)
4    print(personas2)
```

El resultado será el mismo

Problema 3

Generar una lista con todos los valores múltiplos de 8 comprendidos entre 1 y 500.

```
ejercicio302.py > ...
1    lista=[]
2    for x in range(1,500):
3        if x%8==0:
4            lista.append(x)
5    print(lista)
```

Este será el resultado:

```
[8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112, 120, 128, 136, 144, 152, 160, 168, 176, 184
, 192, 200, 208, 216, 224, 232, 240, 248, 256, 264, 272, 280, 288, 296, 304, 312, 320, 328, 336, 344,
 352, 360, 368, 376, 384, 392, 400, 408, 416, 424, 432, 440, 448, 456, 464, 472, 480, 488, 496]
```

Ahora con el método compresión de listas:

```
ejercicio302.py > ...
1    lista=[elemento for elemento in range(1,500) if elemento%8==0]
2    print(lista)
```

El resultado será el mismo.

Otra forma de hacerlo:

```
ejercicio302.py > ...
1    lista=[elemento for elemento in range(8,500,8)]
2    print(lista)
```

Capítulo 303.- Comprensión de listas en Python – 2

Varios for

Podemos utilizar más de un for cuando empleamos las características de 'List Comprehensions' de Python.

Problema

Se tiene una lista de nombres de personas:

nombres=['Juan', 'Pablo', 'Luis', 'Mauro', 'Hector']

Generar una lista cuyos elementos sean a su vez listas con dos nombres cada uno.

Tener en cuenta que nunca se debe combinar el mismo nombre dos veces.

Vamos a programar:

ejercicio303.py > ...

```
1   nombres=['Juan', 'Pablo', 'Luis', 'Mauro', 'Hector']
2   nombres_compuestos=[]
3   for nombres1 in nombres:
4       for nombres2 in nombres:
5           if nombres1!=nombres2:
6               nombres_compuestos.append([nombres1, nombres2])
7
8   print(nombres_compuestos)
```

Este será el resultado:

```
[['Juan', 'Pablo'], ['Juan', 'Luis'], ['Juan', 'Mauro'], ['Juan', 'Hector'], ['Pablo', 'Juan'], ['Pab
lo', 'Luis'], ['Pablo', 'Mauro'], ['Pablo', 'Hector'], ['Luis', 'Juan'], ['Luis', 'Pablo'], ['Luis',
'Mauro'], ['Luis', 'Hector'], ['Mauro', 'Juan'], ['Mauro', 'Pablo'], ['Mauro', 'Luis'], ['Mauro', 'He
ctor'], ['Hector', 'Juan'], ['Hector', 'Pablo'], ['Hector', 'Luis'], ['Hector', 'Mauro']]
```

Ahora con el método compresión de listas:

```
nombres=['Juan', 'Pablo', 'Luis', 'Mauro', 'Hector']
nombres_compuestos=[[nombres1,nombres2] for nombres1 in nombres for
nombres2 in nombres if nombres1!=nombres2]
print(nombres_compuestos)
```

El resultado será el mismo.

```
[['Juan', 'Pablo'], ['Juan', 'Luis'], ['Juan', 'Mauro'], ['Juan', 'Hector'], ['Pablo', 'Juan'], ['Pab
lo', 'Luis'], ['Pablo', 'Mauro'], ['Pablo', 'Hector'], ['Luis', 'Juan'], ['Luis', 'Pablo'], ['Luis',
'Mauro'], ['Luis', 'Hector'], ['Mauro', 'Juan'], ['Mauro', 'Pablo'], ['Mauro', 'Luis'], ['Mauro', 'He
ctor'], ['Hector', 'Juan'], ['Hector', 'Pablo'], ['Hector', 'Luis'], ['Hector', 'Mauro']]
```

Capítulo 304.- Comprensión de listas en Python - 3

Problema

Codificar un programa que muestre en pantalla los números del 1 al 100, sustituyendo los múltiplos de 3 por la palabra "Fizz" y a su vez, los múltiplos de 5 por la palabra "Buzz", para los múltiplos que al tiempo son múltiples de 3 y 5 mostrar el mensaje "FizzBuzz".

La lista debe ser similar a:

1	26	Fizz	76
2	Fizz	52	77
Fizz	28	53	Fizz
4	29	Fizz	79
Buzz	FizzBuzz	Buzz	Buzz
Fizz	31	56	Fizz
7	32	Fizz	82
8	Fizz	58	83
Fizz	34	59	Fizz
Buzz	Buzz	FizzBuzz	Buzz
11	Fizz	61	86
Fizz	37	62	Fizz
13	38	Fizz	88
14	Fizz	64	89
FizzBuzz	Buzz	Buzz	FizzBuzz
16	41	Fizz	91
17	Fizz	67	92
Fizz	43	68	Fizz
19	44	Fizz	94
Buzz	FizzBuzz	Buzz	Buzz
Fizz	46	71	Fizz
22	47	Fizz	97
23	Fizz	73	98
Fizz	49	74	Fizz
Buzz	Buzz	FizzBuzz	Buzz

Vamos a programar:

ejercicio304.py > ...

```
1  for x in range(1,101):
2      if x%3==0 and x%5==0:
3          print("FizzBuzz")
4      elif(x%3==0):
5              print("Fizz")
6      elif(x%5==0):
7              print("Buzz")
```

```
8        else:
9                print(x)
```

Ahora con el método compresión de listas:

```
print(["Fizz"*(not numero%3)+ "Buzz"*(not numero%5) or numero for numero
in range(1,101)])
```

El resultado será:

```
[1, 2, 'Fizz', 4, 'Buzz', 'Fizz', 7, 8, 'Fizz', 'Buzz', 11, 'Fizz', 13, 14, 'FizzBuzz', 16, 17, 'Fizz
', 19, 'Buzz', 'Fizz', 22, 23, 'Fizz', 'Buzz', 26, 'Fizz', 28, 29, 'FizzBuzz', 31, 32, 'Fizz', 34, 'B
uzz', 'Fizz', 37, 38, 'Fizz', 'Buzz', 41, 'Fizz', 43, 44, 'FizzBuzz', 46, 47, 'Fizz', 49, 'Buzz', 'Fi
zz', 52, 53, 'Fizz', 'Buzz', 56, 'Fizz', 58, 59, 'FizzBuzz', 61, 62, 'Fizz', 64, 'Buzz', 'Fizz', 67,
68, 'Fizz', 'Buzz', 71, 'Fizz', 73, 74, 'FizzBuzz', 76, 77, 'Fizz', 79, 'Buzz', 'Fizz', 82, 83, 'Fizz
', 'Buzz', 86, 'Fizz', 88, 89, 'FizzBuzz', 91, 92, 'Fizz', 94, 'Buzz', 'Fizz', 97, 98, 'Fizz', 'Buzz'
]
```

numer0%3 significa que cuando número es igual a 3 el 3%3 retorna 0 al agregar not este 0 se convierte en 1 que multiplicamos por Fizz genera esta cadena, lo mismo pasa con (not numero%5) que imprimirá Buzz, si no es divisible por tres ni por 5 entonces se imprime el número.

No es una forma fácil de leer.

Capítulo 305.- Estructura de datos tipo conjunto: set – 1

Hemos visto en conceptos anteriores las estructuras de datos que vienen por defecto en Python y hacen a la esencia de este lenguaje:

- Listas
- Tuplas
- Diccionarios

La cuarta estructura de datos que presentaremos se llama conjunto (set), esta estructura de datos permite almacenar una colección de elementos que no se pueden repetir y no se encuentran ordenados.

Creación de un conjunto de asignación

Para crear un conjunto de asignación debemos indicar sus elementos encerrados entre llaves y separados por coma.

```
conjunto1={1, 5, 10, 20}
print(conjunto1)
```

Los elementos de un conjunto pueden ser de distinto tipo:

```
conjunto2={"Juan", 20, 6.4, True}
print(conjunto2)
```

Podemos definir elementos de un conjunto de tipo tupla pero no de tipo lista, diccionario o conjunto:

```
conjunto3={("Juan", 33), ("Ana", 44)}
print(conjunto3)
```

Agregar y eliminar elementos de un conjunto

Mediante el método add agregamos un elemento a un conjunto, si el valor ya existe luego no se inserta:

```
lenguajes={"C", "Pascal", "PHP", "Python"}
print(lenguajes)    # {"C", "Pascal", "PHP", "Python"}
lenguajes.add("Ruby")
print(lenguajes)    # {"C", "Pascal", "PHP", "Python", "Ruby"}
lenguajes.add("PHP")
print(lenguajes)    # {"C", "Pascal", "PHP", "Python", "Ruby"}
```

El conjunto "lenguajes" se crea con 4 elementos. Luego se agrega un quinto elemento:

```
lenguajes.add("Ruby")
```

Si intentamos agregar un elemento que ya existe el mismo no se agrega y no genera error:

```
lenguajes.add("PHP")
```

Para eliminar un elemento de un conjunto debemos llamar al método "remove":

```
lenguajes={"C", "Pascal", "PHP", "Python"}
print(lenguajes)
```

```
lenguajes.remove("Pascal")
print(lenguajes)    # {"C", "PHP", "Python", "Ruby"}
```

Si el método remove no encuentra el elemento dentro del conjunto genera una excepción de tipo "KeyError".
Si queremos eliminar un elemento del conjunto y que no genere error si no existe debemos hacer uso del método "discard":

```
lenguajes={"C", "Pascal", "PHP", "Python"}
print(lenguajes)    # {"C", "Pascal", "PHP", "Python"}
lenguajes.discard("Kotlin")
print(lenguajes)
```

Operación de pertenecia 'in'.

Si queremos verificar si un conjunto contiene un determinado valor podemos utilizar el operador 'int':

```
lenguajes={"C", "Pascal", "PHP", "Python"}
if "PHP" in lenguajes:
    print("El lenguaje PHP se encuentra en el conjunto")
else:
    print("El lenguaje PHP no se encuentra en el conjunto")
```

La estructura condicional if se verifica verdadero debido a que la cadena 'PHP' se encuentra contenido en el conjunto "Lenguajes".

Ejemplo práctico:

ejercicio305.py > ...

```
1   lenguajes={"C", "Pascal", "PHP", "Python"}
2   print(lenguajes)
3   try:
4       lenguajes.remove("Lotlin")
5   except KeyError:
6       print("No existe el elemento Kotlin en el conjunto")
7   print(lenguajes)
```

Cuando ejecutemos este será el resultado:

```
{'Pascal', 'Python', 'C', 'PHP'}
No existe el elemento Kotlin en el conjunto
{'Pascal', 'Python', 'C', 'PHP'}
```

Vamos a eliminar e lenguaje "C".

ejercicio305.py > ...

```
1   lenguajes={"C", "Pascal", "PHP", "Python"}
2   print(lenguajes)
3   try:
4       lenguajes.remove("C")
```

```
5   except KeyError:
6       print("No existe el elemento Kotlin en el conjunto")
7   print(lenguajes)
```

Cuando ejecutemos este será el resultado:

```
{'PHP', 'Python', 'Pascal', 'C'}
{'PHP', 'Python', 'Pascal'}
```

Acotaciones.

- Para conocer la cantidad de elementos de un conjunto disponemos de la función 'len':

```
lenguajes={"C", "Pascal", "PHP", "Python"}
print(len(lenguajes))   # imprime un 4.
```

- Para crear un conjunto vacío debemos llamar a la función 'set':

```
lenguajes=set()
lenguajes.add("C")
lenguajes.add("Pascal")
lenguajes.add("PHP")
lenguajes.add("Python")
print(len(lenguajes))   # imprime un 4.
```

Es importante recordar que en Python las llaves abiertas y cerradas crean un diccionario y no un conjunto:

```
productos={}
productos["manzanas"]=39
productos["peras"]=32
productos["lechuga"]=17
print(productos)
```

Resultado:

```
{'manzanas':39, 'peras': 32, 'lechuga': 17}
```

- Un uso común de los conjuntos es eliminar los valores repetitivos de listas y tuplas:

```
edades=[23, 21, 25, 21, 23]
print(edades)
conjunto=set(edades)
print(conjunto)   #    25, 21, 23
```

Se crea un conjunto llamando a set y pasando como parámetro una lista, luego el conjunto no almacena los valores repetidos en la lista.

- Podemos iterar un conjunto mediante la estructura repetitiva for:

```
días={"lunes", "martes", "miércoles"}
for dia in días:
    print(dia)
```

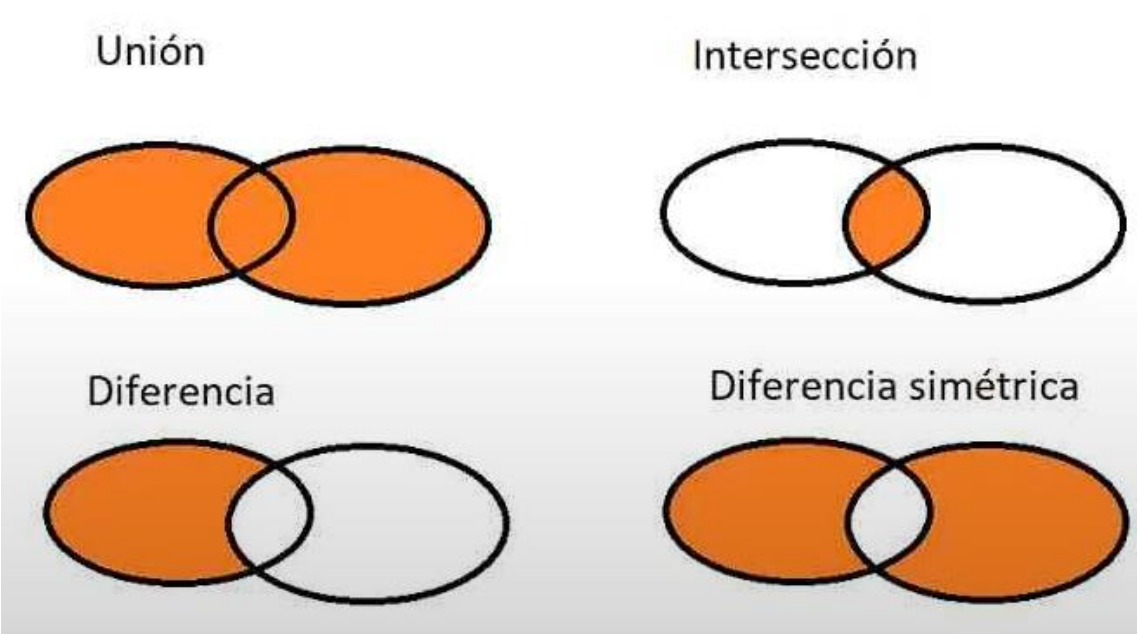
Capítulo 306.- Estructura de datos tipo conjunto: set – 2

Operaciones con conjuntos.

Las operaciones básicas con conjuntos son:

- Unión
- Intersección
- Diferencia
- Diferencia simétrica

Gráficamente estas operaciones con dos conjuntos las podemos representar:



Problema 1

Definir dos conjuntos que almacenen cada uno una serie de lenguajes de programación. Efectuar las cuatros operaciones básicas con dichos conjuntos.

```
lenguajes1={"C", "Pascal", "PHP", "Python"}
lenguajes2={"C++", "Java", "Python"}
print("Lenguajes estructurados")
print(lenguajes1)
print("Lenguajes orientados a objetos")
print(lenguajes2)
conjunto1=lenguajes1 | lenguajes2
print("Todos los lenguajes (Unión)")
print(conjunto1)
conjunto2=lenguajes1 & lenguajes2
print("Intersección de los dos conjuntos de lenguajes (Intersección)")
print(conjunto2)
conjunto3=lenguajes1 - lenguajes2
print("Diferencia de los dos conjuntos de lenguajes (Diferencia)")
print(conjunto3)
conjunto4=lenguajes1 ^ lenguajes2
```

```
print("lenguajes del conjunto lenguajes1 o del conjunto lenguajes2 pero
no en ambos (Diferencia simétrica)")
print(conjunto4)
```

Vamos a ejecutar y este será el resultado:

Lenguajes estructurados
{'C', 'Pascal', 'PHP', 'Python'}
Lenguajes orientados a objetos
{'Java', 'C++', 'Python'}
Todos los lenguajes (Unión)
{'Pascal', 'PHP', 'Python', 'C', 'Java', 'C++'}
Intersección de los dos conjuntos de lenguajes (Intersección)
{'Python'}
Diferencia de los dos conjuntos de lenguajes (Diferencia)
{'C', 'Pascal', 'PHP'}
lenguajes del conjunto lenguajes1 o del conjunto lenguajes2 pero no en ambos (Diferencia simétrica)
{'Java', 'Pascal', 'PHP', 'C', 'C++'}

Los operadores de conjuntos son:

| (Unión)
& (Intersección)
- (Diferencia)
^ (Diferencia simétrica)

En lugar de utilizar los operadores de conjuntos podemos llamar a sus métodos y obtener el mismo resultado:

```
lenguajes1={"C", "Pascal", "PHP", "Python"}
lenguajes2={"C++", "Java", "Python"}
print("Lenguajes estructurados")
print(lenguajes1)
print("Lenguajes orientados a objetos")
print(lenguajes2)
x=lenguajes1.union(lenguajes2)
print("Todos los lenguajes")
print(x)
x=lenguajes1.intersection(lenguajes2)
print("Intersección de los dos conjuntos de lenguajes")
print(x)
x=lenguajes1.difference(lenguajes2)
print("Diferencia de los dos conjuntos de lenguajes")
print(x)
x=lenguajes1.symmetric_difference(lenguajes2)
print("lenguajes del conjunto lenguajes1 o del conjunto lenguajes2 pero
no en ambos")
print(x)
```

Conjuntos disjuntos.

Dos conjuntos son disjuntos si no tienen elementos en común entre ellos. Su intersección es el conjunto vacío.

La clase set dispone del método 'isdisjoint' que retorna True si los conjuntos no tienen elementos en común:

```
días_festivos={"sábado", "domingo"}
días_laborables={"lunes", "martes", "miércoles", "jueves", "viernes"}
if días_laborables.isdisjoint(dias_festivos):
    print("dias_laborables no tienen elementos en común con días_festiovos")
```

Estos dos conjuntos no tienen elementos en común, luego el if se verifica verdadero.

Igualdad de conjuntos, subconjuntos y superconjuntos.

En Python podemos utilizar los operadores:

- conunto1==conjunto2
- conunto1!=conjunto2
- conjunto1<conjunto2 (si el conjunto1 es un subconjunto de conjunto2
- conjunto1<=conjunto2 (si el conjunto1 es un subconjunto o es igual que conjunto2)
- conjunto1>conjunto2 (si el conjunto1 es un superconjunto de conjunto2)
- conjunto1>=conjunto2 (si el conjunto1 es un superconjunto o es igual que conjunto2)

Algunos ejemplos empleando estos operadores:

```
dias_semana={"lunes", "martes",
"miércoles","jueves","viernes","sábado","domingo"}
dias_festivos={"sábado","domingo"}
dias_laborables={"lunes", "martes", "miércoles","jueves","viernes"}
if dias_festivos<dias_semana:
    print("dias_festivos es un subconjunto de dias_semana")
if dias_festivos!=dias_laborables:
    print("dias_festivos es distinto a dias_laborables")
if dias_semana>dias_laborables:
    print("dias_semana es un superconjunto de dias_laborables")
```

Este será el resultado:

```
dias_festivos es un subconjunto de dias_semana
dias_festivos es distinto a dias_laborables
dias_semana es un superconjunto de dias_laborables
```

Capítulo 307.- Estructura de datos tipo conjunto: set – 3

Conjuntos congelados 'frozenset'

Hay otra clase llamada 'frozenset' que permite crear conjuntos inmutables, es decir que no s le pueden agregar o eliminar elementos una vez creado.

```
vocales=frozenset({'a', 'e', 'i', 'o', 'u'})
print(type(vocales)
if 'a' in vocales:
    print("Si contiene la letra a")
```

Problema

Realizar la carga de valores enteros y sumarlos, cada vez que ingresa un valor preguntar al operador si desea ingresar otro valor.

Vamos a programar:

ejercicio307.py > ...

```
1  opciones_salir=frozenset(["no","No"])
2  suma=0
3  while True:
4      valor=int(input("Ingrese un valor:"))
5      suma=suma+valor
6      op=input("Desea ingresar otro valor [si/no]:")
7      if op in opciones_salir:
8          break
9  print(f"La suma de los valores es {suma}")
```

Este será el resultado:

```
Ingrese un valor:1
Desea ingresar otro valor [si/no]:si
Ingrese un valor:2
Desea ingresar otro valor [si/no]:si
Ingrese un valor:3
Desea ingresar otro valor [si/no]:no
La suma de los valores es 6
```

Con frozenset evitamos poder agregar nuevos elementos.

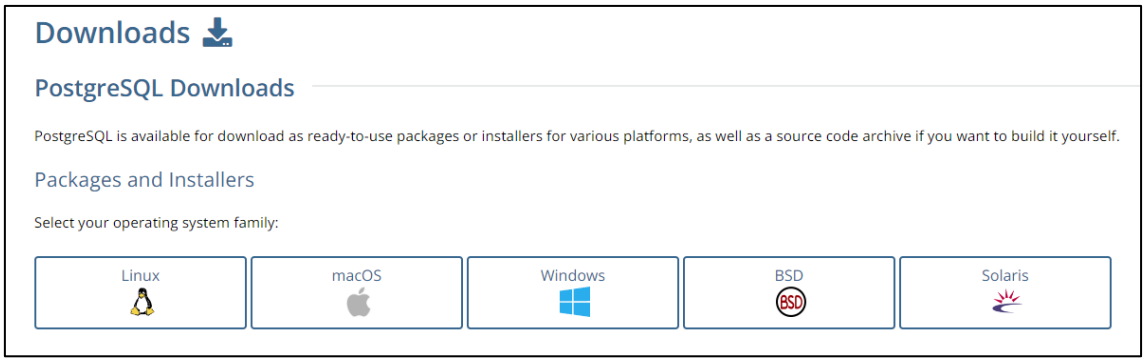
Capítulo 308.- PostgresSQL: Base de datos desde Python

La tercera base de datos que veremos su acceso desde Python es PostgresSQL.

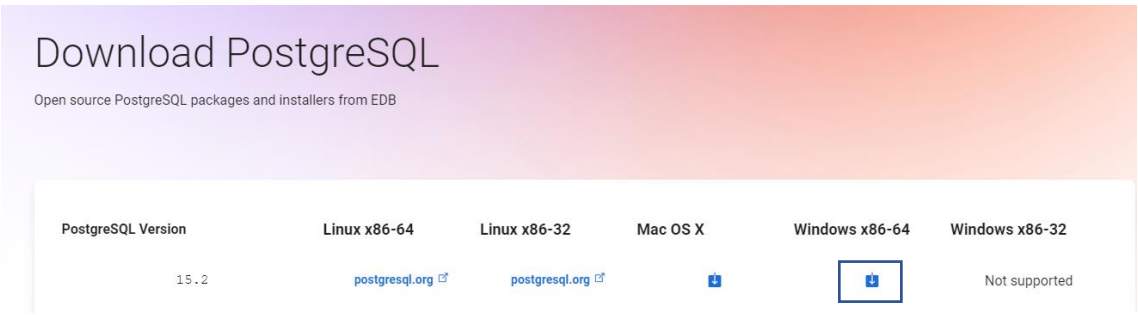
Tenga en cuenta que debe tener instalado el motor de base de datos PostgresSQL para poder seguir este concepto. En el tutorial de PostgressSQL están todos los pasos para su instalación.

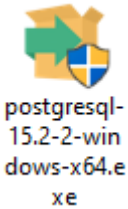
El primer paso será descargar el PostgreSQL:

Acedemos al siguiente enlace: https://www.postgresql.org/download/



Descargamos el instalador.

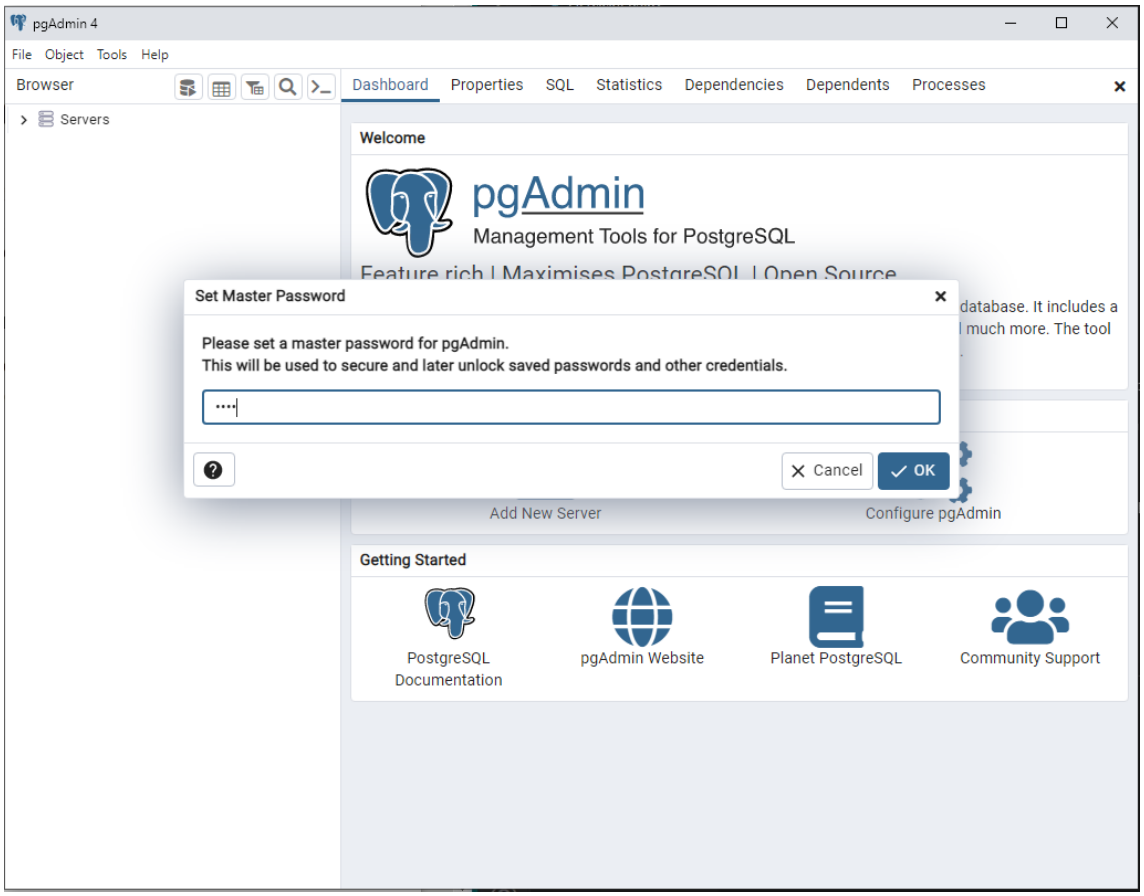




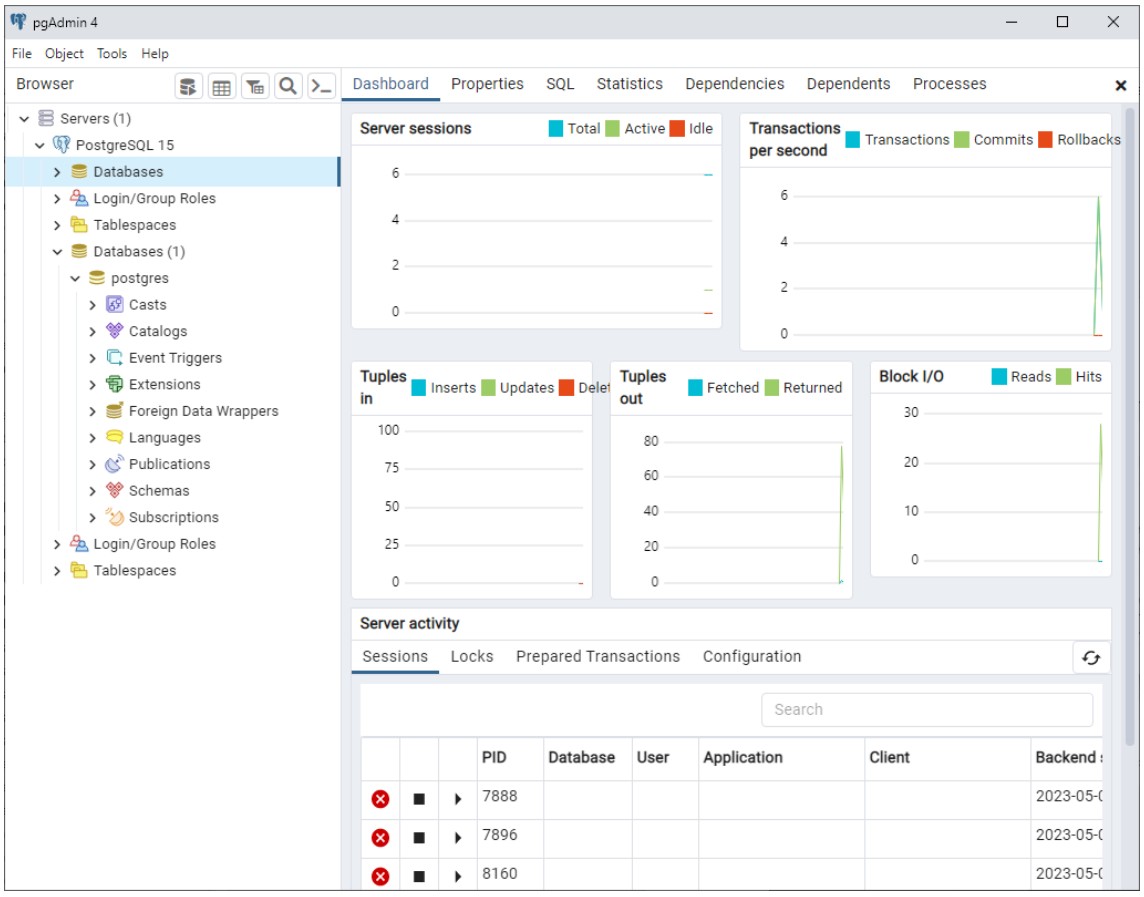
Ejecutamos el archivo que hemos descargado y lo instalamos.

Vamos a ejecutar pgAdmin4.

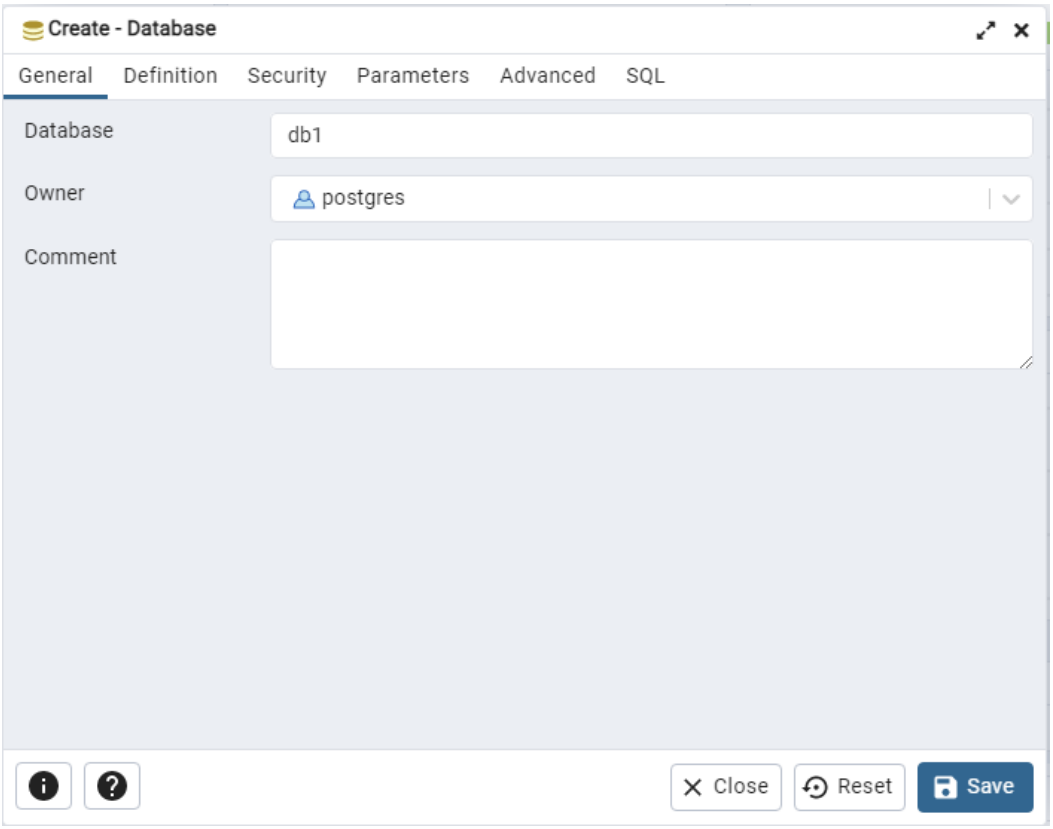
Se abrirá el navegador con la aplicación.



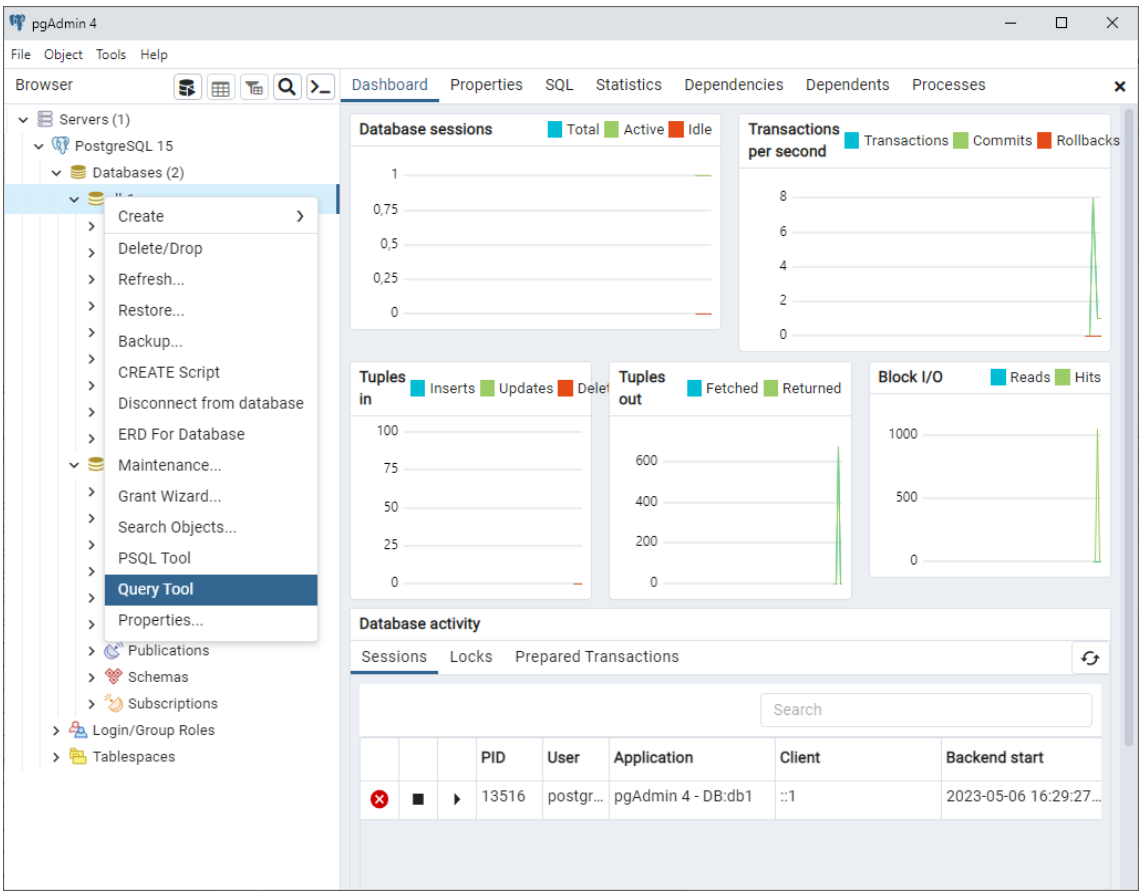
Me pide al password que puse cuando lo estaba instalando.



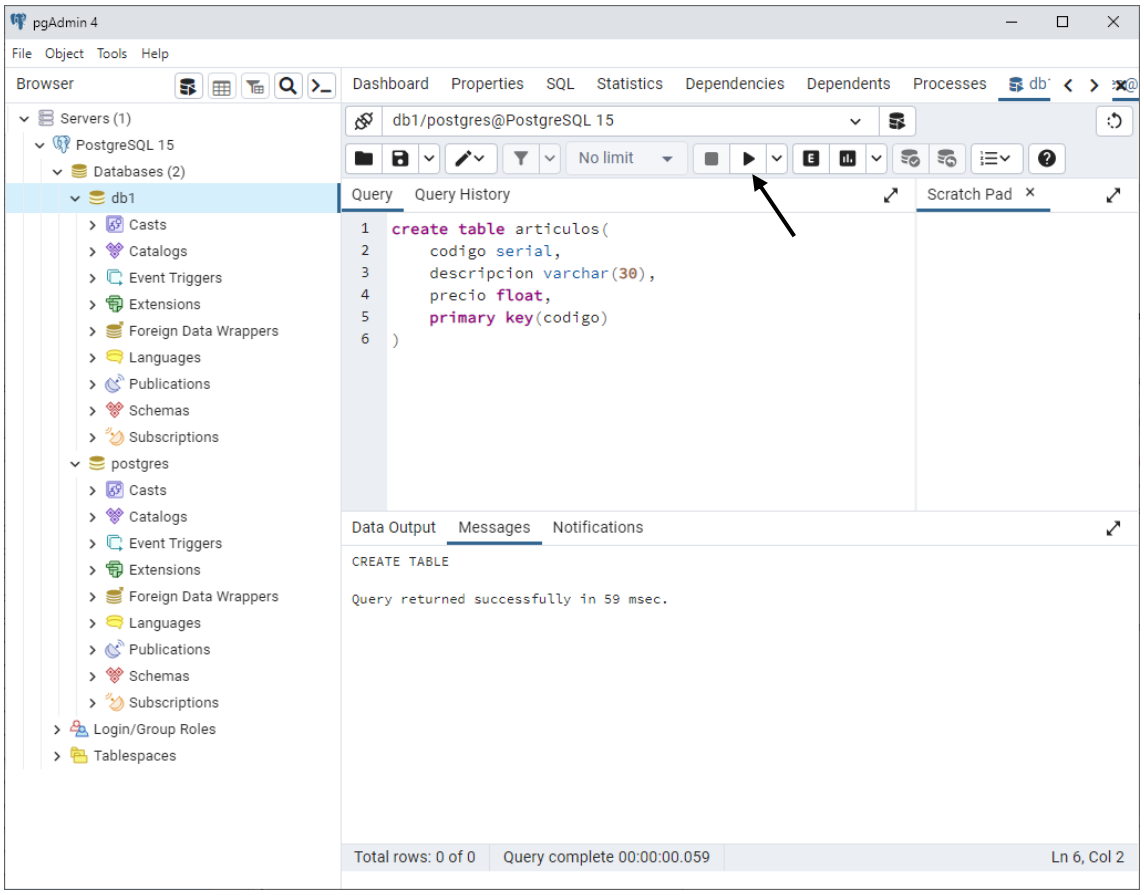
Seleccionamos Database con el botón derecho del ratón y creamos una nueva base de datos.



Le pondremos de nombre db1 seguido del botón Save.

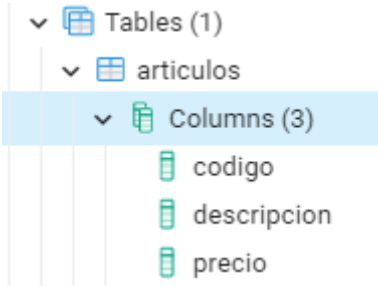


Botón derecho sobre la base de datos que hemos creado y seleccionamos Query Tool.



Creamos al tabla y la ejecutamos.

En la base de datos que hemos creado seleccionamos Shemas, de este public, tables y veremos la tabla articulos con sus correspodiente campos:



Nos vamos a nuestro entorno de programación.

Paquete de Python necesario para conectarnos a PostgreSQL.

Utilizaremos el programa 'pip' que vimos anteriormente para instalar el paquete necesario para interconectar 'Python' y 'PostgreSQL'.

Desde la línea de comandos ejecutaremos el programa pip con el siguiente paquete a instalar:

pip install psycopg2

```
PS D:\CursocompletodePython> pip install psycopg2
Collecting psycopg2
  Downloading psycopg2-2.9.6-cp311-cp311-win_amd64.whl (1.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 14.6 MB/s eta 0:00:00
Installing collected packages: psycopg2
Successfully installed psycopg2-2.9.6
PS D:\CursocompletodePython>
```

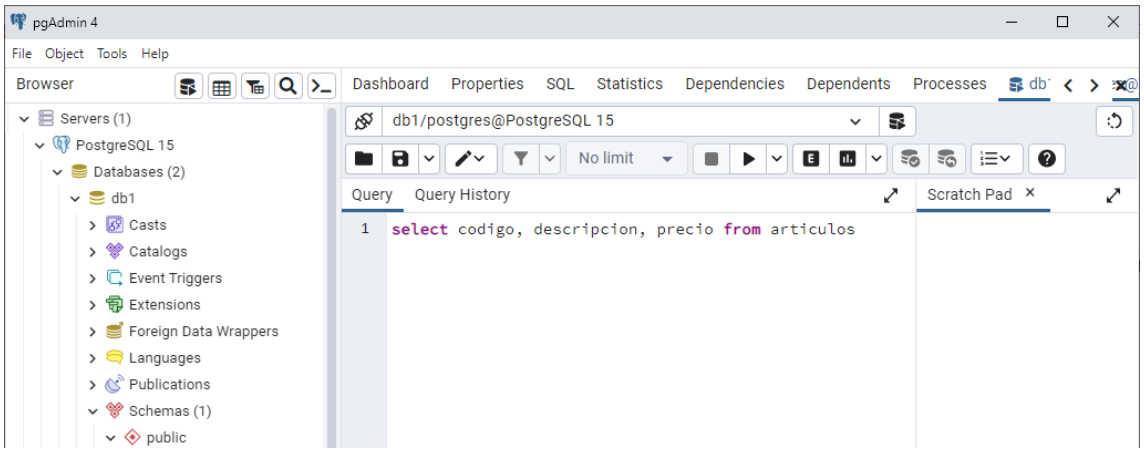
Conexión con el servidor de PostgreSQL

El primer programa que implementaremos nos conectaremos con el servidor de PostgreSQL e insertaremos un par de filas en la tabla 'articulos' que creamos desde el programa 'pgAdmin'.

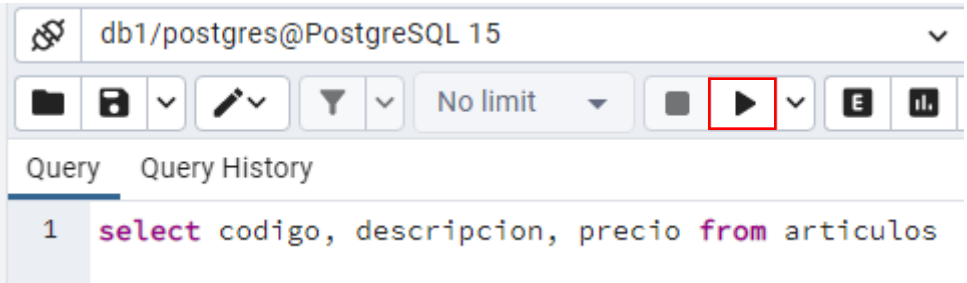
Programa:

```
ejercicio308.py > ...
1   import psycopg2
2
3   conexion1=psycopg2.connect(database="db1", user="postgres", password="123456")
4   cursor1=conexion1.cursor()
5   sql="insert into articulos(descripcion, precio) values (%s,%s)"
6   datos=("naranjas", 23.50)
7   cursor1.execute(sql, datos)
8   datos=("peras", 34)
9   cursor1.execute(sql, datos)
10  datos=("bananas", 25)
11  cursor1.execute(sql, datos)
12  conexion1.commit()
13  conexion1.close()
```

El password tienes que poner el que pusiste cuando instalabas el programa.



Escribimos la consulta y ejecutamos.



Este será el resultado:

	codigo [PK] integer	descripcion character varying (30)	precio double precision
1	1	naranjas	23.5
2	2	peras	34
3	3	bananas	25

Recuperar todas las filas de una tabla

Implementamos un programa que solicite ejecutar un 'select' en la tabla 'articulos' de la base de datos 'db1' y nos retorne todas sus filas.

Programa

ejercicio308_2.py > ...

```
1   import psycopg2
2
3   conexion1=psycopg2.connect(database="db1", user="postgres", password="123456")
4   cursor1=conexion1.cursor()
5   cursor1.execute("select codigo, descripcion, precio from articulos")
6   for fila in cursor1:
7       print(fila)
8   conexion1.close()
```

Este será el resultado.

```
(1, 'naranjas', 23.5)
(2, 'peras', 34.0)
(3, 'bananas', 25.0)
```

Borrado y modificación de filas

Las otras dos actividades fundamentales que podemos hacer con una tabla es borrar filas y modificar datos.

Desarrollaremos un pequeño programa que borre el artículo cuyo código seas el 1 y modifique el precio del artículo cuyo código sea 3.

Vamos a programar:

ejercicio308_3.py > ...

```
1   import psycopg2
2
3   conexion1=psycopg2.connect(database="db1", user="postgres", password="123456")
4   cursor1=conexion1.cursor()
5   cursor1.execute("delete from articulos where codigo=1")
6   cursor1.execute("update articulos set precio=50 where codigo=3")
7   conexion1.commit()
8   cursor1.execute("select codigo, descripcion, precio from articulos")
9   for fila in cursor1:
10      print(fila)
11  conexion1.close()
```

Este será el resultado:

```
(2, 'peras', 34.0)
(3, 'bananas', 50.0)
```

Capítulo 309.- PostgeSQL: Interfaz visual con tkinter y acceso a la base de datos – 1

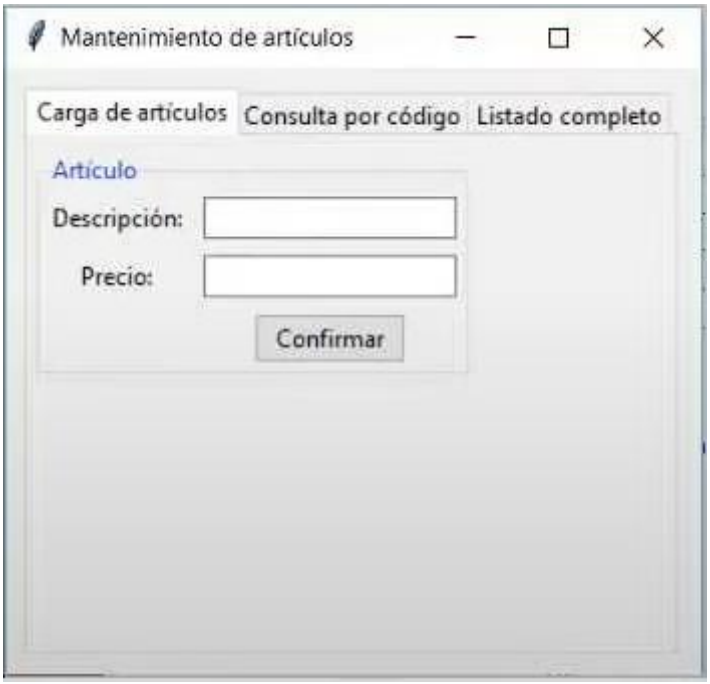
Es muy posible que necesitemos en algunas situaciones acceder a una base de datos de PostgreSQL desde aplicación con una interface visual.

Problema

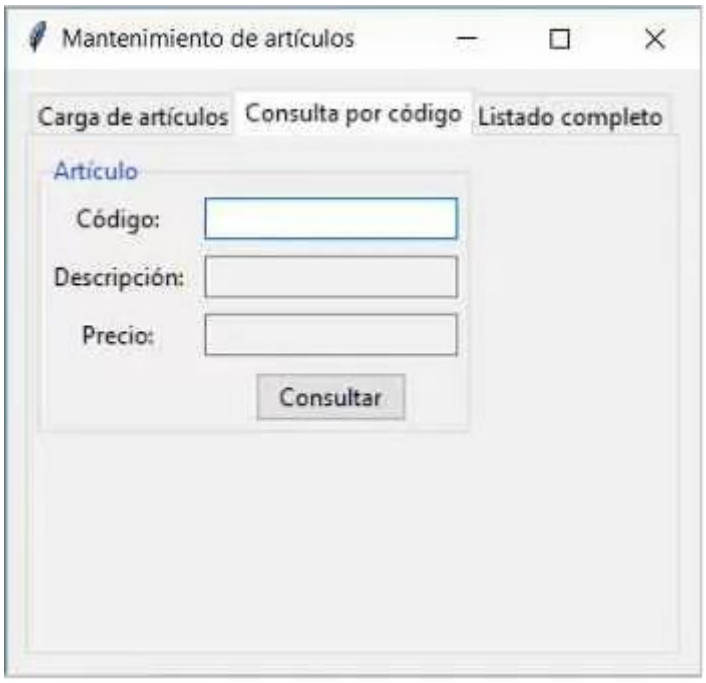
Desarrollar una aplicación visual con la librería tkinter que permita entrar los algoritmos de carga de artículos, consultas por código y listado completo.

Seguimos trabajando con la tabla 'articulos' que creamos en el concepto anterior.

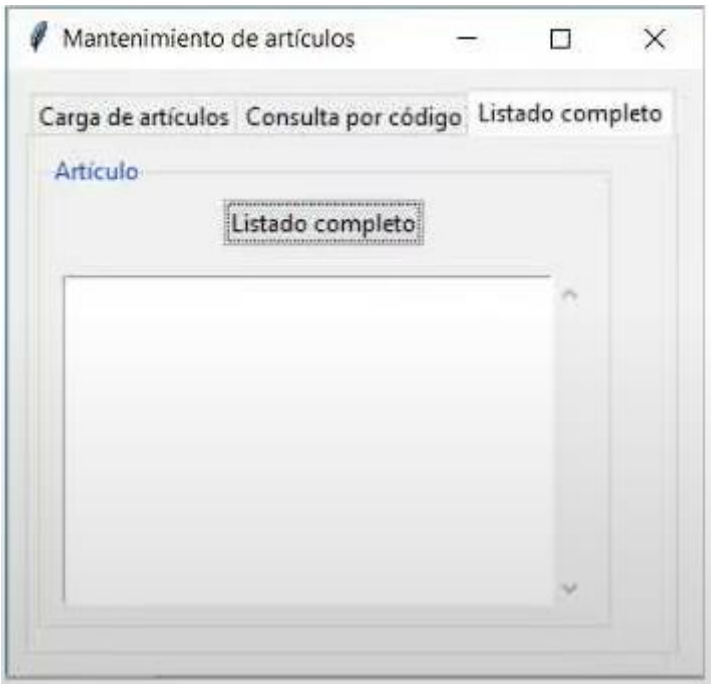
La interfaz visual para la carga debe ser:



La interfaz visual para la consulta:



Y finalmente la interfaz para el listado completo:



Para trabajar un poco más ordenado en la resolución de este problema lo dividiremos en dos módulos 'formularioarticulos.py' y 'articulos.py'

El primer paso será crear una carpeta llamada 'proyecto5' y dentro de esta creamos los dos módulos:

El código de "formularioarticulos.py".

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mb
from tkinter import scrolledtext as st
import articulos

class FormularioArticulos:
    def __init__(self):
        self.articulo1=articulos.Articulos()
        self.ventana1=tk.Tk()
        self.ventana1.title("Mantenimiento de artículos")
        self.cuaderno1 = ttk.Notebook(self.ventana1)
        self.carga_articulos()
        self.consulta_por_codigo()
        self.listado_completo()
        self.cuaderno1.grid(column=0, row=0, padx=10, pady=10)
        self.ventana1.mainloop()

    def carga_articulos(self):
        self.pagina1 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina1, text="Carga de artículos")
```

```
        self.labelframe1=ttk.LabelFrame(self.pagina1,
text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe1, text="Descripción:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.descripcioncarga=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe1,
textvariable=self.descripcioncarga)
        self.entrydescripcion.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1, text="Precio:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.preciocarga=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe1,
textvariable=self.preciocarga)
        self.entryprecio.grid(column=1, row=1, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Confirmar",
command=self.agregar)
        self.boton1.grid(column=1, row=2, padx=4, pady=4)

    def agregar(self):
        datos=(self.descripcioncarga.get(), self.preciocarga.get())
        self.articulo1.alta(datos)
        mb.showinfo("Información", "Los datos fueron cargados")
        self.descripcioncarga.set("")
        self.preciocarga.set("")

    def consulta_por_codigo(self):
        self.pagina2 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina2, text="Consulta por código")
        self.labelframe1=ttk.LabelFrame(self.pagina2, text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe1, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigo=tk.StringVar()
        self.entrycodigo=ttk.Entry(self.labelframe1,
textvariable=self.codigo)
        self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1,
text="Descripción:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.descripcion=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe1,
textvariable=self.descripcion, state="readonly")
        self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)
        self.label3=ttk.Label(self.labelframe1, text="Precio:")
        self.label3.grid(column=0, row=2, padx=4, pady=4)
        self.precio=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe1,
textvariable=self.precio, state="readonly")
```

```
        self.entryprecio.grid(column=1, row=2, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Consultar",
command=self.consultar)
        self.boton1.grid(column=1, row=3, padx=4, pady=4)

    def consultar(self):
        datos=(self.codigo.get(), )
        respuesta=self.articulo1.consulta(datos)
        if len(respuesta)>0:
            self.descripcion.set(respuesta[0][0])
            self.precio.set(respuesta[0][1])
        else:
            self.descripcion.set('')
            self.precio.set('')
            mb.showinfo("Información", "No existe un artículo con dicho
código")

    def listado_completo(self):
        self.pagina3 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina3, text="Listado completo")
        self.labelframe1=ttk.LabelFrame(self.pagina3, text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.boton1=ttk.Button(self.labelframe1, text="Listado completo",
command=self.listar)
        self.boton1.grid(column=0, row=0, padx=4, pady=4)
        self.scrolledtext1=st.ScrolledText(self.labelframe1, width=30,
height=10)
        self.scrolledtext1.grid(column=0,row=1, padx=10, pady=10)

    def listar(self):
        respuesta=self.articulo1.recuperar_todos()
        self.scrolledtext1.delete("1.0", tk.END)
        for fila in respuesta:
            self.scrolledtext1.insert(tk.END,
"código:"+str(fila[0])+"\ndescripción:"+fila[1]+"\nprecio:"+str(fila[2])+
"\n\n")


aplicacion1=FormularioArticulos()
```

Código 'articulos.py'

```
import psycopg2

class Articulos:

    def abrir(self):
        conexion = psycopg2.connect(database="db1", user="postgres",
password="123456")
```

```
        return conexion

    def alta(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="insert into articulos(descripcion, precio) values (%s,%s)"
        cursor.execute(sql, datos)
        cone.commit()

    def consulta(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="select descripcion, precio from articulos where codigo=%s"
        cursor.execute(sql, datos)
        return cursor.fetchall()

    def recuperar_todos(self):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="select codigo, descripcion, precio from articulos"
        cursor.execute(sql)
        return cursor.fetchall()
```

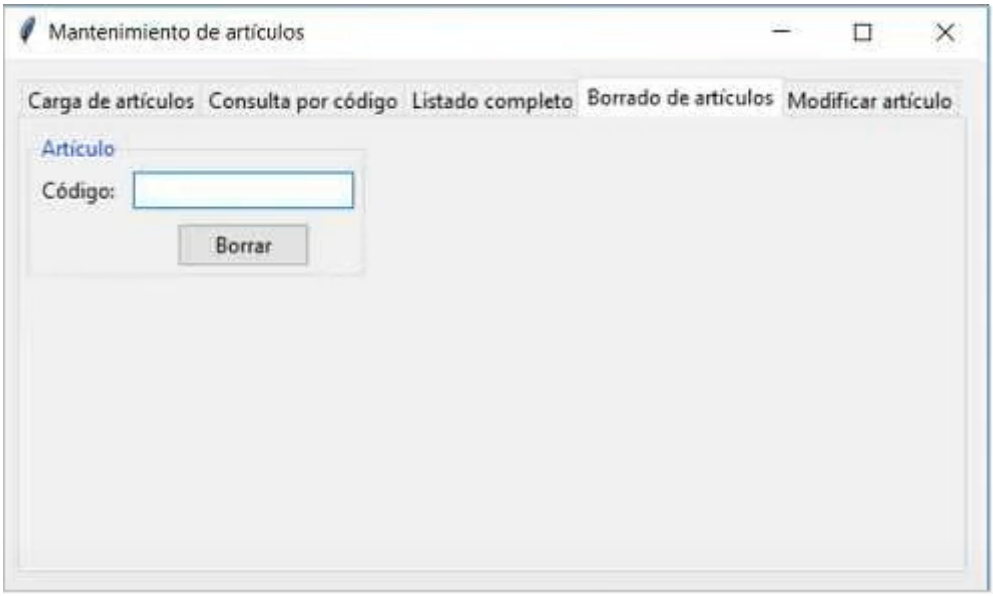
Ahora puedes ejecutar y probar el programa, tienes que ejecutar el que tiene el nombre de formularioarticulos.py.

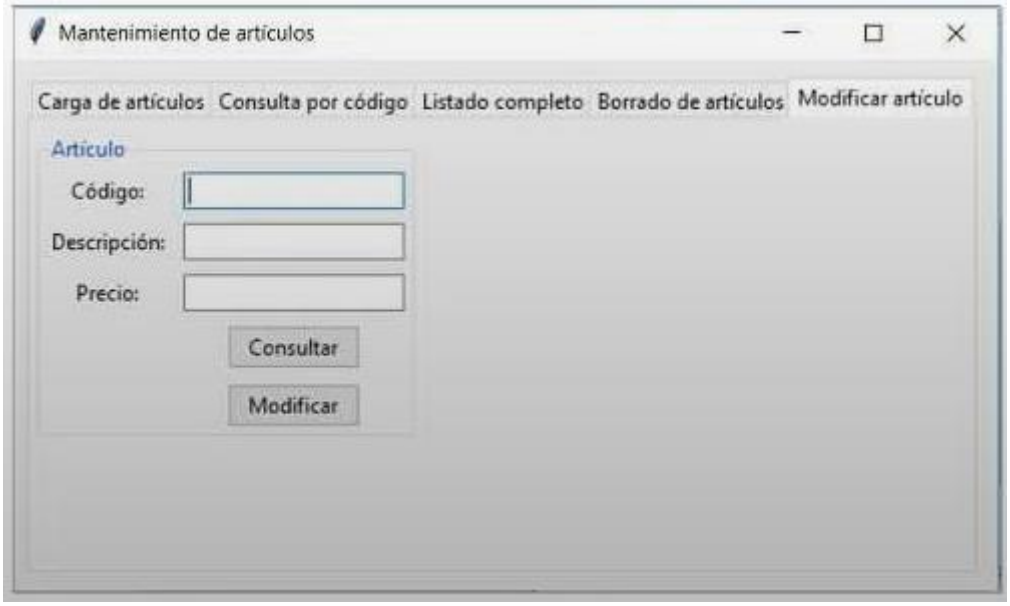
Capítulo 310.- PostgreSQL: Interfaz visual con tkinter y acceso a la base de datos – 2

Problema propuesto

Agregar dos pestañas al programa de administración de artículos que permitan borrar un artículo ingresando su código, y otra opción que permita consultar y modificar la descripción y precio de un artículo.

Las interfaces visuales a implementar son:





Vamos a modificar el programa anterior y ahora debe tener el siguiente código:

Este será el código de formularioarticulos.py.

```
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox as mb
```

```
from tkinter import scrolledtext as st
import articulos

class FormularioArticulos:
    def __init__(self):
        self.articulo1=articulos.Articulos()
        self.ventana1=tk.Tk()
        self.ventana1.title("Mantenimiento de artículos")
        self.cuaderno1 = ttk.Notebook(self.ventana1)
        self.carga_articulos()
        self.consulta_por_codigo()
        self.listado_completo()
        self.borrado()
        self.modificar()
        self.cuaderno1.grid(column=0, row=0, padx=10, pady=10)
        self.ventana1.mainloop()

    def carga_articulos(self):
        self.pagina1 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina1, text="Carga de artículos")
        self.labelframe1=ttk.LabelFrame(self.pagina1,
text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe1, text="Descripción:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.descripcioncarga=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe1,
textvariable=self.descripcioncarga)
        self.entrydescripcion.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1, text="Precio:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.preciocarga=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe1,
textvariable=self.preciocarga)
        self.entryprecio.grid(column=1, row=1, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Confirmar",
command=self.agregar)
        self.boton1.grid(column=1, row=2, padx=4, pady=4)

    def agregar(self):
        datos=(self.descripcioncarga.get(), self.preciocarga.get())
        self.articulo1.alta(datos)
        mb.showinfo("Información", "Los datos fueron cargados")
        self.descripcioncarga.set("")
        self.preciocarga.set("")

    def consulta_por_codigo(self):
        self.pagina2 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina2, text="Consulta por código")
```

```
        self.labelframe1=ttk.LabelFrame(self.pagina2, text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe1, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigo=tk.StringVar()
        self.entrycodigo=ttk.Entry(self.labelframe1,
textvariable=self.codigo)
        self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1,
text="Descripción:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.descripcion=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe1,
textvariable=self.descripcion, state="readonly")
        self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)
        self.label3=ttk.Label(self.labelframe1, text="Precio:")
        self.label3.grid(column=0, row=2, padx=4, pady=4)
        self.precio=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe1,
textvariable=self.precio, state="readonly")
        self.entryprecio.grid(column=1, row=2, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Consultar",
command=self.consultar)
        self.boton1.grid(column=1, row=3, padx=4, pady=4)

    def consultar(self):
        datos=(self.codigo.get(), )
        respuesta=self.articulo1.consulta(datos)
        if len(respuesta)>0:
            self.descripcion.set(respuesta[0][0])
            self.precio.set(respuesta[0][1])
        else:
            self.descripcion.set('')
            self.precio.set('')
            mb.showinfo("Información", "No existe un artículo con dicho
código")

    def listado_completo(self):
        self.pagina3 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina3, text="Listado completo")
        self.labelframe1=ttk.LabelFrame(self.pagina3, text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.boton1=ttk.Button(self.labelframe1, text="Listado completo",
command=self.listar)
        self.boton1.grid(column=0, row=0, padx=4, pady=4)
        self.scrolledtext1=st.ScrolledText(self.labelframe1, width=30,
height=10)
        self.scrolledtext1.grid(column=0,row=1, padx=10, pady=10)
```

```
    def listar(self):
        respuesta=self.articulo1.recuperar_todos()
        self.scrolledtext1.delete("1.0", tk.END)
        for fila in respuesta:
            self.scrolledtext1.insert(tk.END, "código:"+str(fila[0])+
                                              "\ndescripción:"+fila[1]+
                                              "\nprecio:"+str(fila[2])+"\
n\n")

    def borrado(self):
        self.pagina4 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina4, text="Borrado de artículos")
        self.labelframe1=ttk.LabelFrame(self.pagina4,
text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe1, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigoborra=tk.StringVar()
        self.entryborra=ttk.Entry(self.labelframe1,
textvariable=self.codigoborra)
        self.entryborra.grid(column=1, row=0, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Borrar",
command=self.borrar)
        self.boton1.grid(column=1, row=1, padx=4, pady=4)

    def borrar(self):
        datos=(self.codigoborra.get(), )
        cantidad=self.articulo1.baja(datos)
        if cantidad==1:
            mb.showinfo("Información", "Se borró el artículo con dicho
código")
        else:
            mb.showinfo("Información", "No existe un artículo con dicho
código")

    def modificar(self):
        self.pagina5 = ttk.Frame(self.cuaderno1)
        self.cuaderno1.add(self.pagina5, text="Modificar artículo")
        self.labelframe1=ttk.LabelFrame(self.pagina5, text="Artículo")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.label1=ttk.Label(self.labelframe1, text="Código:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.codigomod=tk.StringVar()
        self.entrycodigo=ttk.Entry(self.labelframe1,
textvariable=self.codigomod)
        self.entrycodigo.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1,
text="Descripción:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
```

```
        self.descripcionmod=tk.StringVar()
        self.entrydescripcion=ttk.Entry(self.labelframe1,
textvariable=self.descripcionmod)
        self.entrydescripcion.grid(column=1, row=1, padx=4, pady=4)
        self.label3=ttk.Label(self.labelframe1, text="Precio:")
        self.label3.grid(column=0, row=2, padx=4, pady=4)
        self.preciomod=tk.StringVar()
        self.entryprecio=ttk.Entry(self.labelframe1,
textvariable=self.preciomod)
        self.entryprecio.grid(column=1, row=2, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Consultar",
command=self.consultar_mod)
        self.boton1.grid(column=1, row=3, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1, text="Modificar",
command=self.modifica)
        self.boton1.grid(column=1, row=4, padx=4, pady=4)

    def modifica(self):
        datos=(self.descripcionmod.get(), self.preciomod.get(),
self.codigomod.get())
        cantidad=self.articulo1.modificacion(datos)
        if cantidad==1:
            mb.showinfo("Información", "Se modificó el artículo")
        else:
            mb.showinfo("Información", "No existe un artículo con dicho
código")

    def consultar_mod(self):
        datos=(self.codigomod.get(), )
        respuesta=self.articulo1.consulta(datos)
        if len(respuesta)>0:
            self.descripcionmod.set(respuesta[0][0])
            self.preciomod.set(respuesta[0][1])
        else:
            self.descripcionmod.set('')
            self.preciomod.set('')
            mb.showinfo("Información", "No existe un artículo con dicho
código")
```

```
aplicacion1=FormularioArticulos()
```

Este ahora será el código de artículos.py.

```
import psycopg2

class Articulos:

    def abrir(self):
        conexion = psycopg2.connect(database="db1", user="postgres",
password="123456")
        return conexion

    def alta(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="insert into articulos(descripcion, precio) values (%s,%s)"
        cursor.execute(sql, datos)
        cone.commit()

    def consulta(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="select descripcion, precio from articulos where codigo=%s"
        cursor.execute(sql, datos)
        return cursor.fetchall()

    def recuperar_todos(self):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="select codigo, descripcion, precio from articulos"
        cursor.execute(sql)
        return cursor.fetchall()

    def baja(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="delete from articulos where codigo=%s"
        cursor.execute(sql, datos)
        cone.commit()
        return cursor.rowcount # retornamos la cantidad de filas borradas

    def modificacion(self, datos):
        cone=self.abrir()
        cursor=cone.cursor()
        sql="update articulos set descripcion=%s, precio=%s where
codigo=%s"
        cursor.execute(sql, datos)
        cone.commit()
```

```
        return cursor.rowcount # retornamos la cantidad de filas
modificadas
```

Ahora desde el proyecto formularioarticulos.py lo ejecutas y pruebas todas las opciones.

Capítulo 311.- Módulos y paquetes de Python

Vimos en conceptos anteriores que podemos dividir una aplicación en varios módulos y luego importarlos.

Recordemos que un módulo es un archivo con extensión 'py' y contiene en su interior funciones, clases, definición de variables, etc.

Veremos ahora que un paquete en Python agrupa un conjunto de módulos relacionados y se los localiza en un directorio.

Veremos con un sencillo problema como creamos un paquete en Python.

Problema

Desarrolar una aplicación visual que muestre dos botones. Al presionar el primero mostrar una ventana para el login, y al presionar el segundo mostrar una ventana de mensajes.

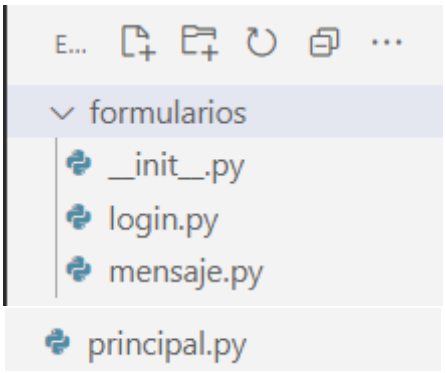
Crear un paquete llamado 'formularios' y en su interior dos módulos llamados 'login.py' y 'mensajes.py'.

La aplicación principal llamarla 'principal.py'.

Debemos crear una carpeta llamada 'formularios' y en su interior tres archivos: 'login.py', 'mensaje.py' y __init__.py.

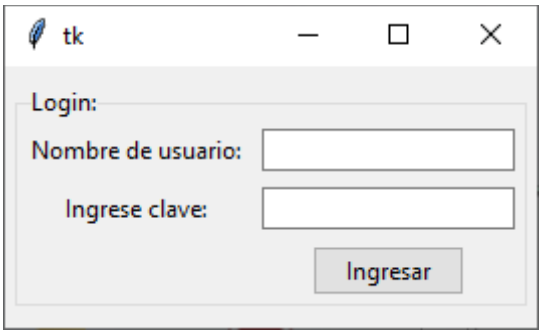
El archivo '__init__.py' generalmente se encuentra vacío y tiene por objetivo indicar al intérprete de Python que dicha carpeta es un paquete.

Es decir que debemos tener los siguientes archivos y carpeta en nuestra aplicación:



El archivo principal.py se encuentra fuera de la carpeta formularios.

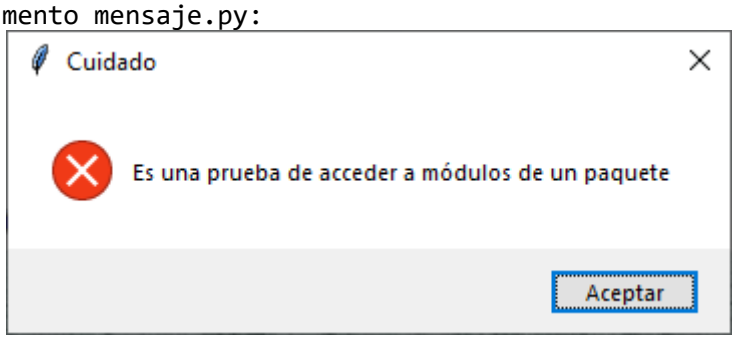
Interface de documento login.py:



Código del documento login.py:

```
import tkinter as tk
from tkinter import ttk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.labelframe1=ttk.LabelFrame(self.ventana1,
text="Login:")
        self.labelframe1.grid(column=0, row=0, padx=5, pady=10)
        self.login()

    def login(self):
        self.label1=ttk.Label(self.labelframe1, text="Nombre de
usuario:")
        self.label1.grid(column=0, row=0, padx=4, pady=4)
        self.entry1=ttk.Entry(self.labelframe1)
        self.entry1.grid(column=1, row=0, padx=4, pady=4)
        self.label2=ttk.Label(self.labelframe1, text="Ingrese
clave:")
        self.label2.grid(column=0, row=1, padx=4, pady=4)
        self.entry2=ttk.Entry(self.labelframe1, show="*")
        self.entry2.grid(column=1, row=1, padx=4, pady=4)
        self.boton1=ttk.Button(self.labelframe1,
text="Ingresar",command=self.ingresar)
        self.boton1.grid(column=1, row=2, padx=4, pady=4)

    def ingresar(self):
        self.ventana1.destroy()


def mostrar():
    aplicacion1=Aplicacion()
```

interfaz documento mensaje.py:

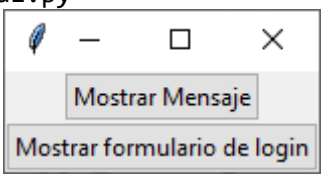


Código del documento mensaje.py:

```
from tkinter import messagebox as mb
```

```
def mostrar(mensaje):
    mb.showerror("Cuidado",mensaje)
```

Interfaz documento principal.py



Código del documento principal.py

```
import tkinter as tk
from tkinter import ttk
# Para tener acceso a la clase Aplicacion del
# documento login.py que se encuentra en al carpeta
# formularios.
import formularios.login
# Para tener acceso al método del documento
# mensaje.py que se encuentra en la capeta
# formularios.
import formularios.mensaje

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.ventana1.title("Hola Mundo")
        self.boton1=ttk.Button(self.ventana1, text="Mostrar Mensaje",
command=self.mostrar_mensaje)
        self.boton1.grid(column=0, row=0)
        self.boton2=ttk.Button(self.ventana1, text="Mostrar formulario de
login", command=self.mostrar_login)
        self.boton2.grid(column=0, row=1)
        self.ventana1.mainloop()

    def mostrar_mensaje(self):
        # Accediento al método mostrar del documento mensaje.py
        formularios.mensaje.mostrar("Es una prueba de acceder a módulos
de un paquete")

    def mostrar_login(self):
        # Accediendo al método login del documento login.py.
        formularios.login.mostrar()

aplicacion1=Aplicacion()
```

Capítulo 312.- Acceso a Internet mediante el paquete urllib – 1

Para acceder a recursos de Internet Python nos proporciona en la Biblioteca estándar un paquete fundamental llamado urllib.

Si bien lo más común es que accedamos a las páginas web a través de un navegador web (Chrome, Firefox, Edge, etc) podemos también hacerlo mediante un programa codificado en Python.

El primer ejemplo nos mostrará la facilidad que tenemos en Python para recuperar un archivo html localizado en Internet.

Lectura de una página HTML.

Recupera la página html 'pagina.html' que se encuentra localizada en:

http://www.peremanelv.com/index.html

Programar:

ejercicio312.py > ...

```
1   from urllib import request
2
3   pagina=request.urlopen("http://www.peremanelv.com/index.html")
4   datos=pagina.read()
5   print(datos)
```

Este será el resultado cuando ejecutemos:

```
b'<!DOCTYPE html>\r\n<html lang="es">\r\n    <head>\r\n        <title>Rob\xc3\xb3tica y programaci\xc3\xb3n</title>\r\n
  <meta charset="utf-8">\r\n        <meta name="description" content="Rob\xc3\xb3tica y programaci\xc3\xb3n">\r\n        <met
a name="keywords" content="Bricklink Setudio, Lego WeDo, Lego Spike Essential, Lego Spike Prime, Lego Mindstorms, mBlock, Mak
e Code Arcde, Python, Html, SCC, JavaScript">\r\n        <link rel="stylesheet" href="misestilos.css">\r\n        <script src
="dni.js"></script> \r\n        \r\n    </head>\r\n    <body onload="horas()">\r\n        <header id="titulo">\r\n
<h1>Rob\xc3\xb3tica y programaci\xc3\xb3n de Pere Manel</h1>\r\n        </header>\r\n        <nav>\r\n            <a id="rojo
" href="index.html">Principal</a> |\r\n            <a href="bricklink.html">Bricklink Studio</a> |\r\n            <a href="we
do.html">Lego WeDo - BOOST</a> |\r\n            <a href="essential.html">Lego Spike Essential</a> |\r\n            <a href="p
rime.html">Lego Spique Prime</a> |\r\n            <a href="mindstorms.html">Lego Mindstorms</a> |\r\n            <a href="mbl
```

Lo primero que hacemos es importar el módulo 'request' que se encuentra en el paquete 'utllib':

```
from urllib import request
```

Mediante la función 'urlopen' recuperamos del servidor el recurso que le indicamos como parámetro.

```
pagina=request.urlopen("http://www.peremanelv.com/index.html")
```

La función 'urlopen' retorna la referencia de un objeto de la clase 'HTTPResponse' y se almacena en la variable 'pagina'.

Procedemos seguidamente a llamar al método 'read' que tiene por objetivo recuperar todo el contenido de la página HTML.

```
datos=pagina.read()
print(datos)
```

Transformar a formato 'utf-8'

El método 'read' recupera los datos como un bloque de bytes, si vemos la imagen anterior donde imprimimos la variable 'datos' podemos comprobar que aparece en comillas y antecedido por el carácter 'b':

b'<!DOCTYPE html>\r\n<html lang="es">\r\n <head>\r\n <title>Rob\xc3\xb3tica y programaci\xc3\xb3n</title>\r\n

<meta charset="utf-8">\r\n <meta name="description" content="Rob\xc3\xb3tica y programaci\xc3\xb3n">\r\n <meta name="keywords" content="Bricklink Setudio, Lego WeDo, Lego Spike. . . . . .

Luego si queremos convertir el bloque de bytes a 'utf-8' debemos hacerlo llamando al método 'decode' indicando el formato respectivo:

ejercicio312.py > ...

```
1   from urllib import request
2
3   pagina=request.urlopen("http://www.peremanelv.com/index.html")
4   datos=pagina.read()
5   datostf8=datos.decode("utf-8")
6   print(datostf8)
```

Este será el resultado:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Robótica y programación</title>
        <meta charset="utf-8">
        <meta name="description" content="Robótica y programación">
        <meta name="keywords" content="Bricklink Setudio, Lego WeDo, Lego Spike Essential, Lego Spike Prime, Lego Mindstorms,
 mBlock, Make Code Arcde, Python, Html, SCC, JavaScript">
        <link rel="stylesheet" href="misestilos.css">
        <script src="dni.js"></script>
```

Si ejecutamos ahora nuevamente la aplicación podemos ver el contenido convertido a texto con formato 'utf-8' (formato de codificación de caracteres más común empleado en Internet para representar múltiples idiomas.

Lectura de una página HTML u otro recurso y posterior grabación del archivo en forma local.

Recuperar la página html 'index.html' y el archivo '"imagenes/Pere.png" que se encuentran localizados en:

http://www.peremanelv.com/index.html
http://www.peremanelv.com/imagenes/Pere.png

Luego grabar los dos archivos en forma local en el equipo donde se está ejecutando el script de Python.

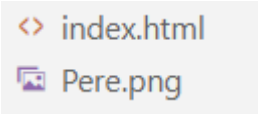
Programa:

ejercicio312.py > ...

```
1   from urllib import request
2
3   pagina=request.urlopen("http://www.peremanelv.com/index.html")
4   datos=pagina.read()
5   archivo1=open("index.html", "wb")
```

```
6   archivo1.write(datos)
7   archivo1.close()
8
9   imagen=request.urlopen("http://www.peremanelv.com/imagenes/Pere.png")
10  datos=imagen.read()
11  archivo2=open("Pere.png", "wb")
12  archivo2.write(datos)
13  archivo2.close()
```

Lo ejecutamos, en la consola no veremos nada pero si podemos observar que se han guardado dos ficheros.



Procedemos primero a importar el módulo 'request' del paquete ¡ulrlib'.

```
from urllib import request
```

Recuperamos del servidor el recurso 'index.html' en formato binario:

```
pagina=request.urlopen("http://www.peremanelv.com/index.html")
datos=pagina.read()
```

Creamos un archivo binario en nuestro equipo con el nombre 'index.html' y procedemos a almacenar el bloque de bytes recuperados del servidor:

```
archivo1=open("index.html", "wb")
archivo1.write(datos)
```

Cerramos el archivo que acabamos de crear llamando al método 'close'.

```
archivo1.close()
```

Los mismos pasos damos para recuperar el archivo 'Pere.png' y crea un archivo en nuestro equipo.

```
imagen=request.urlopen("http://www.peremanelv.com/imagenes/Pere.png")
datos=imagen.read()
archivo2=open("Pere.png", "wb")
archivo2.write(datos)
archivo2.close()
```

Es decir no es importante el formato del archivo (html/jpg/gif/json etc.) para recuperarlo de un servidor de Internet.

Módulo 'error' del paquete 'urllib'.

Si el recurso no se encuentra en el servidor de Internet o se genera cualquier otro tipo de error, podemos capturar la excepción 'HTTPError' del paquete 'urllib'.

Confeccionamos un script que intente recuperar una página HTML que no se encuentra en el servidor:

http://www.scratchya.com.ar/pythonya/ejercicio336/paginax.html

Luego capturamos la excepción 'HTTPError'

Programa

ejercicio312_1.py > ...

```
1  from urllib import request
2  from urllib import error
3  try:
4      pagina=request.urlopen("http://www.scratchya.com.ar/pythonya/ejercicio336/paginax.html")
5      datos=pagina.read().decode("utf-8")
6      print(datos)
7  except error.HTTPError as err:
8      print(f"Código de respuesta HTTP devuelto del servidor {err.code}")
9      print(f"No exite el recurso {err.filename}")
```

Este será el resultado:

```
Código de respuesta HTTP devuelto del servidor 404
No exite el recurso http://www.scratchya.com.ar/pythonya/ejercicio336/paginax.html
```

Lo primero que hacemos además de importar el módulo 'request', es importar el módulo 'error' también almacenado en el paquete 'urllib':

```
from urllib import request
from urllib import error
```

Al método 'urlopen' lo debemos disponer dentro de un bloque try:

```
try:
    pagina=request.urlopen("http://www.scratchya.com.ar/pythonya/ejercici
o336/paginax.html")
    datos=pagina.read().decode("utf-8")
    print(datos)
```

El método 'urlopen' puede generar una excepción de tipo 'HTTPError':

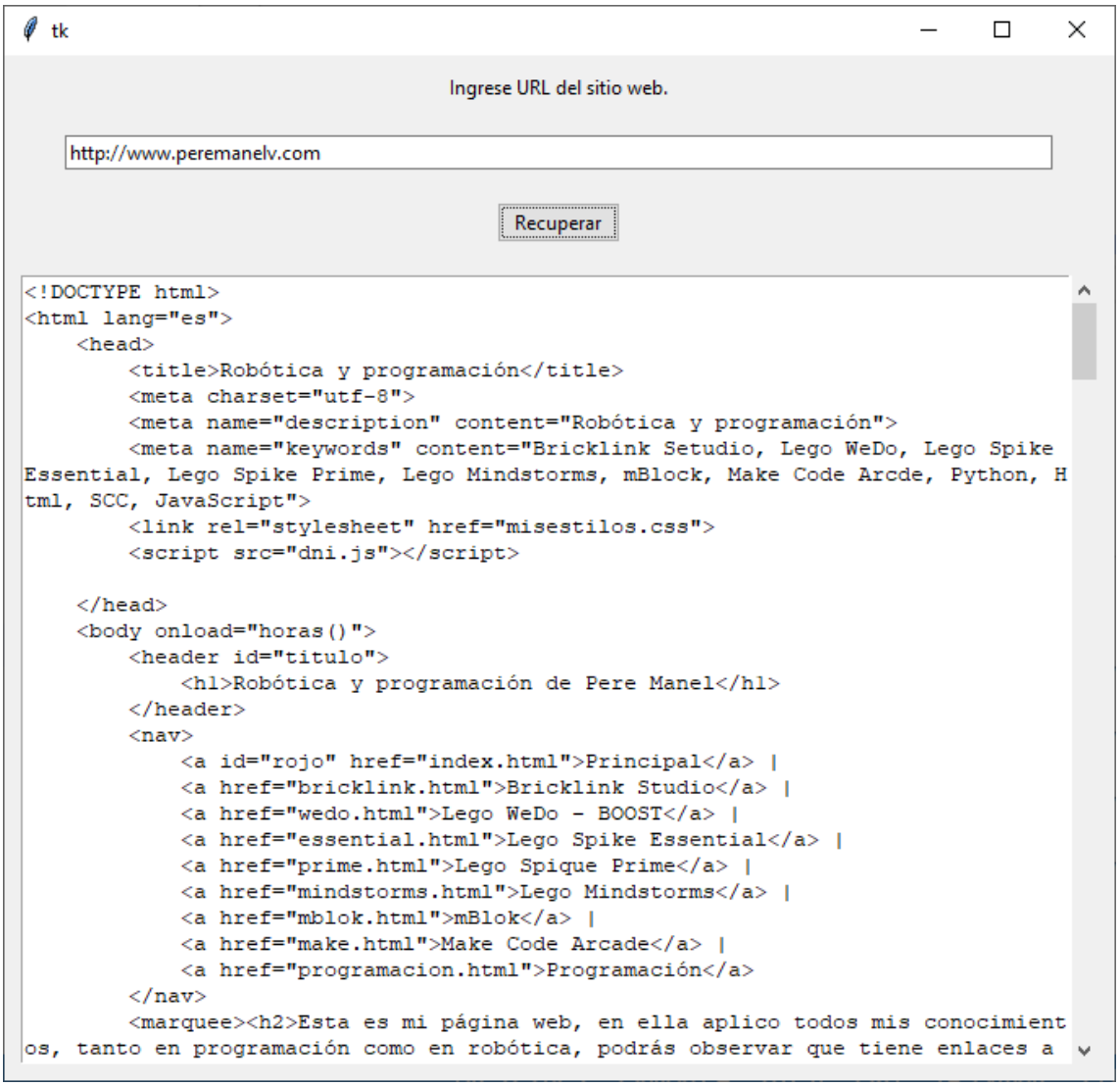
```
except error.HTTPError as err:
    print(f"Código de respuesta HTTP devuelto del servidor {err.code}")
    print(f"No exite el recurso {err.filename}")
```

Como el archivo 'paginax.html' no existe luego se ejecuta el bloque except donde podemos consultar los datos almacenados en la excepción.

Capítulo 313.- Acceso a Internet mediante el paquete utllib – 2

Problema propuesto

Confeccionar una aplicación visual con tkinter que permita ingresar en un control de tipo 'Entry' la URL de un sitio web y al presionar un botón recuperar los datos y mostrarlos en un control de tipo 'ScrolledText':

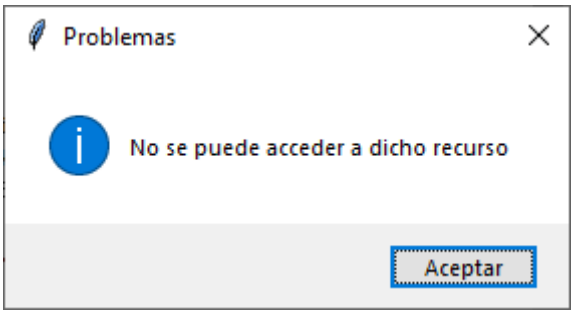


Vamos a programar:

```
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext as st
from urllib import request
from urllib import error
from tkinter import messagebox as mb

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
```

```
        self.label1=ttk.Label(self.ventana1, text="Ingrese URL del sitio
web.")
        self.label1.grid(column=0, row=0, padx=10, pady=10)
        self.varurl=tk.StringVar()
        self.entry1=ttk.Entry(self.ventana1, width=100,
textvariable=self.varurl)
        self.entry1.grid(column=0, row=1, padx=10, pady=10)
        self.boton1=ttk.Button(self.ventana1, text="Recuperar",
command=self.recuperar)
        self.boton1.grid(column=0, row=2, padx=10, pady=10)
        self.scrolledtext1=st.ScrolledText(self.ventana1, width=80,
height=30)
        self.scrolledtext1.grid(column=0, row=3, padx=10, pady=10)
        self.ventana1.mainloop()

    def recuperar(self):
        try:
            pagina=request.urlopen(self.varurl.get())
            datos=pagina.read().decode("utf-8")
            self.scrolledtext1.delete("1.0", tk.END)
            self.scrolledtext1.insert(tk.INSERT, datos)
        except error.HTTPError as err:
            mb.showinfo("Problemas", "No se puede acceder a dicho
recurso")



# Bloque principal

aplicacion1=Aplicacion()
```

Ya puedes ejecutar, si introduces una url que no existe obtendrás el siguiente mensaje:



Capítulo 314.- Procesar archivos con formato JSON en Python módulo json – 1

Vimos en el concepto anterior como podemos acceder a recursos de Internet desde Python. Un recurso muy común para transferir datos en Internet son los archivos con formato JSON.

La Biblioteca estándar de Python cuenta con un módulo que nos facilita trabajar con archivos JSON.

El formato JSON se lo puede transformar en Python a un diccionario. Por ejemplo:

ejercicio314.py > ...

```
1   import json
2   cadena="""
3       {
4           "codigo":"1",
5           "descripción":"patatas",
6           "precio":"13.45"
7       }
8   """
9   print(cadena)  # Imprimimos un string
10  diccionario=json.loads(cadena)
11  print(diccionario)  # Imprimimos un diccionario
```

Este será el resultado:

```
{
    "codigo":"1",
    "descripción":"patatas",
    "precio":"13.45"
}

{'codigo': '1', 'descripción': 'patatas', 'precio': '13.45'}
```

De forma similar si disponemos de un vector, luego la transformación será a una lista de elementos de tipo diccionario:

ejercicio314.py > ...

```
1   import json
2   cadena="""
3   [
4       {
5           "codigo":"1",
6           "descripción":"patatas",
7           "precio":"13.45"
8       },
```

```
9        {
10           "codigo":"2",
11           "descripción":"manzanas",
12           "precio":"45"
13       }
14   ]
15   """
16   print(cadena)  # Imprimimos un string
17   lista=json.loads(cadena)
18   print(lista)  # Imprimimos una lista
```

Este será el resultado:

```
[
    {
        "codigo":"1",
        "descripción":"patatas",
        "precio":"13.45"
    },
     {
        "codigo":"2",
        "descripción":"manzanas",
        "precio":"45"
    }
]
```

```
[{'codigo': '1', 'descripción': 'patatas', 'precio': '13.45'}, {'codigo': '2'
, 'descripción': 'manzanas', 'precio': '45'}]
```

Problema 1

Implementar un sitio WEB en PHP que retorne un archivo en formato JSON con los datos de diferentes artículos.

Para resolver el problema debemos tener un conocimiento del lenguaje PHP y del gestor de base de datos MySQL.

Cree una base de datos llamada 'pythonya' y una tabla 'articulos' con los siguientes datos:

```
CREATE TABLE articulos (
  codigo int(11) NOT NULL AUTO_INCREMENT,
  descripcion varchar(50) DEFAULT NULL,
  precio float DEFAULT NULL,
  PRIMARY KEY(codigo)
);

insert into articulos values (1, 'patatas', 15);
insert into articulos values (2, 'manzanas', 24);
insert into articulos values (3, 'peras', 45.3);
insert into articulos values (4, 'naranjas', 22);
insert into articulos values (5, 'pomelos', 29);
```

```
insert into articulos values (6, 'frutilla', 130);
insert into articulos values (7, 'anana', 75);
```

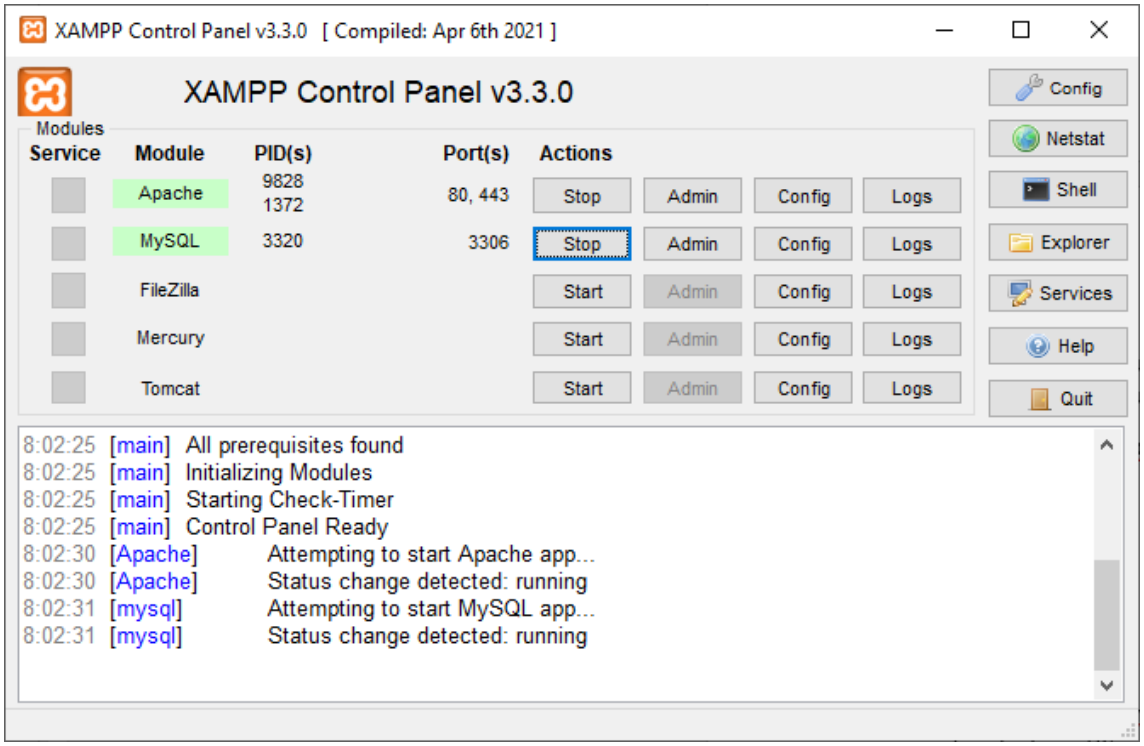
Programa: retornararticulos.php

```
<?php
header('Content – Type: application/json');

$server="localhost";
$usuario="root";
$clave="";
$base="pythonya";
$conexion=mysqli_connect($server, $usuario, $clave, $base) or die("problemas");
mysqli_set_charset($conexion, 'utf8');

$datos=mysqli_query($conexion, "SELECT codigo, descripcion, precio from articulos");
$resultado=mysqli_fetch_all($datos, MYSQLI_ASSOC);
echo json_encode($resultado);
?>
```

Si ya ha estudiado un poco de PHP sabe que al solicitar ésta página desde un navegador web lo que sucede es que retorna un archivo en formato JSON.

Vamos a realizar la práctica:

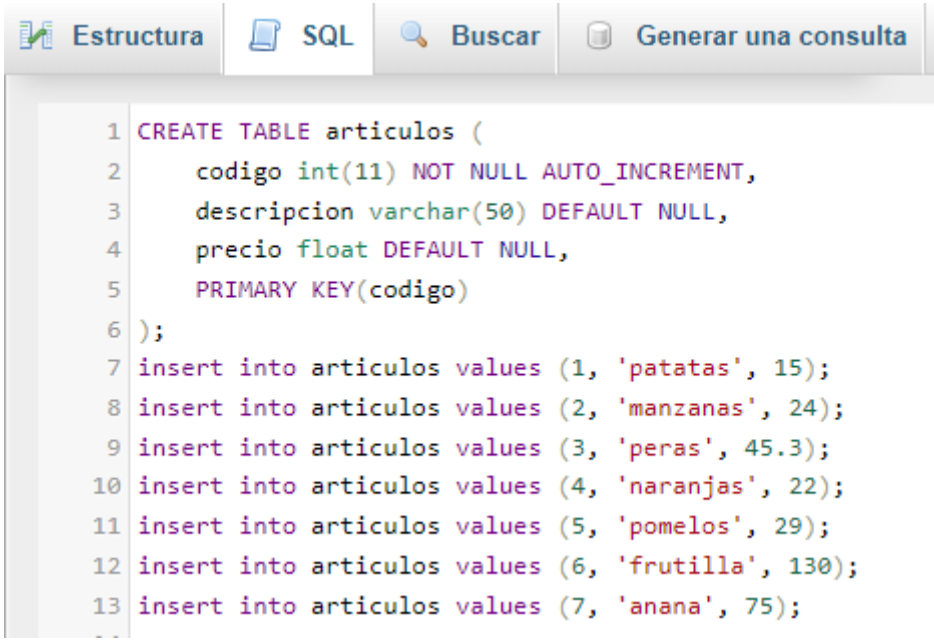
Vamos a ejecutar XAMPP Control Panel.



Vamos a entrar en phpMyAdmin para crear la base de datos.

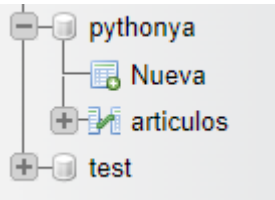
Creamos la base de datos:

Vamos a crear la tabla y agregar los registros:




```
CREATE TABLE articulos (
    codigo int(11) NOT NULL AUTO_INCREMENT,
    descripcion varchar(50) DEFAULT NULL,
    precio float DEFAULT NULL,
    PRIMARY KEY(codigo)
);
insert into articulos values (1, 'patatas', 15);
insert into articulos values (2, 'manzanas', 24);
insert into articulos values (3, 'peras', 45.3);
insert into articulos values (4, 'naranjas', 22);
insert into articulos values (5, 'pomelos', 29);
insert into articulos values (6, 'frutilla', 130);
insert into articulos values (7, 'anana', 75);
```

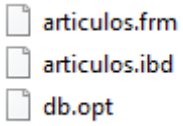
Y le damos al botón Continuar.



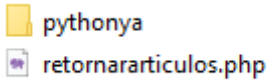
			codigo	descripcion	precio
Editar	Copiar	Borrar	1	patatas	15
Editar	Copiar	Borrar	2	manzanas	24
Editar	Copiar	Borrar	3	peras	45.3
Editar	Copiar	Borrar	4	naranjas	22
Editar	Copiar	Borrar	5	pomelos	29
Editar	Copiar	Borrar	6	frutilla	130
Editar	Copiar	Borrar	7	anana	75

Ahora accederemos a la carpeta:

C:\xampp\htdocs\pythonya



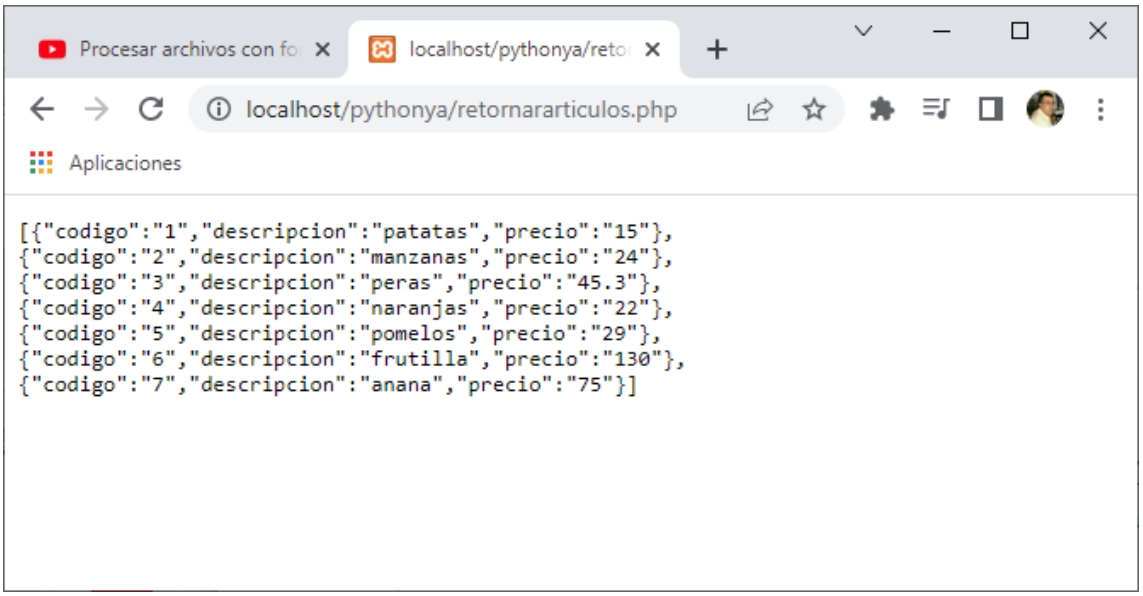
Creamos una carpeta con el nombre pythonya, y arrastramos estos archivos en ella.



Copiamos el archivo PHP en dicha carpeta:

Ahora desde el navegador:

Escribimos la siguiente url: http://localhost/pythonya/retornararticulos.php



Ya tenemos acceso a los registros de la tabla.

Ahora implementar una aplicación en Python que recupere y muestre los datos del archivo JSON generado en un servidor de Internet accediendo a una base de datos MySQL a través de PHP:

Programa:

```
from urllib import request
import json

pagina=request.urlopen("http://localhost/pythonya/retornararticulos.php")
datos=pagina.read().decode("utf-8")
print("_"*100)
lista=json.loads(datos) # Convertimos el string en una lista
print(lista) # Imprimimos la lista
print("_"*100)
```

```
for elemento in lista:
    print(f"{elemento['codigo']}  {elemento['descripcion']:50} {elemento['precio']:>12}")
```

Este será el resultado:

```
[{'codigo': '1', 'descripcion': 'patatas', 'precio': '15'}, {'codigo': '2', 'descripcion': 'manzanas
', 'precio': '24'}, {'codigo': '3', 'descripcion': 'peras', 'precio': '45.3'}, {'codigo': '4', 'desc
ripcion': 'naranjas', 'precio': '22'}, {'codigo': '5', 'descripcion': 'pomelos', 'precio': '29'}, {'
codigo': '6', 'descripcion': 'frutilla', 'precio': '130'}, {'codigo': '7', 'descripcion': 'anana', '
precio': '75'}]
```

```
1  patatas                                                  15
2  manzanas                                                 24
3  peras                                                  45.3
4  naranjas                                                 22
5  pomelos                                                  29
6  frutilla                                                130
7  anana                                                    75
```

Importamos los módulos request y json.

```
from urllib import request
import json
```

Solicitamos al servidor local (localhost) el recurso 'retornararticulos.php' localizado en la carpeta 'pythonya':

```
pagina=request.urlopen("http://localhost/pythonya/retornararticulos.php")
```

Leemos y decodificamos a formato 'utf-8':

```
datos=pagina.read().decode("utf-8")
```

La variable 'datos' es un string:

```
print(lista) # Imprimimos la lista
```

Mediante el método 'loads' del módulo 'json' transformamos el contenido del string en una lista con elementos de tipo lista:

```
lista=json.loads(datos) # Convertimos el string en una lista
```

Imprimimos la lista completa:

```
print(lista) # Imprimimos la lista
```

Hacemos una impresión de cada elemento de la lista con una formato más legible para el usuario:

```
for elemento in lista:
    print(f"{elemento['codigo']}  {elemento['descripcion']:50} {elemento['precio']:>12}")
```

Capítulo 315.- Procesar archivos con formato JSON en Python módulo json – 2

Problema 2

Ingresar por teclado el código de un artículo, seguidamente recuperar los datos de dicho artículo del servidor local. Pasa el código del artículo como parámetro 'GET' en la llamada.

La aplicación en PHP que recupera los datos de un determinado artículo que llega como parámetro es:

Programa: retornarunarticulo.php

```
<?php
header('Content – Type: application/json');

$server="localhost";
$usuario="root";
$clave="";
$base="pythonya";
$conexion=mysqli_connect($server, $usuario, $clave, $base) or die("problemas");
mysqli_set_charset($conexion, 'utf8');

$datos=mysqli_query($conexion, "SELECT codigo, descripcion, precio from articulos where codigo=$_GET[codigo]");
$resultado=mysqli_fetch_all($datos, MYSQLI_ASSOC);
echo json_encode($resultado);
?>
```

Recuerda que lo tienes que guardar en la siguiente dirección:

C:\xampp\htdocs\pythonya

Es importante notar que ahora recuperamos solo una fila de la tabla 'articulos', es decir aquel que coincide con el código de artículo pasado en la llamada:

$datos=mysqli_query($conexion, "SELECT codigo, descripcion, precio from artículos where codigo=$_GET[codigo]");

Ahora veamos el programa Python que llama a la página PHP anterior.

```
from urllib import request
import json
codigo=input("Ingrese el código del artículo a consultar:")
pagina=request.urlopen(f"http://localhost/pythonya/retornarunarticulo.php?codigo={codigo}")
datos=pagina.read().decode("utf-8")
lista=json.loads(datos)
if len(lista)>0:
    print(f"Descripción:{lista[0]['descripcion']}")
    print(f"Precio:{lista[0]['precio']}")
else:
    print("No existe un artículo con el código ingresado")
```

Cuando ejecutemos este será el resultado:

```
Ingrese el código del artículo a consultar:2
Descripción:manzanas
Precio:24
```

Si ejecutamos de nuevo e introducimos un código inexistente:

```
Ingrese el código del artículo a consultar:8
No existe un artículo con el código ingresado
```

El resultado de ejecutar la aplicación y cargar un código de artículo a consultar (recuerde tener activo el servidor web con PHP y MySQL):

Cuando nos conectamos al servidor le pasamos un parámetro llamado código y un número que ingresamos previamente por teclado.

```
codigo=input("Ingrese el código del artículo a consultar:")
pagina=request.urlopen(f"http://localhost/pythonya/retornarunarticulo.php
?codigo={codigo}")
```

Una vez que recuperemos los datos del servidor procedemos a convertir el string devuelto:

```
lista=json.loads(datos)
```

Si la lista no está vacía procedemos a mostrar la descripción y precio del artículo consultado:

```
if len(lista)>0:
    print(f"Descripción:{lista[0]['descripcion']}")
    print(f"Precio:{lista[0]['precio']}")
```

En caso contrario significa que intentamos consultar un código de artículo inexistente:

```
else:
    print("No existe un artículo con el código ingresado")
```

Capítulo 316.- Procesar archivos con formato JSON en Python módulo json – 3

Confeccionar una aplicación visual con tkinter que permita mostrar todos los artículos. Recuperar del servidor web llamando al recurso 'retornarartículos.php'.

Disponer dos botones web para poder navegar entre los distintos registros recuperados.

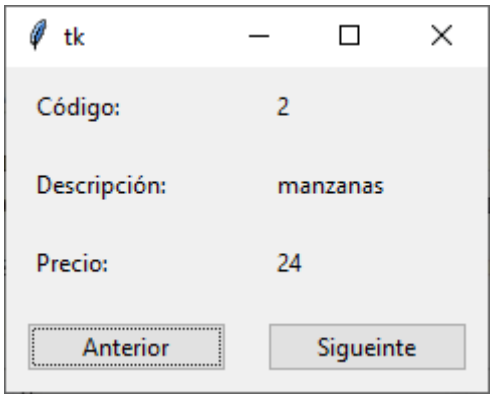
La interfaz visual debe ser similar a:

Vamos a programar, este lo he realizado sin mirar la solución:

```
import tkinter as tk
from tkinter import ttk
from urllib import request
import json

class Aplicacion:

    registro=0
    lista=[]

    def __init__(self):
        self.ventana1=tk.Tk()
        self.labelcodigo=ttk.Label(self.ventana1, text="Código:",
width=15, anchor="w")
        self.labelcodigo.grid(column=0, row=0, padx=10, pady=10)
        self.codigo=tk.StringVar()
        self.labelMosCodigo=ttk.Label(self.ventana1, text="",
textvariable=self.codigo, width=15, anchor="w")
        self.labelMosCodigo.grid(column=1, row=0, padx=10, pady=10)

        self.labeldescripcion=ttk.Label(self.ventana1,
text="Descripción:", width=15, anchor="w")
        self.labeldescripcion.grid(column=0, row=1, padx=10, pady=10)
        self.descripcion=tk.StringVar()
        self.labelMosdescripcion=ttk.Label(self.ventana1, text="",
textvariable=self.descripcion, width=15, anchor="w")
        self.labelMosdescripcion.grid(column=1, row=1, padx=10, pady=10)
```

```
        self.labelprecio=ttk.Label(self.ventana1, text="Precio:",
width=15, anchor="w")
        self.labelprecio.grid(column=0, row=3, padx=10, pady=10)
        self.precio=tk.StringVar()
        self.labelMosPrecio=ttk.Label(self.ventana1, text="",
textvariable=self.precio, width=15, anchor="w")
        self.labelMosPrecio.grid(column=1, row=3, padx=10, pady=10)

        botonAnt=ttk.Button(self.ventana1, text="Anterior", width=15,
command=self.anterior)
        botonAnt.grid(column=0, row=4, padx=10, pady=10)
        botonSig=ttk.Button(self.ventana1, text="Sigueinte", width=15,
command=self.siguiente)
        botonSig.grid(column=1, row=4, padx=10, pady=10)

        pagina=request.urlopen("http://localhost/pythonya/retornararticul
os.php")
        datos=pagina.read().decode("utf-8")
        self.lista=json.loads(datos)
        if len(self.lista)>0:
            self.codigo.set(str(self.lista[self.registro]['codigo']))
            self.descripcion.set(self.lista[self.registro]['descripcion']
)
            self.precio.set(str(self.lista[self.registro]['precio']))

        self.ventana1.mainloop()

    def anterior(self):
        if self.registro>0:
            self.registro=self.registro-1
            self.codigo.set(str(self.lista[self.registro]['codigo']))
            self.descripcion.set(self.lista[self.registro]['descripcion']
)
            self.precio.set(str(self.lista[self.registro]['precio']))

    def siguiente(self):
        if self.registro<len(self.lista)-1:
            self.registro=self.registro+1
            self.codigo.set(str(self.lista[self.registro]['codigo']))
            self.descripcion.set(self.lista[self.registro]['descripcion']
)
            self.precio.set(str(self.lista[self.registro]['precio']))

# Bloque principal
aplicacion1=Aplicacion()
```

Este será el resultado:



Ahora te muestro otra forma de poder realizarlo:

```
from urllib import request
import json
import tkinter as tk
from tkinter import ttk

class Aplicacion:
    def __init__(self):
        self.ventana1=tk.Tk()
        self.label1=ttk.Label(text="Código:", width=25)
        self.label1.grid(column=0, row=0, padx=10, pady=10)
        self.label2=ttk.Label(text="", width=25)
        self.label2.grid(column=1, row=0, padx=10, pady=10)

        self.label3=ttk.Label(text="Descripción:", width=25)
        self.label3.grid(column=0, row=1, padx=10, pady=10)
        self.label4=ttk.Label(text="", width=25)
        self.label4.grid(column=1, row=1, padx=10, pady=10)

        self.label5=ttk.Label(text="Precio:", width=25)
        self.label5.grid(column=0, row=2, padx=10, pady=10)
        self.label6=ttk.Label(text="", width=25)
        self.label6.grid(column=1, row=2, padx=10, pady=10)

        self.boton1=ttk.Button(self.ventana1, text="Anterior",
command=self.anterior, width=25)
        self.boton1.grid(column=0, row=3, padx=10, pady=10)
        self.boton2=ttk.Button(self.ventana1, text="Siguiente",
command=self.siguiente, width=25)
        self.boton2.grid(column=1, row=3, padx=10, pady=10)
        self.articulos=[]
        self.recuperar_articulos()
        self.indice=0
        self.mostrar_articulo()
        self.ventana1.mainloop()

    def anterior(self):
```

```
        if self.indice>0:
            self.indice-=1
            self.mostrar_articulo()

    def siguiente(self):
        if self.indice<len(self.articulos)-1:
            self.indice+=1
            self.mostrar_articulo()

    def recuperar_articulos(self):
        pagina=request.urlopen("http://localhost/pythonya/retornararticul
os.php")
        datos=pagina.read().decode("utf-8")
        self.articulos=json.loads(datos)

    def mostrar_articulo(self):
        if len(self.articulos)>0:
            self.label2.config(text=self.articulos[self.indice]['codigo']
)
            self.label4.config(text=self.articulos[self.indice]['descripc
ion'])
            self.label6.config(text=self.articulos[self.indice]['precio']
)

aplicacion1=Aplicacion()
```

Capítulo 317.- Procesar archivos con formato JSON en Python módulo json – 4

El sitio web

https://jsonplaceholder.typicode.com/

Se puede utilizar para recuperar datos de diferentes formatos (JSON por ejemplo) y probar nuestros algoritmos.

Confeccionar una aplicación en Python que recupere el archivo JSON de la siguiente dirección web:

https://jsonplaceholder.typicode.com/posts

Nos retorna un archivo JSON con un formato similar a:


```
[
  {
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere repellat provident occaecati excepturi
optio reprehenderit",
    "body": "quia et suscipit\nsuscipit recusandae consequuntur
expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum
rerum est autem sunt rem eveniet architecto"
  },
  {
    "userId": 1,
    "id": 2,
    "title": "qui est esse",
    "body": "est rerum tempore vitae\nsequi sint nihil reprehenderit
dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel
nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui
neque nisi nulla"
  }]
```


Convertir los datos recuperados a una lista y mediante un for mostrar los atributos "userID", "id", "title" y "body".

Vamos a programar:

```
from urllib import request
import json

pagina=request.urlopen("https://jsonplaceholder.typicode.com/posts")
datos=pagina.read().decode("utf-8")
lista=json.loads(datos) # Convertimos el string en una lista
for elemento in lista:
    print(f"userId: {elemento['userId']}")
    print(f"id: {elemento['id']}")
    print(f"title: {elemento['title']}")
    print(f"body:\n{elemento['body']}")
    print("-"*80)
```

Este será el resultado:

```
userId: 1
id: 1
title: sunt aut facere repellat provident occaecati excepturi optio reprehenderit
body:
quia et suscipit
suscipit recusandae consequuntur expedita et cum
reprehenderit molestiae ut ut quas totam
nostrum rerum est autem sunt rem eveniet architecto
--------------------------------------------------------------------------------
userId: 1
id: 2
title: qui est esse
body:
est rerum tempore vitae
sequi sint nihil reprehenderit dolor beatae ea dolores neque
fugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis
qui aperiam non debitis possimus qui neque nisi nulla
--------------------------------------------------------------------------------
userId: 1
id: 3
title: ea molestias quasi exercitationem repellat qui ipsa sit aut
body:
et iusto sed quo iure
voluptatem occaecati omnis eligendi aut ad
voluptatem doloribus vel accusantium quis pariatur
molestiae porro eius odio et labore et velit aut
--------------------------------------------------------------------------------

......

--------------------------------------------------------------------------------
userId: 10
id: 98
title: laboriosam dolor voluptates
body:
doloremque ex facilis sit sint culpa
soluta assumenda eligendi non ut eius
sequi ducimus vel quasi
veritatis est dolores
--------------------------------------------------------------------------------
userId: 10
id: 99
title: temporibus sit alias delectus eligendi possimus magni
body:
quo deleniti praesentium dicta non quod
aut est molestias
molestias et officia quis nihil
itaque dolorem quia
--------------------------------------------------------------------------------
userId: 10
id: 100
title: at nam consequatur ea labore ea harum
body:
cupiditate quo est a modi nesciunt soluta
ipsa voluptas error itaque dicta in
autem qui minus magnam et distinctio eum
accusamus ratione error aut
--------------------------------------------------------------------------------
```

Capítulo 318.- Procesar archivos con formato JSON en Python módulo json – 5

Hacer la misma actividad que el problema anterior con el recurso que devuelve la dirección:

https://jsonplaceholder.typicode.com/users


```
{
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",
  "address": {
    "street": "Kulas Light",
    "suite": "Apt. 556",
    "city": "Gwenborough",
    "zipcode": "92998-3874",
    "geo": {
      "lat": "-37.3159",
      "lng": "81.1496"
    }
  },
  "phone": "1-770-736-8031 x56442",
  "website": "hildegard.org",
  "company": {
    "name": "Romaguera-Crona",
    "catchPhrase": "Multi-layered client-server neural-net",
    "bs": "harness real-time e-markets"
  }
},
{
  "id": 2,
  "name": "Ervin Howell",
  "username": "Antonette",
  "email": "Shanna@melissa.tv",
  "address": {
    "street": "Victor Plains",
    "suite": "Suite 879",
    "city": "Wisokyburgh",
    "zipcode": "90566-7771",
    "geo": {
      "lat": "-43.9509",
      "lng": "-34.4618"
    }
  },
  "phone": "010-692-6593 x09125",
  "website": "anastasia.net",
  "company": {
    "name": "Deckow-Crist",
    "catchPhrase": "Proactive didactic contingency",
    "bs": "synergize scalable supply-chains"
  }
},
```

Vamos a programar:

```
from urllib import request
import json

pagina=request.urlopen("https://jsonplaceholder.typicode.com/users")
datos=pagina.read().decode("utf-8")
lista=json.loads(datos) # Convertimos el string en una lista
for elemento in lista:
    print(f"id: {elemento['id']}")
    print(f"name: {elemento['name']}")
    print(f"username: {elemento['username']}")
    print(f"email:{elemento['email']}")
    print("address")
    print(f"street:{elemento['address']['street']}")
    print(f"suite:{elemento['address']['suite']}")
    print(f"city:{elemento['address']['city']}")
    print(f"zipcode:{elemento['address']['zipcode']}")
    print("geolocalización")
    print(f"lat:{elemento['address']['geo']['lat']}")
    print(f"lng:{elemento['address']['geo']['lng']}")
    print(f"phone:{elemento['phone']}")
    print(f"website:{elemento['website']}")
    print("company")
    print(f"name:{elemento['company']['name']}")
    print(f"catchPhrase:{elemento['company']['catchPhrase']}")
    print(f"bs:{elemento['company']['bs']}")
    print("-"*80)
```

Este será el resultado:

```
id: 1
name: Leanne Graham
username: Bret
email:Sincere@april.biz
address
street:Kulas Light
suite:Apt. 556
city:Gwenborough
zipcode:92998-3874
geolocalización
lat:-37.3159
lng:81.1496
phone:1-770-736-8031 x56442
website:hildegard.org
company
name:Romaguera-Crona
catchPhrase:Multi-layered client-server neural-net
bs:harness real-time e-markets
--------------------------------------------------
```

. . . . . .

Hasta el último:

```
id: 10
name: Clementina DuBuque
username: Moriah.Stanton
email:Rey.Padberg@karina.biz
address
street:Kattie Turnpike
suite:Suite 198
city:Lebsackbury
zipcode:31428-2261
geolocalización
lat:-38.2386
lng:57.2232
phone:024-648-3804
website:ambrose.net
company
name:Hoeger LLC
catchPhrase:Centralized empowering task-force
bs:target end-to-end models
-----------------------------------------------
```

Capítulo 319.- Serialización y deserialización en Python (módulo json)

En el concepto anterior trabajamos con la deserialización de recurso JSON localizados en un servidor y su posterior transformación a listas con elementos de tipo diccionario.

Definiciones básicas.

- Serialización: Consiste en convertir un objeto de Python (normalmente una lista o diccionario) en un string.
- Deserialización: Consiste en convertir un string en un objeto de Python (normalmente una lista o diccionario).

Como vemos son los dos caminos posibles para transformar una estructura de datos en otra estructura.

Problema

Definir un string en Python con la estructura interna de un archivo JSON.

Deserializar el string y convertirlo a una lista de Python. Luego volver a serializar la lista a un string.

Transformación de datos entre Python y JSON

Los tipos de datos en JSON son muy similares a los tipos de datos en Python, pero no son exactamente iguales.

La conversión en los procesos de serializar y deserializar son:

Python	JSON
dic	object
list o tupla	array
str	string
int y float	number
True	true
False	false
None	null

Vamos a programar:

eejrcicio319.py > ...

```
1   import json
2
3   cadena1="""
4       [
5           {
6               "codigo":"1",
7               "descripcion":"patatas",
```

```
8            "precio":"12"
9        },
10       {
11           "codigo":"2",
12           "descripcion":"naranjas",
13           "precio":"25"
14       }
15   ]
16 """
17 print(type(cadena1))
18 lista=json.loads(cadena1)
19 print(lista)
20 print(type(lista))
21
22 cadena2=json.dumps(lista)
23 print(cadena2)
24 print(type(cadena2))
```

Este será el resultado:

```
<class 'str'>
[{'codigo': '1', 'descripcion': 'patatas', 'precio': '12'}, {'codigo'
: '2', 'descripcion': 'naranjas', 'precio': '25'}]
<class 'list'>
[{"codigo": "1", "descripcion": "patatas", "precio": "12"}, {"codigo"
: "2", "descripcion": "naranjas", "precio": "25"}]
<class 'str'>
```