



python 3

CURSO COMPLETO DE PYTHON



Con Dimas

[Descripción breve](#)

Aprende a programar en Python con este curso desde cero completo para principiantes actualizado a 2022. Aprende cómo instalar Python en Windows y cómo configurar Visual Studio Code para programar con Python paso a paso. Escanea este código QR para seguir sus videotutoriales.

Pere Manel Verdugo Zamora
pereverdugo@gmail.com

1.- Introducción

1.- ¿Qué es Python?

- Lenguaje interpretado de alto nivel.
- Multipropósito.
- Orientado a objeto (o no).
- Open Source, con una comunidad enorme



2.- ¿Para qué sirve Python?



Desarrollo web (Backend)



Machine Learning, Data Science



Desarrollo de Apps



Aplicaciones de Escritorio

Networking, herramientas línea de comandos, testing, web scraping...

3.- ¿Cómo tomar este curso?

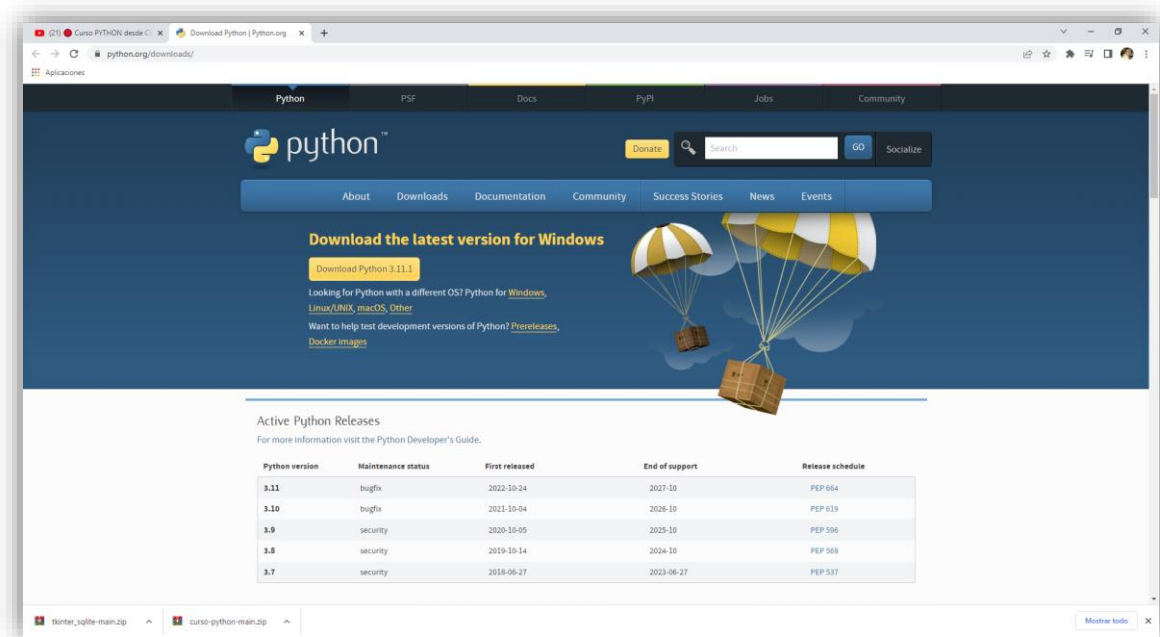
- De forma activa, no te limites a escuchar.
- Si eres nuevo en el tema, ve paso a paso.
- Explora más allá de esta lista de reproducción.



Instalar Python con la última versión.

Normalmente ya en Windows ya tenemos una versión de Python pero para obtener la última versión estable de Python vamos a acceder a la siguiente página web:

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

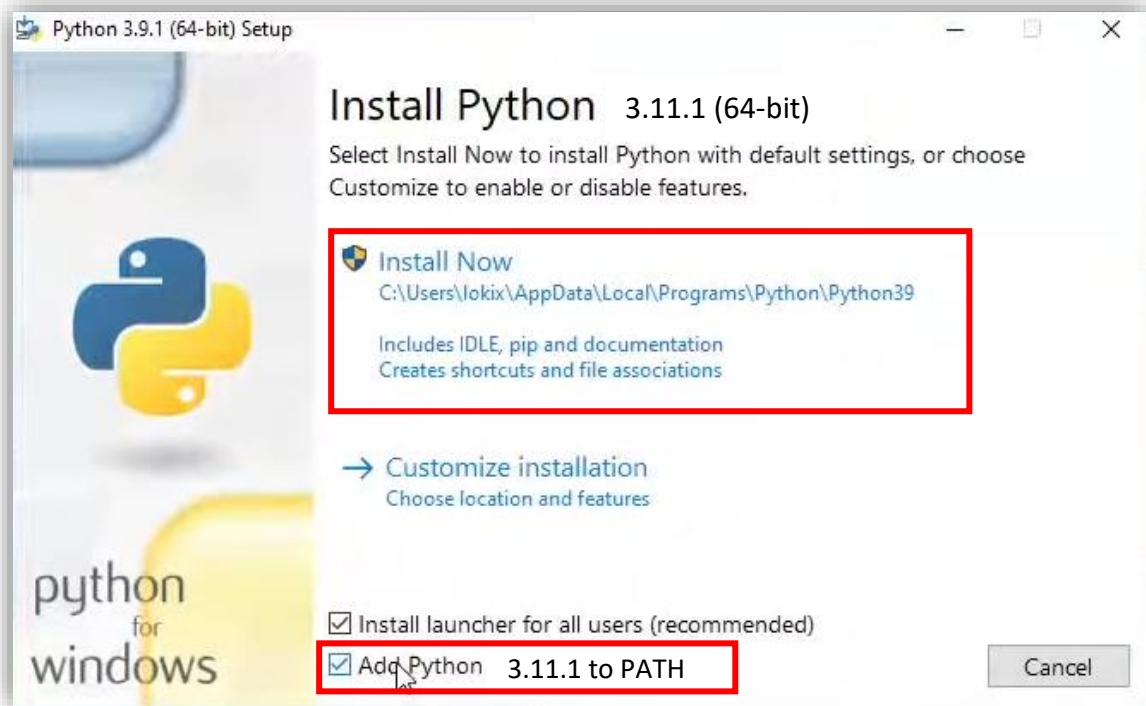


Seleccionaremos el botón Download Python 3.11.1.

Y accederemos a la carpeta de descargas.



Instalaremos dicho archivo.



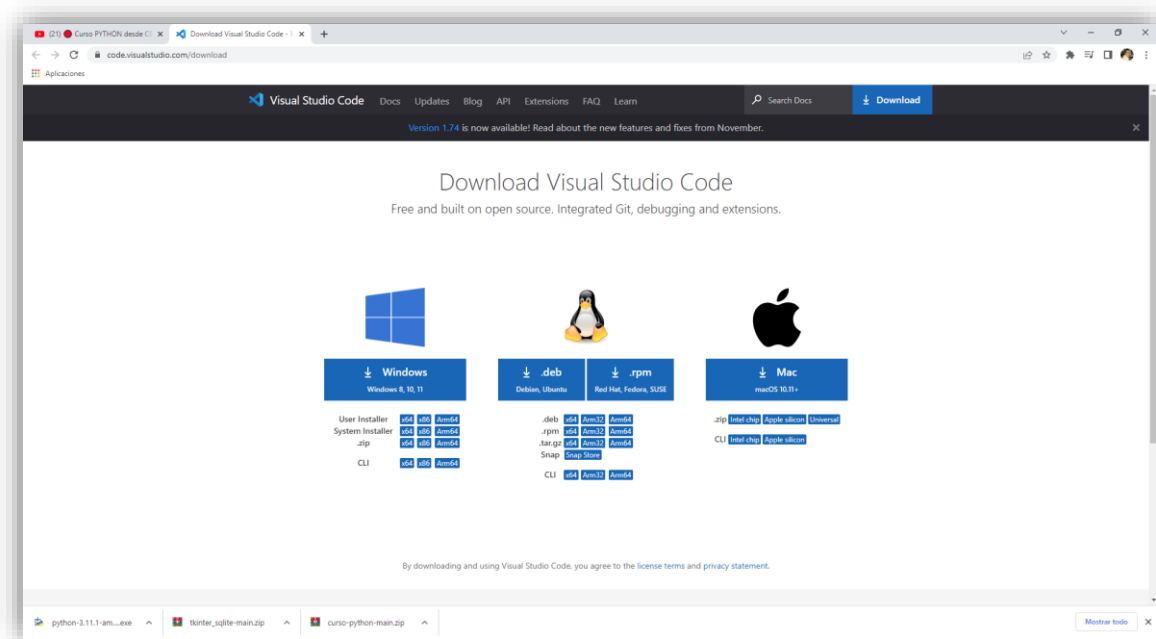
Muy importante la casilla Add Python 3.11.1 to PATH la tenemos que activar, para que podamos ejecutar Python desde cualquier carpeta de nuestro ordenador.

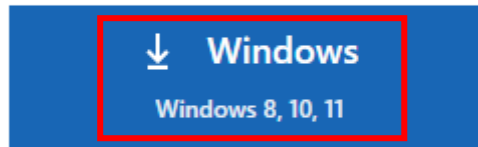
Seleccionaremos Install Now.

Sigue con los siguientes pasos con los valores por defecto hasta finalizar la instalación.

Para poder editar nuestros programas vamos a editar con Visual Studio Code que lo podrás descargar desde la siguiente página web.

<https://code.visualstudio.com/download>





User Installer	x64	x86	Arm64
System Installer	x64	x86	Arm64
.zip	x64	x86	Arm64
CLI	x64	x86	Arm64

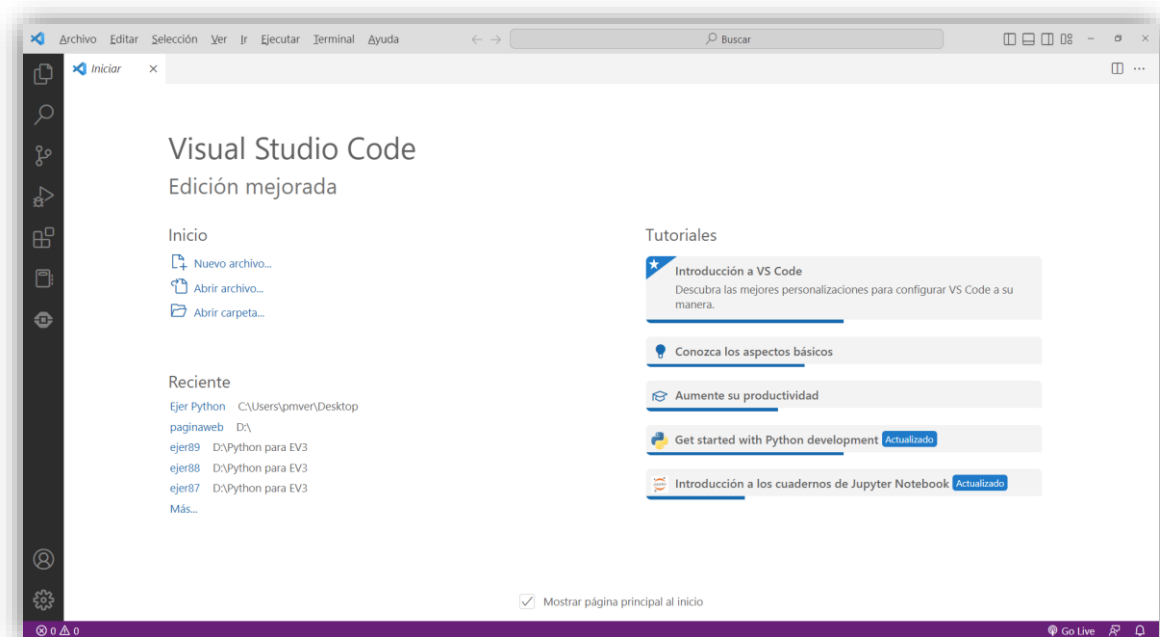
Descargaremos la versión para Windows.

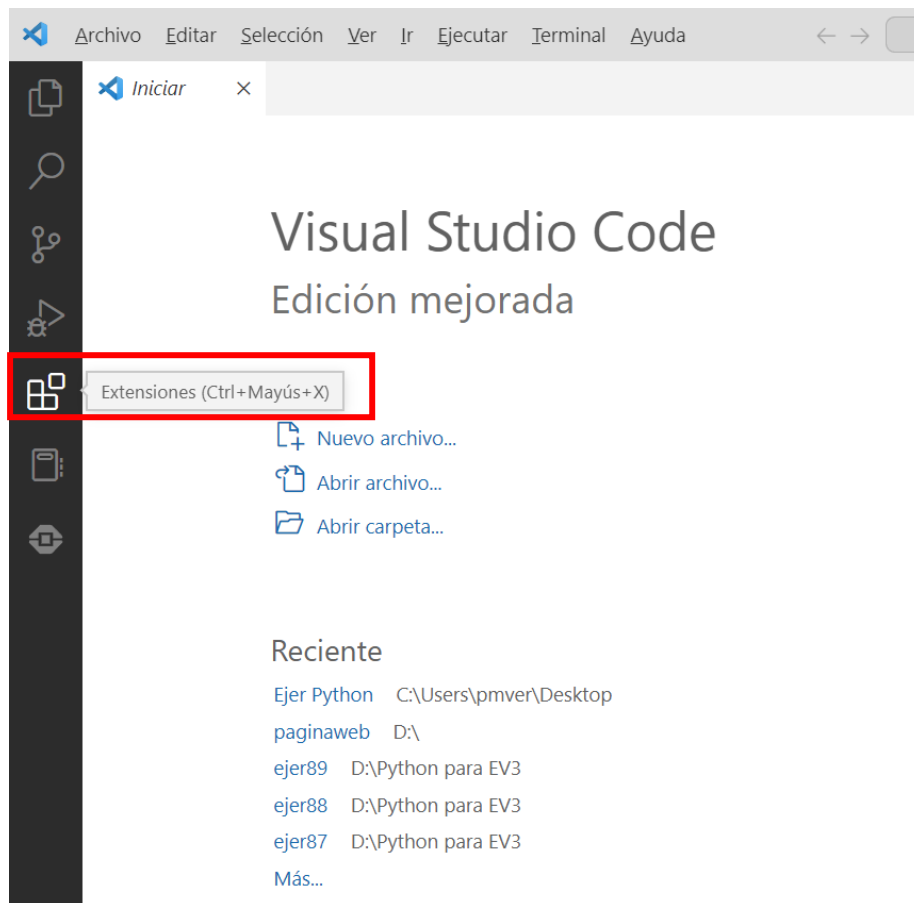
Vamos a la carpeta de descargas.



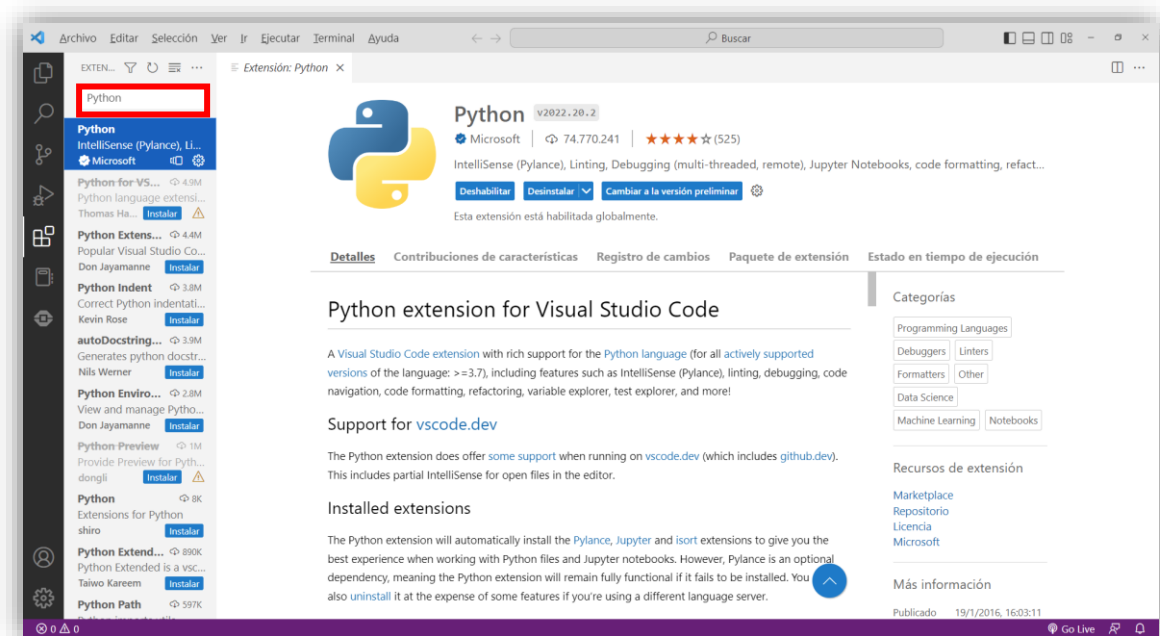
Procederemos a su instalación.

Vamos a ejecutar el programa una vez instalado.





Nos iremos a extensiones para agregar la de Python.

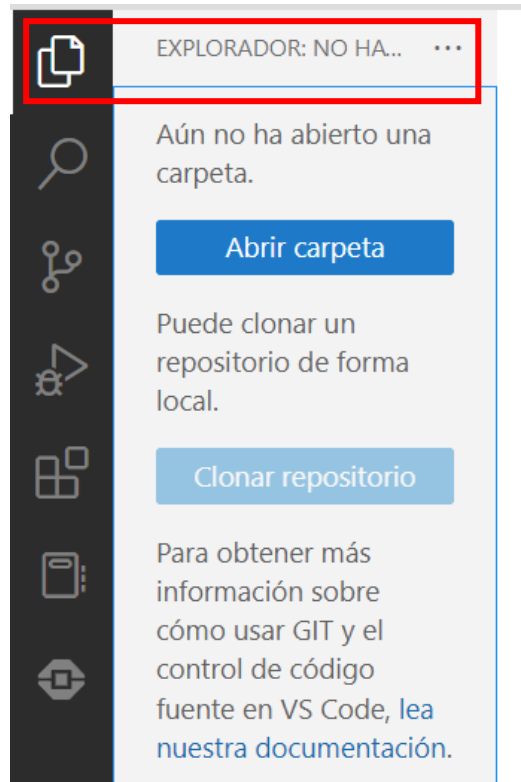


En el buscador escribiremos Python y procederemos a su instalación.

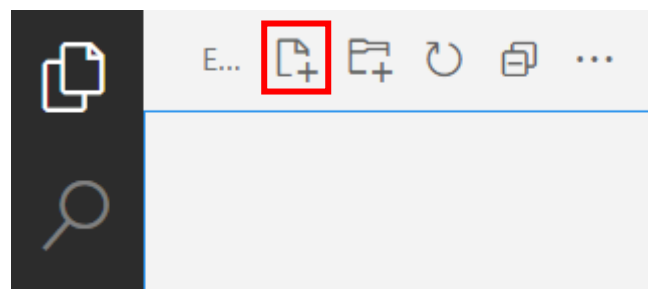
Instalar



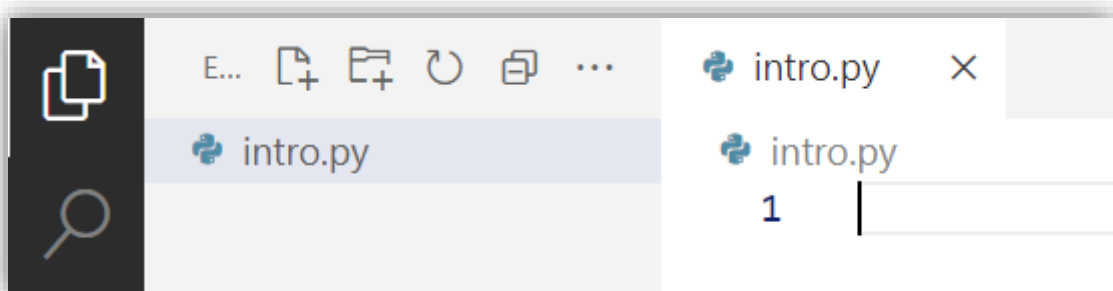
En mi caso no se muestra el botón instalar porque ya la tengo instalada.



Seleccionaremos Explorador y el botón Abrir Carpeta, donde seleccionaremos una carpeta que hemos creado con anterioridad donde irán todos los proyectos del curso.



Vamos a crear nuestro primer proyecto este se llamara intro.py, todo nuestros proyectos tienen que tener una extensión .py.

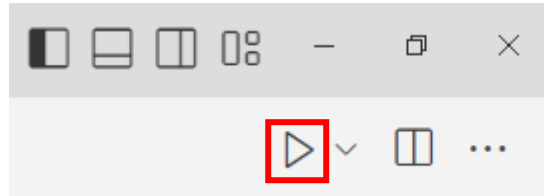


Vamos a escribir el siguiente código:

```
intro.py
1 print("Hola Mundo")
```

Con la sentencia print nos mostrara el texto que se encuentra entre los paréntesis.

Para ejecutar tenemos dos opciones:

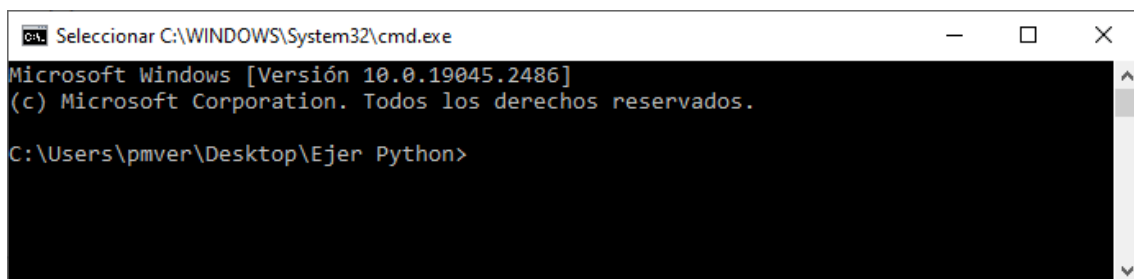


En la parte superior derecha encontramos el botón de ejecutar.

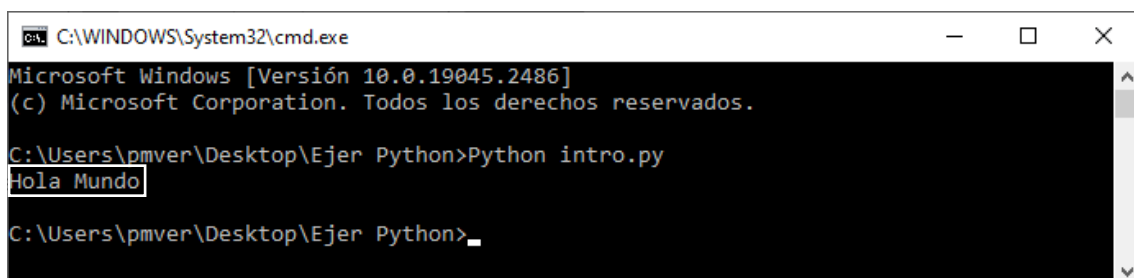


En la parte inferior de mostrará la consola con el correspondiente texto.

La segunda opción consiste en presionar las teclas Ctrl + Shift + C, con ello se nos abrirá el terminal en la carpeta donde se encuentra nuestro proyecto.



Para ejecutar el programa podemos escribir: Python intro.py



También funciona escribiendo py intro.py, de este modo no tenemos que escribir la palabra completa de Python. Perfecto todo funciona correctamente y ya podemos seguir con el curso.

2.- Tipo de DATOS, VARIABLES y Operaciones Aritméticas en Python.

A continuación vamos a realizar un nuevo proyecto.

Capitulo2.py

```
1  #  Capítulo 2: Variables, Tipos de Datos y Operadores Aritméticos.
```

Este código aparece de color verde y al principio hay una almohadilla, esto significa que es un comentario, que podemos agregar a un programa para comentar lo que hace parte del código, y si en un futuro tenemos que modificar el programa u otra persona tuviera que modificar el programa estos comentarios le podrían servir de ayuda. Cuando ejecutamos un programa las líneas que son comentarios son ignorados por el mismo.

Otro ejemplo de comentario de varias líneas.

Capitulo2.py

```
1  """
2  Esto es un comentario de varias líneas
3  para ello tiene que ir entre tres comillas
4  """
```

También es muy útil si queremos que parte del código que estamos escribiendo no se ejecute.

Variables

¿Qué son las variables? Las variables son un lugar de la memoria de nuestro ordenador donde vamos a guardar una información en concreto. Esta información a lo largo de nuestro programa la podremos leer y modificar.

Nosotros solo tenemos que darle un nombre a la variable y asignarle un valor.

Tipos de datos

Son los tipos de información que podemos guardar en las variables. Podemos guardar números enteros, números decimales, textos, etc.

Estas variables se almacenarán de forma distinta según el tipo que sea.

Capitulo2.py > ...

```
1  #  Tipos de datos numéricos
2
3  numero_entero = 1
4  print(numero_entero)
5
6  numero_decimal = 2.46
7  print(numero_decimal)
8
```

```

9     numero_entero = numero_entero + 2
10    print(numero_entero)
11
12    numero = numero_entero + numero_decimal
13    print(numero)
14
15    numero_entero += 2
16    print(numero_entero)
17
18    variable_numerica = 20
19    variable_numerica = variable_numerica - 10
20    print(variable_numerica)
21
22    variable = 10
23    variable = variable / variable_numerica
24    print(variable)
25
26    numero = 2/2+3
27    print(numero)
28
29    multiplicacion = 2*3
30    print(multiplicacion)
31    multiplicacion = numero_entero*numero_decimal
32    print(multiplicacion)
33
34    potencia = 2**3
35    modulo = 11%2
36    print(potencia)
37    print(modulo)

```

Línea 3 asignamos a la variable `numero_entero` el valor de 1.

Línea 4 mostramos por consola el valor de la variable `numero_entero`.

Línea 6 Asignamos a la variable `numero_decimal` el valor de 2.46.

Línea 7 mostramos por consola el valor de la variable `numero_decimal`.

Línea 9 a la variable `número_entero` le incrementamos su valor en 2.

Línea 10 Mostramos el valor actual de la variable `numero_entero`.

Línea 12 a la variable `numero` le asignamos el valor `numero_entero` más `numero_decimal`.

Línea 13 Mostramos el valor de la variable `número` por consola, esta variable será de tipo decimal.

Línea 15 La variable `numero_entero` la incrementamos en 2. Las siguientes expresiones hacen lo mismo. **`numero_entero = numero_entero + 2`** que **`numero_entero += 2`**.

Línea 18 A la variable `_numerica` le asignamos el valor de 20.

Línea 19 A la variable `_numerica` le restamos a su valor 10.

Línea 20 Mostramos el valor de la variable `_numerica` por consola.

Línea 22 Asignamos a la variable el valor de 10.

Línea 23 A la variable le asignamos el valor que tiene la variable dividido por el valor de la variable `variable_numerica`.

Línea 24 Mostramos por consola el valor de la variable.

Línea 26 a la variable `numero` le asignamos el valor de la siguiente operación $2 / 2 + 3$.

Línea 27 mostramos el valor de la variable `numero` por consola.

Línea 29 asignamos a la variable `multiplicación` el resultado de $2 * 3$ (Multiplicación)

Línea 30 mostramos su valor por consola.

Línea 31 A la variable `multiplicacion` le asignamos el valor de multiplicar las variables `numero_entero` por `numero_decimal`.

Línea 32 muestra por consola el valor de la variable `multiplicación`.

Línea 34 asignamos a la variable `potencia` el valor de 2^3 que se representan con `2 **`.

Línea 35 asignamos a la variable `modulo` en resto de la división entre 11 y 2.

Línea 36 Mostramos por consola el valor de la variable `potencia`.

Línea 37 Mostramos por consola el valor de la variable `modulo`.

Vamos a ejecutar para ver el resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo2.py"
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo2.py"
1
2.46
3
5.46
5
10
1.0
4.0
6
12.3
8
1
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Capitulo2.py > ...

```
1  # Tipos de Datos: Texto
2  texto = "Esta es nuestra frase"
3  text = "Texto 'entre comillas' guay"
4  textt = 'Texto "entre comillas" guay'
5  txt = 'Hola'
6
7  print(texto)
8  print(text)
9  print(textt)
10 print(txt)
11
12 texto_con_formato = """ Esto es un texto
13 |     con formato
14 | """
15 print(texto_con_formato)
```

Línea 2 Asignamos a la variable texto “Esta es nuestra frase”.

Línea 3 Asignamos a la variable text “Texto ‘entre comillas’ guay” queremos que se muestren las comillas simples.

Línea 4 Asignamos a la variable textt ‘Texto “entre comillas” guay’ queremos que se muestren las comillas dobles.

Línea 5 Asignamos a la variable txt ‘Hola’

Línea 7 mostramos por consola el valor de la variable texto.

Línea 8 mostramos por consola el valor de la variable text.

Línea 9 mostramos por consola el valor de la variable textt.

Línea 10 mostramos por consola el valor de la variable txt.

Línea 12-14 Asignamos un texto a la variable texto_con_formato, que para conservar el formato estas tienen que ir entre 3 comillas dobles.

Línea 15 mostramos por consola en valor de la variable texto_con_formato.

Ejecutamos el código para ver el resultado por consola.

```
Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo2.py"
```

```
Esta es nuestra frase
```

```
Texto 'entre comillas' guay
```

```
Texto "entre comillas" guay
```

```
Hola
```

```
    Esto es un texto
```

```
        con formato
```

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo2.py"
```

```
Esta es nuestra frase
```

```
Texto 'entre comillas' guay
```

```
Texto "entre comillas" guay
```

```
Hola
```

```
    Esto es un texto
```

```
        con formato
```

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo2.py"
```

```
Esta es nuestra frase
```

```
Texto 'entre comillas' guay
```

```
Texto "entre comillas" guay
```

```
Hola
```

```
    Esto es un texto
```

```
        con formato
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

3.- ¿Cómo usar CONDICIONALES en Python?

```
1  #  Entrada de datos del usuario. Identificación del tipo de datos.
2  edad = input("Introduce tu edad por favor: ")
3  tipo_de_dato = type(edad)
4  print(edad)
5  print(tipo_de_dato)
```

En la línea 1 añadimos un comentario, recuerda este es ignorado a la hora de ejecutarse.

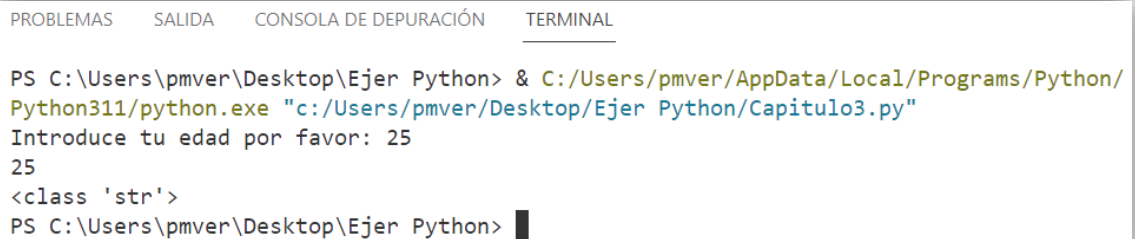
En la línea 2 mediante la sentencia input hacemos que el usuario pueda introducir un dato que después se almacenará en la variable edad.

En la línea 3 hacemos que en la variable tipo_de_dato se almacene el tipo de variable que es edad.

En la línea 4 mostramos el contenido de la variable edad.

En la línea 5 mostramos el contenido de la variable tipo_de_dato donde se almacena el tipo de variable que es edad.

Si ejecutamos el programa este será el resultado en consola:



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
Introduce tu edad por favor: 25
25
<class 'str'>
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```
1  #  BOOLEANOS, IF
2  verdadero = True
3  falso = False
4
5  if(verdadero == True):
6  |    print("Soy verdadero")
7
8  if(falso == True):
9  |    print("Eiiii")
10
11 if(falso == False):
12 |    print("Efectivamente, falso es falso")
```

En la línea 1 agregamos un comentario.

En la línea 2 definimos una variable de tipo booleano, llamada verdadero y con el valor True.

En la línea 3 definimos una variable de tipo booleano, llamada falso y con el valor False.

En la línea 5 con un condicional comparamos si la variable verdadero es igual a True.

En la línea 6 en caso de que se cumpla la condición anterior mostrará un mensaje por consola “Soy verdadero”.

En la línea 8 con otro condicional comparamos si la variable falso es igual a True.

En la línea 9 en caso que la condición se cumpla imprimirá por consola “Eiiii” de lo contrario no imprimirá nada.

En la línea 11 con otro condicional comparamos si la variable falso es igual a False.

En la línea 12 si la condición anterior se cumple imprimirá por consola el siguiente mensaje, “Efectivamente, falso es falso”).

Si ejecutamos el programa este será el resultado por consola:



```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
Soy verdadero
Efectivamente, falso es falso
PS C:\Users\pmver\Desktop\Ejer Python>
```

```
1  # Operadores de Comparación ==, !=, <, >, >=, <=
2  edad = input("Introduce tu edad por favor: ")
3  edad = int(edad)
4
5  if(edad >= 18):
6      print("Eres mayor de edad, puedes entrar")
7  elif(edad < 0):
8      print("Eso es imposible")
9  elif(edad < 15):
10     print("Eres muy muy pequeño")
11 else:
12     print("Eres menor, no puedes entrar")
13
14 print("Fin del if")
```

Línea 1 agregamos un comentario.

Línea 2 Por mediación de la sentencia input hacemos que el usuario introduzca su edad, esta se almacena en la variable edad como dato String.

Línea 3 El dato que contiene la variable edad lo convertimos en int en entero, ya que con el tipo String no podemos comparar si este dato es mayor o menor.

Línea 5 con un condicional comparamos si la variable edad es mayor o igual a 18.

Línea 6 de ser así mostrará un mensaje por consola “Eres mayor de edad, puedes entrar”.

Línea 7 si no se cumple la condición comparamos de nuevo si edad es menor a 0.

Línea 8 de ser así mostrará un mensaje por consola “Eso es imposible”.

Línea 9 si no se cumple la condición comparamos de nuevo si edad es menor de 15.

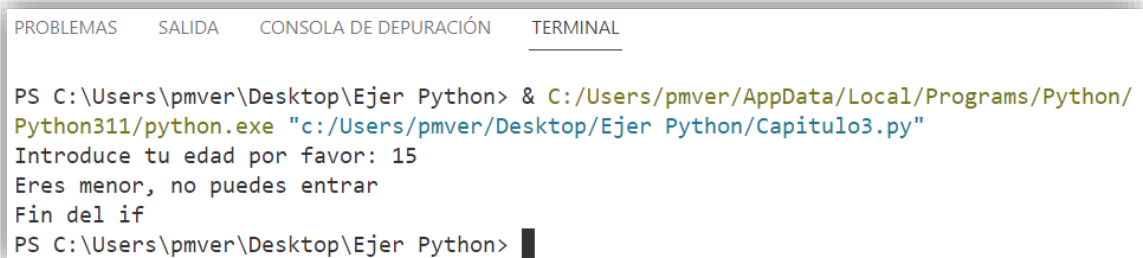
Línea 10 Si se cumple la condición anterior mostrará por consola “Eres muy muy pequeño”.

Línea 11 si no se han cumplido ninguna de las condiciones anteriores

Línea 12 Mostrará el siguiente mensaje por consola “Eres menor no puedes entrar”)

Línea 14 Muestra por consola “Fin del fin”.

Si ejecutamos el programa este será el resultado por la consola:



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
Introduce tu edad por favor: 15
Eres menor, no puedes entrar
Fin del if
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```
1  # Operadores Lógicos and, or, not
2  edad = input("Introduce tu edad por favor: ")
3  edad = int(edad)
4  if(type(edad) == int):
5      if(edad > 120 or edad < 0):
6          print("No me lo creo")
7      elif(edad > 18 and edad < 65):
8          print("Eres adulto")
9  elif(edad < 18 and edad > 15):
10     print("Eres un adolescente")
11 else:
12     print("Pues nada")
13 else:
14     print("Tu edad debería ser un número entero")
```

En la línea 1 agregamos un comentario.

En la línea 2 por mediación de la sentencia input el usuario tiene que introducir su edad.

En la línea 3 convertimos la variable edad en tipo int.

En la línea 4 comparamos si la variable edad es de tipo int.

En la línea 5 con un condicional anidado al condicional anterior comparamos si edad es mayor de 120 o edad es menor a 0.

En la línea 6 si la condición anterior se cumple mostrara por consola el siguiente mensaje “No me lo creo”.

En la línea 7 si la condición anterior no se cumple vamos a comparar si edad en mayor de 18 y a la vez en menor de 65.

En la Línea 8 si la condición anterior se cumple mostrará por consola el mensaje “Eres adulto”

En la línea 9 si la condición anterior no se cumple compararemos si edad es menor de 18 y a la vez edad es mayor de 15.

En la línea 10 si la condición anterior se cumple, mostrará por consola "Eres un adolescente".

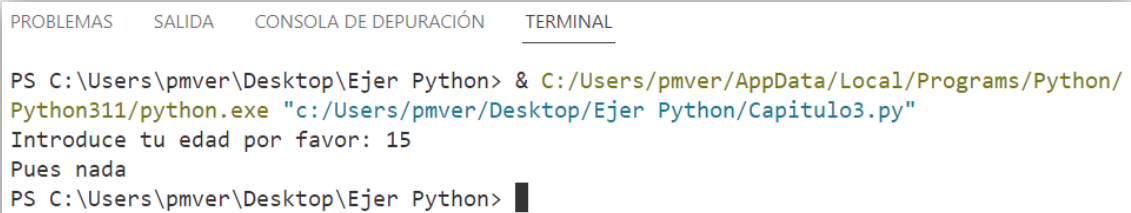
En la línea 11 si no se cumple ninguna de las condiciones de las líneas 5, 7, 9 se ejecutará la siguiente instrucción.

En la línea 12 Mostrará por consola el siguiente mensaje "Pues nada".

En la línea 13 si la condición de la línea 4 no se cumple se ejecutará la instrucción siguiente.

En la línea 14 Mostrará por consola "Tu edad debería ser un número entero".

Este será el resultado en consola cuando ejecutemos el programa:



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
Introduce tu edad por favor: 15
Pues nada
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Ejercicio práctico:

Comprobar si un número introducido por el usuario es par.

Si el usuario no introduce un número, o el número es decimal el programa debe avisar de que los datos no son correctos.

Una pista:

isnumeric -> El método isnumeric() devuelve True si todos los caracteres son numéricos (0-9), de lo contrario, False. Los exponentes como ²y ³/₄, también se consideran valores numéricos.

Num%2 = 0 -> Para controlar si un número es par o impar.

Solución:

```
1  num = input("Introduce un número: ")
2
3  if(not(num.isnumeric())):
4      print("Datos incorrectos. El número debe ser un entero")
5  else:
6      num = int(num)
7      if(num%2 == 0):
8          print("El número es par")
9      else:
10         print("El número es impar")
```

isdigit -> El método isdigit() devuelve True si todos los caracteres son dígitos, de lo contrario False. Los exponentes, como ², también se consideran dígitos.

isdecimal -> El método isdecimal() devuelve True si todos los caracteres son decimales (0-9).

4.- Funciones en Python con ejercicios resueltos.

```
1  # Definición de las funciones
2  def saludar():
3      print("Hola ¿Qué tal?")
4
5  print("Apartado 1: Funciones básicas. Función para saludar")
6  saludar()
7  saludar()
```

En la línea 2 creamos una función con la sentencia def llamada saludar.

En la línea 3 muestra el mensaje “Hola ¿Qué tal?”

En la línea 5 muestra por consola “Apartado 1: Funciones básicas. Función para saludar”.

En la línea 6 y 7 llamamos a la función saludar(), las funciones no se ejecutan si previamente son llamadas.

Cuando ejecutemos el programa este será el resultado en la consola:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
Apartado 1: Funciones básicas. Función para saludar
Hola ¿Qué tal?
Hola ¿Qué tal?
PS C:\Users\pmver\Desktop\Ejer Python>
```

```
1  # Funciones con argumentos
2  def saludar(nombre):
3      print("Hola "+nombre+" ¿Qué tal?")
4
5  print("Apartado 2: Funciones con argumentos. Saludar con nombre")
6  saludar("Marta")
7  saludar("Juan")
```

Las funciones con argumentos son aquellas funciones que cuando las llamemos le tenemos que pasar un argumento, de este modo el resultado de la función variará, según el argumento que le pasemos.

Línea 2 creamos una función con la sentencia def llamada saludar a la que le tenemos que pasar un nombre cuando la llamemos.

Línea 3 la variable nombre es el valor que mostrará cuando la llamemos con su correspondiente argumento.

En la línea 6 y 7 llamamos a la función pero con los argumentos “Marta” y “Juan” este será el resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
Apartado 2: Funciones con argumentos. Saludar con nombre
Hola Marta ¿Qué tal?
Hola Juan ¿Qué tal?
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Podrás observar que el resultado es distinto.

```
1 # Funciones con retorno
2 def suma(a, b):
3     resultado = a + b
4     return resultado
5
6 def sumaDos(a, b):
7     return a+b
8
9 print("Apartado 3: Funciones que retornan un valor. Función para sumar dos números")
10 print(suma(2, 2))
11 print(sumaDos(2, 2))
```

Las funciones con retorno son aquellas funciones que además de pasarle parámetros nos retorna un valor.

En la línea 2 creamos una función llamada suma a la que le tendremos que pasar dos parámetros.

En la línea 3 a la variable resultado le asignamos el valor de a + b.

En la línea 4 retornamos el valor de la variable resultado.

En la línea 6 creamos una segunda función llamada sumados a la que le tendremos que pasar dos parámetros.

En la línea 7 retornamos el valor de la operación a + b (El resultado de las dos funciones serán lo mismo).

En la línea 10 y 11 llamamos a las respectivas funciones pero como estas retornan valores utilizaremos la sentencia print para mostrar su resultado en la consola.

Este será el resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
Apartado 3: Funciones que retornan un valor. Función para sumar dos números
4
4
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```
1 # Funciones con argumentos con valor por defecto
2 def saludarPorDefecto(nombre="Dimas"):
3     print("Hola "+nombre+" ¿Qué tal?")
4
5 print("Apartado 4: Funciones con argumentos por defecto. Función para saludar")
6 saludarPorDefecto()
7 saludarPorDefecto("Ana")
```

En la línea 2 creamos una función llamada `saludarPorDefecto` con el parámetro `nombre="Dimas"` esto significa que al llamar esta función sin argumentos el valor por defecto será "Dimas".

En la línea 3 muestra por consola el texto "Hola " + nombre + " ¿Qué tal?" donde nombre es el valor que le pasemos por argumentos, siendo por defecto "Dimas".

En la línea 6 y 7 llamamos a la función una sin argumentos y la otra con el argumento de "Ana", cuando ejecutemos el programa este será el resultado en la consola:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
Apartado 4: Funciones con argumentos por defecto. Función para saludar
Hola Dimas ¿Qué tal?
Hola Ana ¿Qué tal?
PS C:\Users\pmver\Desktop\Ejer Python>

1  # Funciones que retornan varios valores
2  def sumaYResta(a, b):
3      suma = a + b
4      resta = a - b
5      return suma, resta
6
7  print("Apartado 5: Funciones con múltiples retornos. Función para saludar")
8  resultadoSuma, resultadoResta = sumaYResta(10, 5)
9  print("El resultado de la suma es: " + str(resultadoSuma) + "\n Y el de la resta: " + str(resultadoResta))
```

Podemos realizar una función y esta que retorne más de un valor.

En la línea 2 creamos una función `sumaYResta` a la que le tendremos que pasar dos parámetros.

En la línea 3 a la variable `suma` le pasamos la suma de las variables `a + b` (Argumentos que añadiremos cuando llamemos la función).

En la línea 4 a la variable `resta` le pasamos la resta de las variables `a - b`.

En la línea 5 retornamos los valores de `suma` y `resta`.

En la línea 8 definimos dos variables `resultadoSuma` y `resultadoResta` y le asignamos los resultados que nos retorna la función, es importante repetir el orden de las variables con respecto a los valores retornados.

En la línea mostramos el resultado por consola.

El `\n` lo que hace es un salto de línea.

Las variables al ser de tipo entero no podremos concatenarlas con texto, si no las reconvertimos a tipo String, esto se hace con `str`.

Este será el resultado en consola:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
```

Apartado 5: Funciones con múltiples retornos. Función para saludar

El resultado de la suma es: 15

Y el de la resta: 5

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Ejercicio práctico:

Función máximo -> Dados dos números, la función debe encontrar cuál de los dos es el más grande y retornarlos. Extra: Se debe comprobar que los argumentos de la función sean de tipo int o float. Si alguno de los dos no lo es, mostrar un mensaje de error y retornar False.

Solución:

```
1 def maximo(num1, num2):
2     if((type(num1)==int or type(num1)==float) and (type(num2)==int or type(num2)==float)):
3         if(num1 > num2):
4             return num1
5         else:
6             return num2
7     else:
8         print("Los argumentos no son válidos")
9         return False
10
11 print("Apartado 6: Ejercicio 1. Encontrar el máximo de dos números")
12
13 num = maximo(5, 1)
14 if(type(num) != bool):
15     print("El máximo es: " + str(num))
16
17 num = maximo("hola", 2)
18 if(type(num) != bool):
19     print("El máximo es: " + str(num))
```

Ejercicio práctico:

Mini calculadora. Pedirle al usuario una operación y dos números. Las operaciones pueden ser suma, resta y potencia. Si introduces una operación diferente a estas, mostrar un mensaje de error. Si la operación es correcta, mostrará el resultado.

Solución:

```
1 print("Apartado 6: Ejercicio 2. Mini calculadora")
2
3 def calculadora():
4     operacion = input("¿Qué operación quieres hacer? Las operaciones posibles son:\n-suma\n-resta\n-potencia\n")
5     num1 = float(input("Introduce el primer valor: "))
6     num2 = float(input("Introduce el segundo valor: "))
7     if(operacion == "suma"):
8         print(num1 + num2)
9     elif(operacion == "resta"):
10        print(num1 - num2)
11    elif(operacion == "potencia"):
12        print(num1**num2)
13    else:
14        print("Operación errónea. Las operaciones posibles son:\n-suma\n-resta\n-potencia\n")
15
16 calculadora()
```

5.- ¿Cómo hacer un BUCLE en PYTHON? (Con ejemplos incluidos)

```
1  # Breve introducción a las listas
2  numeros = [0, 1, 2, 3, 4, 5]
3  nombres = ["Jorge", "Juan", "Javier", "Jairo"]
4  variada = ["Juan", 3, False, True, 2, "arroz"]
5
6  longitudnumeros = len(numeros)
7  longitudnombres = len(nombres)
8  longitudvariada = len(variada)
9
10 print(longitudnumeros)
11 print(numeros)
12 print(longitudnombres)
13 print(nombres)
14 print(longitudvariada)
15 print(variada)
```

Una lista es una variable que contiene más de un valor, estos se almacenan en un índice que empieza por 0.

Línea 2 en la lista números almacenamos los siguientes números:

números						
índice	0	1	2	3	4	5
valores	0	1	2	3	4	5

Línea 3 en la lista nombres almacenaremos los siguientes datos:

nombres				
índice	0	1	2	3
valores	"Jorge"	"Juan"	"Javier"	"Jairo"

Línea 4 en la lista variada almacenamos datos de diferente tipo:

variada					
índice	0	1	2	3	5
valores	"Juan"	3	False	True	"arroz"

En las líneas 6, 7 y 8 con la sentencia len() podemos asignarle el número de elementos a las variables longitudnumeros, longitudnombre y longitudvariada.

En las líneas 10, 11, 12, 13, 14 y 15 mostramos por consola el número de elementos que tienen cada lista, así como su contenido.

Vamos a ejecutar y este será el resultado en la consola:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
6
[0, 1, 2, 3, 4, 5]
4
['Jorge', 'Juan', 'Javier', 'Jairo']
6
['Juan', 3, False, True, 2, 'arroz']
PS C:\Users\pmver\Desktop\Ejer Python>
```

```
1  # Elementos de una lista
2  numeros = [0, 1, 2, 3, 4, 5]
3  nombres = ["Jorge", "Juan", "Javier", "Jairo"]
4  print("Primer elemento de numeros")
5  longitudnombres = len(nombres)
6  print(numeros[0])
7  print("Último elemento de nombres")
8  print(nombres[longitudnombres-1])
```

En la líneas 2 y 3 almacenamos datos en las listas números y nombres.

En la línea 5 asignamos a la variable longitudnombres el número de elementos que tiene la lista nombres.

En la línea 6 imprimimos por consola el primer elemento de la lista número ya que entre corchetes seleccionamos el índice 0.

En la línea 8 imprimimos por consola el último elemento de la lista nombres, es la longitud de la lista menos 1, ya que las listas empiezan por el índice 0 y el último elemento se encuentra en el índice 5, pero la lista tienen 6 elementos.

Si ejecutamos el programa esta será la información que se mostrará en la consola:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
Primer elemento de numeros
0
Último elemento de nombres
Jairo
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```
1  # Bucles: Bucle for
2  ✓ for x in range(5):
3  |     print(x)
```

Los bucles for tienen la siguiente sintaxis:

for (variable) in range(número de ciclos) seguido de dos puntos :

En la línea 2 hacemos un bucle que se repetirá 5 veces, el valor de la variable x en cada bucle obtendrá los valores de 0, 1, 2, 3, 4.

Todo lo que afecta al bucle tiene que ir a 4 espacios hacia la derecha, esto se denomina estructuración dentada.

En la línea 3 mostramos por consola los valores de la variable x.

Si ejecutamos el programa este será el resultado en la consola:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
0
1
2
3
4
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```
1  for letra in "Dimas":
2      print(letra)
```

Podemos hacer un bucle para que realice un recorrido por todos los caracteres de un string.

En la línea 1 la variable letra en cada bucle irá asumiendo carácter por carácter de la palabra "Dimas".

En la línea 2 dentro del bucle imprime el valor de la variable letra que se le asigna a cada bucle.

Si ejecutamos el programa, este será el resultado en la consola:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
D
i
m
a
s
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```

1  # Mini ejemplo, imprimir solo las vocales de una palabra
2
3  for letra in "cocodrilo":
4      if (letra == "a" or letra == "e" or letra == "i" or letra == "o" or letra == "u"):
5          print(letra)
6      else:
7          print("No es una vocal")

```

En este ejemplo vamos a realizar un recorrido de carácter por carácter de la palabra “cocodrilo” que irá asumiendo la variable letra en cada bucle.

Línea 3 realizamos un ciclo for para que la variable letra vaya asumiendo cada uno de los caracteres de la palabra “cocodrilo”.

Línea 4 con un condicional controlaremos que la variable letra pueda ser igual a “a” o “e” o “i” o “o” o “u”.

Línea 5 cuando cumpla la condición anterior que imprima en consola el valor de la variable letra.

Línea 6 si no es así que imprima por consola “No es una vocal”.

Este será el resultado del programa cuando lo ejecutemos.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData
/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Des
ktop/Ejer Python/Capitulo3.py"
No es una vocal
o
No es una vocal
o
No es una vocal
No es una vocal
No es una vocal
i
No es una vocal
o
PS C:\Users\pmver\Desktop\Ejer Python>

```

```

1  #   Mini ejemplo, iterar sobre una lista
2  numeros = [0,1,2,3,4,5]
3  print("Lista original de numeros: ")
4  print(numeros)
5
6  ✓ for num in numeros:
7      |     print(num)
8      |     numeros[num] += 10
9
10 print("Lista modificada de numeros: ")
11 print(numeros)

```

Partiendo de una lista de números con los valores del 0 hasta el 5.

En la línea 4 imprimimos por consola los valores de la lista.

En la línea 6 en un bucle que realiza un recorrido por toda la lista, en cada bucle muestra el valor de un elemento, a continuación el valor de dicho elemento lo incrementamos en 10.

En la línea 11 muestra el contenido de la lista números con sus nuevos valores.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData
/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Des
ktop/Ejer Python/Capitulo3.py"

```

Lista original de numeros:

```
[0, 1, 2, 3, 4, 5]
```

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

```
5
```

Lista modificada de numeros:

```
[10, 11, 12, 13, 14, 15]
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```

1  #   Bucles While
2  i = 0
3  ✓ while(i < 10):
4      |     print(i)
5      |     i += 1

```

El bucle while que lo podemos traducir como mientras.

La sintaxis:

while (condición):

En la línea 2 asignamos a la variable i el valor de 0.

En la línea 3 le decimos que mientras el valor de i sea menos de 10 que el bucle continúe.

En la línea 4 hacemos que el valor de i se muestre por consola.

En la línea 5 hacemos que la variable i que hace funciones de contador su valor se incremente en 1, si omitimos esta línea la variable i no cambiaría de valor y se produciría un bucle infinito.

Cuando ejecutemos este será el resultado en consola:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
```

```
0
1
2
3
4
5
6
7
8
9
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```
1  # Encontrar la primera letra a
2  letraEncontrada = False
3  letra = "a"
4  frase = "En este momento estoy buscando la letra a"
5  pos = 0
6
7  while(not(letraEncontrada)):
8      if(frase[pos] == "a"):
9          letraEncontrada = True
10         print(f"Encontrada la letra {letra} en la posición {pos}")
11     else:
12         pos += 1
```

En la línea 2 a la variable letraEncontrada de tipo booleana le asignamos el valor de False.

En la línea 3 a la variable letra le asignamos el valor "a".

En la línea 4 a la variable frase le asignamos el siguiente texto “En este momento estoy buscando la letra a”.

En la línea 5 le asignamos a la variable pos el valor de 0.

En la línea 7 le estamos diciendo mientras la variable letraEncontrada que tiene el valor false le agregamos el operador not, esta se convierte en verdadera, así que el bucle continúa.

En la línea 8 si en la variable frase en la posición (este en cada bucle irá incrementando en 1) es igual == a “a”.

En la línea 9 si la condición anterior se cumple la variable letraEncontrada de tipo booleano pasará a ser True.

En la línea 10 mostramos por consola en texto con formato.

El texto con formato antes de las comillas se pone una f, después de las comillas se escribe el texto y entre llaves pondremos las variables, en este caso las variables numéricas no hace falta tener que reconvertirlas con str.

En la línea 11 Si la condición de la línea 8 no se cumple, significa que no encuentra la letra a en este caso la variable pos que su función es de contador se incrementa en 1.

Cuando lo ejecutemos este será el resultado en consola:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData
/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Des
ktop/Ejer Python/Capitulo3.py"
Encontrada la letra a en la posición 26
PS C:\Users\pmver\Desktop\Ejer Python> 
```

break → Es una sentencia que permite parar un bucle por completo en cuanto se da o deja de darse una condición.

continue → El uso de continue al igual que el ya visto break, nos permite modificar el comportamiento de los bucles while y for.

Concretamente, continue se salta todo el código restante en la iteración actual y vuelve al principio en el caso de que aún queden iteraciones por completar.

pass → La declaración de pass se utiliza como marcador de posición para el código futuro. Cuando se ejecuta la declaración pass, no sucede nada, pero evita obtener un error cuando no se permite el código vacío.

```

1  # Break, continue, pass
2  frase = "No sé cuando encontraré la letra a"
3  letra = "a"
4
5  # Break
6  for x in frase:
7      if(x == "a"):
8          print(f"Encontrada en la posición: {frase.index(x)}")
9          break

```

En la línea 2 asignamos a la variable frase el valor de "No sé cuando encontraré la letra a".

En la línea 3 a la variable letra le asignamos el valor de "a".

En la línea 6 realizamos un bucle for para que la variable x si le vaya asignando carácter a carácter hasta completar el contenido de la variable frase.

En la línea 7 comparamos si la variable x es igual a "a".

Línea 8 en el caso que la condición anterior se cumpla imprimirá por consola la posición donde ha encontrado la letra a.

Línea 9 y además se interrumpe el bucle saliendo de él, sin tener en cuenta si ha terminado de realizar todas las iteraciones.

Si ejecutamos el programa este será el resultado en consola:

PROBLEMAS	SALIDA	CONSOLA DE DEPURACIÓN	TERMINAL
<pre> PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData /Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Des ktop/Ejer Python/Capitulo3.py" Encontrada en la posición: 8 PS C:\Users\pmver\Desktop\Ejer Python> </pre>			

```

1  # Continue
2  frase = "Hola Ana"
3  count = 0
4  for x in frase:
5      if(x == "a"):
6          count = count + 1
7          continue
8      print("No hemos encontrado nada")
9  print(count)

```

Al encontrar la sentencia continue no termina la iteración actual del bucle para seguir con el siguiente, el resto de la iteración que hay después de continue es ignorado.

En la línea 2 asignamos a la variable frase el valor de "Hola Ana"

En la línea 3 asignamos a la variable count el valor de 0, la utilizaremos como contador.

En la línea 4 en un ciclo for la variable x realizará un recorrido de carácter a carácter sobre el contenido de la variable frase.

En la línea 5 Si la variable x es igual "a"

En la línea 6 si se cumple la condición anterior la variable count se incrementará en 1.

En la línea 7 y además al ejecutar la sentencia continue continuará con la siguiente iteración ignorando el código que se encuentra por debajo del ciclo for.

En la línea 8 muestra un mensaje por consola "No hemos encontrado nada".

En la línea 9 una vez finalizadas todas las iteraciones del bucle for imprimirá por consola el número de veces que a encontrado la letra a en minúsculas.

Una vez ejecutemos el código este será el resultado en la consola:

```
PROBLEMAS    SALIDA    CONSOLA DE DEPURACIÓN    TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData
/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Des
ktop/Ejer Python/Capitulo3.py"
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
2
PS C:\Users\pmver\Desktop\Ejer Python> 
```

```
1  # Pass
2  frase = "frase corta"
3  for x in frase:
4      if(x == "a"):
5          pass
6      print("Seguimos con la iteración")
```

En la línea 2 asignamos a la variable frase el valor "frase corta"

En la línea 3 en un ciclo for la variable x irá asumiendo el valor de cada carácter de la variable frase.

En la línea 4 si x es igual "a".

En la línea 5 Si en la línea anterior se cumple con la condición al ejecutar pass, sencillamente no hará nada.

En la línea 6 mostrará un mensaje por consola "Seguimos con la iteración").

Este será el resultado en la consola una vez ejecutemos el programa:

PROBLEMAS

SALIDA

CONSOLA DE DEPURACIÓN

TERMINAL

```
ktop/Ejer Python/Capitulo3.py"
```

```
Seguimos con la iteración
```

```
Seguimos con la iteración
```

```
Seguimos con la iteración
```

```
Seguimos con la iteración
```

```
Seguimos con la iteración
```

```
Seguimos con la iteración
```

```
Seguimos con la iteración
```

```
Seguimos con la iteración
```

```
Seguimos con la iteración
```

```
Seguimos con la iteración
```

```
Seguimos con la iteración
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

No observamos ninguna diferencia si la variable x tiene el valor de "a" u otro valor.

```
1  # Else
2  frase = "Quiero contar todas las a de esta frase"
3  count = 0
4
5  for x in frase:
6      if(x == "a"):
7          count = count + 1
8          continue
9      print("No hemos encontrado nada")
10 else:
11     print(f"Hemos encontrado {count} veces la letra a")
```

Podemos utilizar la sentencia else con un ciclo for, cuando terminan todas las iteraciones del bucle a continuación se ejecuta las instrucciones que hay en el else.

Si utilizando un break rompemos las iteraciones del bucle las instrucciones else no se ejecutarán.

Línea 2 asignamos a la variable frase el siguiente valor “Quiero contar todas las a de esta frase”.

Línea 3 asignamos a la variable count el valor de 0 (Realizará funciones de contador).

Línea 5 En el ciclo for la variable x irá asumiendo carácter a carácter el contenido de la variable frase.

Línea 6 comparamos si x tiene el valor de “a”.

Línea 7 si la condición anterior se cumple la variable contador se incrementará en 1.

Línea 8 Y además al encontrar la sentencia continue saltamos a la siguiente iteración ignorando el resto del código que se encuentra en la parte inferior del bucle.

Línea 9 muestra en consola el siguiente mensaje “No hemos encontrado nada”

Línea 10 con la sentencia else cuando haya terminado todas las iteraciones del bucle se ejecutarán las instrucciones que contenga.

Línea 11 mostramos en consola con formato las veces que se ha encontrado la letra a.

Si ejecutamos este será el resultado en la consola:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
No hemos encontrado nada
Hemos encontrado 6 veces la letra a
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Ejercicio práctico:

El usuario debe adivinar un número entre 0 y 10.

El programa deberá pedir que el usuario introduzca un número y debe decir si ha acertado, si el número es menor o mayor que el que ha introducido.

```
1  def pedirYComprobar(numero):
2      num = int(input("Introduce el número: "))
3      if(num == numero):
4          print("Has acertado")
5          return True
6      elif(numero > num):
7          print("Casi! Prueba con un número más grande")
8          return False
9      elif(numero < num):
10         print("Casi! Prueba con un número pequeño")
11         return False
12
13     while(True):
14         if(pedirYComprobar(7)):
15             break
16     else:
17         print("Cuando hay un break, no se ejecuta el else")
18
```

En la línea 14 al hacer la comparación también se ejecuta la función, cuando acierta el número la sentencia break hace que salgamos del bucle.

```
20  while(not(pedirYComprobar(7))):
21      pass
```

En la línea 20 con este segundo ejemplo mientras no acertemos el número no saldremos del bucle, pero al encontrar la sentencia pass no realizará nada.

Cuando deje de cumplir la condición del while se terminarán las iteraciones.

6.- Listas en Python

```
# Capítulos 6: Listas
# Apartado 1: Repaso de conceptos básicos
input("Presiona 'enter' para avanzar al apartado -> 1. Repaso de Conceptos básicos: ")
lista = [1, 2, 3, 4, 5] # Declaración de una lista
elemento = lista[0] # Acceder al elemento de la primera posición de la lista
longitud = len(lista) # Número de elementos o longitud de la lista

print("Lista de números: ")
print(lista)
print(f"Primer elemento: {elemento}")
print(f"Longitud de la lista: {longitud}")
print("Iterar sobre los elementos de una lista: ")
for num in lista: # Iterar sobre los elementos de una lista
    print(num)

# Apartado 2: Indexado y sublistas
input("Presiona 'enter' para avanzar al apartado -> 2. Indexado y sublistas: ")
lista = ["Dale", "un", "buen", "like", "al", "vídeo"]
# En Python, las listas pueden tener índices negativos. Empiezan a contar des del final.
# El último elemento está en la posición -1 y el penúltimo en la -2.

ultimoElemento = lista[-1] # Accediendo al último valor de la lista
penultimo = lista[-2] # Accediendo al penúltimo valor de la lista

# Para acceder a un subconjunto de elementos de una lista, utilizamos la nomenclatura:
# lista[indice_inicio:indice_final]. Esto devuelve una lista con los elementos cuyos
# índices se encuentren entre los utilizamos en los brackets.

sublista = lista[2:4] # sublista = ["buen", "like"]
sublista2 = lista[:3] # Sublista con los elementos de las posiciones 0, 1 y 2. Posición 3 no
incluída.
sublista3 = lista[2:] # Sublista que empieza en la posición 2, hasta el final de la lista.

sublista4 = lista[-4: -1] # Sublista usando índices negativos

print("Lista original: ")
print(lista)
print("Sublista del elemento dos hasta el 3 : ")
print(sublista)
print("Sublista des del inicio hasta el elemento 2: ")
print(sublista2)
print("Sublista des del elemento 2 hasta el final: ")
print(sublista3)
print("Sublista usando índices negativos: ")
print(sublista4)
```

```

# Apartado 3: Comprobar si una lista contiene o no un elemento
input("Presiona 'enter' para avanzar al apartado -> 3. Comprobar si un elemento pertenece o no a la lista: ")
if "like" in lista:
    print("El elemento 'like' pertenece a la lista")

if "manzana" not in lista:
    print("El elemento 'manzana' no está en la lista")

# Apartado 4: Modificar elementos en una lista
input("Presiona 'enter' para avanzar al apartado -> 4. Modificar elementos de una lista: ")

lenguajes = ["C", "Java", "Python", "JavaScript", "Kotlin", "Ruby"]

print("Lista inicial: ")
print(lenguajes)

lenguajes[0] = "C++" # Modificar el valor del primer elemento de la lista

print("Lista con el primer elemento modificado: ")
print(lenguajes)

lenguajes[2:4] = ["Anaconda", "TypeScript"] # Modificar varios elementos a la vez

print("Lista tras modificar varios elementos a la vez: ")
print(lenguajes)

lenguajes[4:5] = ["C#", "Visual Basic"]
print("Lista tras añadir más de un elemento en una posición")
print(lenguajes)

# Apartado 5: Métodos de las listas: Añadir elementos
# En Python podemos utilizar métodos con las listas.
# Para ejecutar estos métodos: variableDeTipoLista.metodo()
input("Presiona 'enter' para avanzar al apartado -> 5. Métodos de las listas: Añadir Elementos")

lenguajes = ["C", "Java", "Python", "JavaScript", "Kotlin", "Ruby"]

print("Lista original: ")
print(lenguajes)

print("Insertar un elemento en una posición determinada, sin borrar ningún otro elemento: ")
lenguajes.insert(1, "C++") # Insertamos el elemento "C++" en la posición 1. El resto de elementos se desplazan hacia la derecha
print(lenguajes)

```

```
print("Añadir un elemento al final de la lista: ")
lenguajes.append("Scala") # Esta función añade un elemento al final de la lista
print("Lenguajes")
```

```
print("Añadir una lista a otra: ")
sublista = ["TypeScript", "Matlab", "R"]
print("Lista 1: ")
print(lenguajes)
print("Lista 2: ")
print(sublista)
lenguajes.extend(sublista)
print("Lista combinada: ")
print(lenguajes)
```

```
# Apartado 6: Métodos de las listas: Eliminar elementos
input("Presiona 'enter' para avanzar al apartado -> 6. Métodos de las listas: Eliminar Elementos")
frutas = ["plátano", "kiwi", "papaya", "melocotón", "cereza"]
```

```
print("Lista original: ")
print(frutas)
```

```
print("Eliminar el último elemento de la lista: ")
elem = frutas.pop() # La función pop si argumento elimina y devuelve el último elemento.
```

```
print(f"El elemento {elem} ha sido eliminado de la lista de frutas")
print(frutas)
```

```
print("Eliminar el elemento número X de la lista: ")
elem = frutas.pop(2) # Se le puede pasar el índice del elemento a eliminar como argumento.
print(f"El elemento {elem} ha sido eliminado de la lista de frutas")
print(frutas)
```

```
print("Eliminar un elemento de la lista por su valor: ")
frutas.remove("plátano") # La función remove eliminar un elemento por su valor. También lo devuelve
print("Lista con el elemento eliminado: ")
print(frutas)
```

```
print("Eliminar el elemento número X de la lista utilizando 'del' : ")
del frutas[0] # Eliminar el elemento de la posición 0 mediante la keyword 'del'
print(frutas)
```

```
print("Vaciar una lista: ")
frutas.clear() # Vaciar todos los elementos de una lista.
print("frutas")
```

Apartado 7: Ejercicio

input("Presiona 'enter' para avanzar al apartado -> 7. Ejercicio")

ENUNCIADO: Tenemos un texto donde queremos encontrar palabras clave.

Las palabras clave pueden ser hasta 5 y deberemos pedirselas al usuario

y guardarlas en una lista.

Si el usuario quiere poner menos de 5 palabras clave, deberá escribir

"fin" para terminar de introducir datos. Si el usuario introduce

números o nada, deberemos eliminarlos de la lista antes de la búsqueda.

En otra lista, deberemos guardar el número de veces que aparece cada

palabra clave en nuestro texto. Por ejemplo, si las palabras clave son

ordenador y portátil y aparecen 5 y 7 veces respectivamente, nuestras listas

deberían ser así:

- keywords = ["ordenador", "portátil"]

- ocurrencias = [5, 7]

Pista: Podemos pasar de una cadena de texto a una lista de palabras mediante

el método texto.split(). Por ejemplo:

texto = "hola que tal"

print(texto.split())

SOLUCIÓN:

texto = """Seguramente hayas notado que tu productividad ha bajado desde que trabajas desde casa.

Es muy importante que logremos teletrabajar efectivamente y de manera autorregulada.

Esto se traduce en finalizar antes nuestras tareas y evitar jornadas laborales interminables.

El primer consejo y uno de los más importantes ya te lo he dado en el apartado anterior.

Tenemos que construir un espacio de trabajo en el que nos sintamos cómodos y dispongamos de todas las herramientas necesarias para teletrabajar.

En la medida de lo posible, es importante teletrabajar siempre en el mismo lugar.

De esta forma, nuestro cerebro asocia el sitio con la acción de trabajar y nos hará estar más focalizados en nuestras tareas."""

keywords = []

ocurrencias = []

Obtener palabras clave

keywords = []

ocurrencias = []

for x in range(5):

keyword = input("Introduce una palabra clave. Si no quieres escribe 'fin': ")

if keyword == "fin":

break

```

else:
    keywords.append(keyword)

print("Palabras clave introducidas: ")
print(keywords)

posicion = 0

while(True):
    if posicion >= len(keywords):
        break
    if keywords[posicion] == "":
        keywords.pop(posicion)
    elif keywords[posicion].isnumeric():
        keywords.pop(posicion)
    else:
        posicion += 1

print("Palabras clave corregidas: ")
print(keywords)

texto = texto.replace('.', '').replace(',', '').split()

for x in range(len(keywords)):
    ocurrencias.append(0)

print("Lista de ocurrencias inicializada: ")
print(ocurrencias)

for palabra in texto:
    for keyword in keywords:
        if keyword == palabra:
            posicion = keywords.index(keyword)
            ocurrencias[posicion] += 1
            break

print("Resultado final: ")
print(keywords)
print(ocurrencias)

```

7.- Ejercicio

texto = """Seguramente hayas notado que tu productividad ha bajado desde que trabajas desde casa.
Es muy importante que logremos teletrabajar efectivamente y de manera autorregulada. Esto se traduce en finalizar antes nuestras tareas y evitar jornadas laborales interminables. El primer consejo y uno de los más importantes ya te lo he dado en el apartado anterior. Tenemos que construir un espacio de trabajo en el que nos sintamos cómodos y dispongamos de todas las herramientas necesarias para teletrabajar.
En la medida de lo posible, es importante teletrabajar siempre en el mismo lugar. De esta forma, nuestro cerebro asocia el sitio con la acción de trabajar y nos hará estar más focalizados en nuestras tareas."""

```
keywords = []      # Definimos una lista llamada keywords vacía.
ocurrencias = []   # Definimos una lista llamada ocurrencias vacía.

# Obtener palabras clave
for x in range(5): # Un bucle de 5 iteraciones
    keyword = input("Introduce una palabra clave o 'fin' si ya no quieres poner más: ")
    if keyword == "fin": # Si contestamos con fin se rompen las iteraciones del bucle.
        break
    else:
        keywords.append(keyword) # De lo contrario lo agregamos a la lista.

print("Palabras clave introducidas")

# Eliminar números o strings vacíos
for keyword in keywords:
    if keyword == "" or keyword.isnumeric(): # Si hemos introducido una cadena vacía o un
        keywords.remove(keyword)           # número estos se eliminarán.

# Inicializamos la lista de ocurrencias y preparamos el texto

texto = texto.replace('.', '').replace(',', '').split() # split() se utiliza para dividir una cadena en
for x in range(len(keywords)):                          # una lista, por defecto si no detallamos nada
    ocurrencias.append(0)                                # la separa por espacios.

for palabra in texto: # La variable palabra irá asumiendo todos los valores de la lista texto.
    for keyword in keywords: # Comparará con cada palabra que hemos introducido.
        if palabra == keyword: # Compara si a encontrado alguna de estas palabras.
            ocurrencias[keywords.index(keyword)] += 1 # En la lista ocurrencias en su correspon-
            break                                     # diente índice incrementa un +1

print(keywords) # Muestra la lista keyword con las palabras que introdujimos.
```

```
print(ocurrencias) # Muestra la lista ocurrencias con el número de veces que ha encontrado
Este será el resultado: # la palabra.
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
11/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
Introduce una palabra clave o 'fin' si ya no quieres poner más: casa
Introduce una palabra clave o 'fin' si ya no quieres poner más: finalizar
Introduce una palabra clave o 'fin' si ya no quieres poner más: trabajo
Introduce una palabra clave o 'fin' si ya no quieres poner más: siempre
Introduce una palabra clave o 'fin' si ya no quieres poner más: fin
Palabras clave introducidas
['casa', 'finalizar', 'trabajo', 'siempre']
[1, 1, 1, 1]
PS C:\Users\pmver\Desktop\Ejer Python> █
```

8.- ¿Qué es una CADENA en Python?

import os # Estas librería no permite utilizar instrucciones del sistema operativo como cls.

Capítulos 8: Strings

Apartado 1: Repaso de conceptos básicos

Distintas formas de declarar Strings

input("Presiona 'enter' para avanzar al apartado -> 1. Repaso de Conceptos básicos: ")

os.system("cls") # Gracias a la librería os.

texto = "¿Hola qué tal?"

print("Texto simple: ")

print(texto)

Si queremos mostrar comillas dobles en nuestro texto: ""

deberemos envolver el texto en comillas simples ->

comillasSimples = 'el personaje afirmó "me convertiré en rey" rotundamente'

print("Texto que contiene comillas dobles: ")

print(comillasSimples)

Si queremos mostrar comillas simples en nuestro texto: ""

deberemos envolver el texto en comillas dobles ->

comillasDobles = "el personaje afirmó 'me convertiré en rey' rotundamente"

print("Texto que contiene comillas simples: ")

print(comillasDobles)

Para crear un texto formateado (saltos de línea, sangrías)

deberemos envolver el texto en TRES comillas dobles ->

textoFormateado = """

===== TEXTO FORMATEADO =====

Podemos poner:

- Saltos de línea.

- Tabulaciones.

=====

"""

print("Texto formateado: ")

print(textoFormateado)

Conversión de datos numéricos a Strings

numeroEntero = 7

numeroDecimal = 3.14

print("Tipo de dato antes de la conversión: ")

print(type(numeroEntero))

print(type(numeroDecimal))

```
strNumeroEntero = str(numeroEntero)
strNumeroDecimal = str(numeroDecimal)
```

```
print("Tipo de dato después de la conversión: ")
print(type(strNumeroEntero))
print(type(strNumeroDecimal))
```

Conversión de booleanos a Strings

```
print("También podemos convertir booleanos por ejemplo: ")
print(type(str(True)))
```

Número de caracteres en un String:

```
texto = "123456789"
longitudTexto = len(texto)
print("La variable 'texto' contiene: " + str(longitudTexto) + " caracteres")
```

Concatenar Strings:

```
texto1 = "012345"
texto2 = "6789"
textoConcatenado = "Números del 0 al 9" + texto1 + texto2
print(textoConcatenado)
```

Apartado 2: Los strings son listas

En este apartado descubriremos que realmente los strings son Arrays

o vectores (similar a las listas).

```
input("Presiona 'enter' para avanzar al apartado -> 2. Los Strings son listas: ")
os.system("cls")
texto = "0123456789"
print("Texto original:\n" + texto)
```

Acceder al primer carácter de un String

```
primerCaracter = texto[0]
print("Primer caracter: ")
print(primerCaracter)
```

Acceder a un subgrupo de caracteres de un String

```
subString = texto[1:6]
print("Substring: ")
print(subString)
```

Iterar sobre un String:

```
print("Iterar sobre un string: ")
texto = "0123456789"
for num in texto:
    print("Valor actual: " + num)
```

```
# EJERCICIO (mini): Tenemos un string únicamente compuesto por números enteros.  
# Debemos sumar su valor uno a uno y mostrar por pantalla la suma  
# total de todos los números.
```

```
valores = "984823185"  
print("Ejercicio: Sumar todos los números del string -> " + valores)
```

```
# Solución:
```

```
total = 0  
for valor in valores:  
    total += int(valor)  
  
print(f"El total es: {total}")
```

```
# Apartado 3: Format Strings  
# Nos permiten generar strings combinando texto con  
# otras variables de forma sencilla y legible.
```

```
input("Presiona 'enter' para avanzar al apartado -> 3. Format Strings: ")  
os.system("cls")
```

```
cantidad = 5  
precio = 15.80
```

```
print("Forma cutre: ")  
print("Has comprado " + str(cantidad) + " unidades. Cada unidad vale " + str(precio) + "€. En  
total son: " + str(cantidad*precio) + "€")
```

```
print("Forma avanzada: ")  
formatString = "Has comprado {} unidades. Cada una vale {}€. En total son  
{€}".format(cantidad, precio, cantidad*precio)  
print(formatString)
```

```
print("God mode: ")  
print(f"Has comprado {cantidad} unidades. Cada una vale {precio}€. En total son  
{precio*cantidad}€")
```

9.- ¿Qué son los MÉTODOS de CADENAS en Python?

```
1  import os
2  import logging
3  # Capítulos 9: Métodos de Strings
4  # Apartado 1: Métodos find, index y count
5
6  # Los métodos find e index sirven para encontrar conjuntos
7  # de caracteres dentro de un string (por ejemplo, una
8  # palabra dentro de una frase)
9
10 input("Presiona 'enter' para avanzar al apartado -> 1. Métodos find, index y count: ")
11 os.system("cls")
12
13 # Vamos a usar los métodos para comprobar que una dirección de correo
14 # contiene una '@' y únicamente una.
15
16 mail = "correo@gmail.com"
17
18 # En caso de encontrar el substring, ambos métodos hacen lo mismo.
19 posFind = mail.find("@")
20 posIndex = mail.index("@")
21 print(f"La posición encontrada con find es: {posFind}.\nLa posición encontrada con index es: {posIndex}")
```

Línea 2 Import logging → El registro es un medio para rastrear eventos que ocurren cuando se ejecuta algún software. El registro es importante para el desarrollo, la depuración y la ejecución de software. Si no tienes ningún registro y su programa falla, hay muy pocas posibilidades de que detecte la causa del problema.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

La posición encontrada con find es: 6.

La posición encontrada con index es: 6

PS C:\Users\pmver\Desktop\Ejer Python> █

```
1  import os
2  import logging
3  # Capítulos 9: Métodos de Strings
4  # Apartado 1: Métodos find, index y count
5
6  # En caso de no encontrar el substring, find devuelve -1 pero index lanza un error.
7  mail = "correo@gmail.com"
8  posFind = mail.find("")
9  print(f"La posición encontrada es: {posFind}")
10 try:
11     posIndex = mail.index("")
12 except BaseException:
13     logging.exception("Ha ocurrido el siguiente error: ")
14
15 # Para comprobar que únicamente aparece una vez, usamos el método count:
16 apariciones = mail.count('@')
17 print(f"El caracter '@' aparece {apariciones} vez")
```

Con la sentencia try: el código que escribimos a continuación si este diera un error en lugar de detenerse el programa, con la sentencia except BaseException registramos el error pero no se detiene la ejecución del programa.

Por este motivo se ejecutan las líneas que vienen después de except línea 16 y 17.

```
La posición encontrada es: -1
ERROR:root:Ha ocurrido el siguiente error:
Traceback (most recent call last):
  File "c:\Users\pmver\Desktop\Ejer Python\Capitulo3.py", line 11, in <module>
    posIndex = mail.index("*")
                ^^^^^^^^^^^^^^^^^
ValueError: substring not found
El caracter '@' aparece 1 vez
PS C:\Users\pmver\Desktop\Ejer Python> 
```

```
1  # Apartado 2: Métodos split, replace y join:
2  # El método split sirve para dividir un string mediante un delimitador
3  # y convertir cada división en un elemento de una lista.
4  # El método replace sirve para reemplazar un substring por otro.
5  # El método join sirve para unir una lista de strings en un único string
6  # mediante un caracter determinado.
7  import os
8  input("Presiona 'enter' para avanzar al apartado -> 2. Métodos split, replace y join: ")
9  os.system("cls")
10
11 compra = "chocolate, pan, agua, plátanos, verduras, pipas, orégano"
12 print(compra)
13
14 # ¿Cómo podemos saber cuántas cosas hay que comprar?
15 listaCompra = compra.split(',')
16 numeroElementos = len(listaCompra)
17 print(listaCompra)
18 print(numeroElementos)
19
20 # Uso del método replace para separar los elementos mediante '-' en vez de ','.
21 print("Modificamos el separador de los elementos del string 'compra': ")
22 compra = compra.replace(',', '-', ' ')
23 print(compra)
24
25 # Uso del método join para convertir una lista en un string
26 compra = "/".join(listaCompra)
27 print(compra)
```

En la ejecución del programa podrás observar el resultado de los métodos.

```
chocolate, pan, agua, plátanos, verduras, pipas, orégano
['chocolate', 'pan', 'agua', 'plátanos', 'verduras', 'pipas', 'orégano']
7
Modificamos el separador de los elementos del string 'compra':
chocolate - pan - agua - plátanos - verduras - pipas - orégano
chocolate/pan/agua/plátanos/verduras/pipas/orégano
PS C:\Users\pmver\Desktop\Ejer Python> 
```

Ejercicio:

Tenemos unas descripciones de algunos productos. En ellas se incluyen el precio de cada producto. Debemos encontrar el precio en € y mostrarlos en pantalla. El precio puede tener 0, o 2 decimales y siempre seguido del símbolo '€', por ejemplo:

- 9.99€
- 10€
- 10.5€

Bonus:

Deberemos modificar las descripciones para que el precio se muestre en dólares. La conversión es: 1€ = 1.21\$. No hace falta modificar la variable original de la descripción, podemos retornar una copia con el precio convertido.

Solución:

```
1 des1 = "Este bolso de cuero de la marca: Miguel Cors tiene un precio de 199.99€. Es una oferta especial."
2 des2 = "Las botas de la marca: Nique valen 89€. Nunca han estado tan rebajadas."
3 des3 = "¡Tenemos el bambú en oferta! Cómpralo ahora por 1.2€ el kg ¡No la dejes pasar!"
4
5 def findPriceAndReplace(txt):
6     txtList = txt.split()
7     pos = -1
8     indx = -1
9     conversion = 1.21
10    for palabra in txtList:
11        pos = palabra.find("€")
12        if pos != -1:
13            indx = txtList.index(palabra)
14            break
15    precio = txtList[indx]
16    precio = precio.split('€')[0]
17    txtList[indx] = str(float(precio)*conversion) + '$'
18    newDesc = " ".join(txtList)
19    return precio, newDesc
20 precio, newDesc = findPriceAndReplace(des1)
21 print(precio)
22 print(newDesc)
23 precio, newDesc = findPriceAndReplace(des2)
24 print(precio)
25 print(newDesc)
26 precio, newDesc = findPriceAndReplace(des3)
27 print(precio)
28 print(newDesc)
```

Línea 16 con la sentencia Split, del valor que tiene precio que en el primer ejemplo es 199.99€ elimina el euro para quedarse en 199.99.

El parámetro [0] evita que al respuesta sea:

['199.99', ''] en lugar de 199.99

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe
"c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
199.99
Este bolso de cuero de la marca: Miguel Cors tiene un precio de 241.9879$ Es una oferta especial.
89
Las botas de la marca: Nique valen 107.69$ Nunca han estado tan rebajadas.
1.2
¡Tenemos el bambú en oferta! Cómpralo ahora por 1.452$ el kg ¡No la dejes pasar!
PS C:\Users\pmver\Desktop\Ejer Python> □
```

```

1  # Apartado 3: Otros métodos útiles
2  # En este apartado veremos algunos métodos que pueden ser de
3  # utilidad para manipular strings.
4  import os
5
6  input("Presiona 'enter' para avanzar al apartado -> 3. Otros métodos útiles ")
7  os.system("cls")
8
9  texto = "esto Es un texto de Ejemplo"
10 print("\nMétodo startswith: comprobar si un string empieza con el valor especificado")
11 print(texto.startswith('e'))
12 print("\nMétodo upper: Retorna una copia del string en mayúsculas")
13 print(texto.upper())
14 print("\nMétodo title: Retorna una copia del string pero la primera letra de cada palabra está en mayúsculas")
15 print(texto.title())
16 print("\nMétodo capitalize: Retorna una copia del string con la primera letra en mayúsculas")
17 print(texto.capitalize())
18 print("\nMétodo rjust: Retorna una copia del string con X caracteres justificado hacia la derecha")
19 print(texto.rjust(len(texto) + 8))

```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

Método startswith: comprobar si un string empieza con el valor especificado
True

Método upper: Retorna una copia del string en mayúsculas
ESTO ES UN TEXTO DE EJEMPLO

Método title: Retorna una copia del string pero la primera letra de cada palabra está en mayúsculas
Esto Es Un Texto De Ejemplo

Método capitalize: Retorna una copia del string con la primera letra en mayúsculas
Esto es un texto de ejemplo

Método rjust: Retorna una copia del string con X caracteres justificado hacia la derecha
 esto Es un texto de Ejemplo
 PS C:\Users\pmver\Desktop\Ejer Python> █

Ejercicio:

Formatear texto. Debemos formatear el siguiente texto siguiendo una serie de normas:

- ‘#’ a principio de línea significa que es un título y deberemos convertirlo todo a mayúsculas.
- ‘##’ A principio de línea significa que es un subtítulo y deberemos poner la primera letra de cada palabra en mayúsculas.
- ‘###’ Deberemos poner únicamente la primera letra en mayúscula.
- Si la línea empieza con ¡-’ deberemos añadir sangría.

Nota: El método `splitlines()` retorna una lista de strings. Separa mediante saltos de línea.

Solución:

```

1  > texto = """
2  #este es el título principal
3
4  - Esto es una lista.
5  - De cosas que podemos hacer.
6
7  ##este es un subtítulo
8
9  Esto es una pequeña introducción.
10 - Esto es otra lista
11 - De más cosas que podemos hacer.
12 """
13
14 > def formatText(txt):
15     txtList = txt.splitlines()
16     for indx in range(len(txtList)):
17         if(txtList[indx].startswith('###')):
18             txtList[indx] = txtList[indx].replace('###', '')
19             txtList[indx] = txtList[indx].capitalize()
20         elif(txtList[indx].startswith('##')):
21             txtList[indx] = txtList[indx].replace('##', '')
22             txtList[indx] = txtList[indx].title()
23         elif(txtList[indx].startswith('#')):
24             txtList[indx] = txtList[indx].replace('#', '')
25             txtList[indx] = txtList[indx].upper()
26         elif(txtList[indx].startswith('-')):
27             txtList[indx] = txtList[indx].rjust(len(txtList[indx]) + 8)
28     return '\n'.join(txtList)
29
30     print(formatText(texto))

```

Este será el resultado:

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"

```

ESTE ES EL TÍTULO PRINCIPAL

```

    - Esto es una lista.
    - De cosas que podemos hacer.

```

Este Es Un Subtítulo

```

Esto es una pequeña introducción.
    - Esto es otra lista
    - De más cosas que podemos hacer.

```

```

PS C:\Users\pmver\Desktop\Ejer Python> █

```

10.- ¿Cómo funciona un DICCIONARIO en Python?

```
1  import os
2
3  #  Capítulos 10: DICCIONARIOS
4  #  Apartado 1: conceptos básicos
5
6  input("Presiona 'enter' para avanzar al apartado -> 1. Conceptos básicos: ")
7  os.system("cls")
8
9  #  Declaración de diccionarios:
10 #  Los diccionarios son conjuntos de elementos con la estructura clave: valor.
11 #  Cada elemento de un diccionario contiene una clave y un valor asociado a esta.
12 #  Las claves pueden ser Strings o int. Los valores pueden ser cualquier tipo de dato.
13
14 print("Diccionarios: ")
15 alejandro = {"Nombre": "Alejandro", "Edad": 30, "Ciudad": "Barcelona",
16 |           | "Aficiones": ["videojuegos", "lectura", "cine"]}
17 print(alejandro)
18 ana = {"Nombre": "Ana", "Edad": 32, "Ciudad": "Madrid",
19 |      "Aficiones": ["golf", "python", "pesca"]}
20 print(ana)
21
22 #  Los diccionarios pueden contener otros diccionarios:
23 print("\nDiccionario de diccionarios: ")
24 trabajadores = {0: alejandro, 1: ana}
25 print(trabajadores)
26
27 #  Los diccionarios no pueden contener claves repetidas:
28 print("\nDiccionario con claves repetidas: ")
29 a = {
30 |     "año": 1998,
31 |     "año": 2003
32 | }
33 print(a)
34
35 #  Longitud de un diccionario:
36 print(f"El diccionario tiene {len(alejandro)} elementos")
```

Línea 15 se crea un diccionario con el nombre de alejandro.

Línea 17 Muestra el contenido del diccionario alejandro.

Línea 18 se crea un diccionario con el nombre de ana.

Línea 20 muestra los datos por consola del diccionario ana.

Línea 24 creamos un diccionario llamado trabajadores que contiene a su vez los diccionarios Alejandro y ana.

Línea 25 muestra por consola el contenido de los diccionarios que tiene el diccionario trabajadores.

Si ejecutas el programa este será el resultado por consola.

```
Diccionarios:
{'Nombre': 'Alejandro', 'Edad': 30, 'Ciudad': 'Barcelona', 'Aficiones': ['videojuegos', 'lectura', 'cine']}
{'Nombre': 'Ana', 'Edad': 32, 'Ciudad': 'Madrid', 'Aficiones': ['golf', 'python', 'pesca']}
```

```
Diccionario de diccionarios:
{0: {'Nombre': 'Alejandro', 'Edad': 30, 'Ciudad': 'Barcelona', 'Aficiones': ['videojuegos', 'lectura', 'cine']}, 1: {'Nombre': 'Ana', 'Edad': 32, 'Ciudad': 'Madrid', 'Aficiones': ['golf', 'python', 'pesca']}}
```

Diccionario con claves repetidas:

```
{'año': 2003}
```

El diccionario tiene 4 elementos

PS C:\Users\pmver\Desktop\Ejer Python> █

```
1  import os
2  # Apartado 2: Añadir, eliminar y modificar elementos de un diccionario
3
4  input("Presiona 'enter' para avanzar al apartado -> 2. Añadir, eliminar y modificar elementos de un diccionario: ")
5  os.system("cls")
6  print("\nDiccionario original: ")
7  alejandro = {"Nombre": "Alejandro", "Edad": 30, "Ciudad": "Barcelona",
8  |           | "Aficiones": ["videojuegos", "lectura", "cine"]}
9  print(alejandro)
10
11 print("\nAñadir elemento: ")
12 alejandro["cargo"] = "CEO"
13 print(alejandro)
14 print("Añadir un elemento mediante el método 'update': ")
15 alejandro.update({"sueldo": 2000})
16 print(alejandro)
17
18 print("\nEliminar un elemento ")
19 del alejandro["cargo"]
20 print(alejandro)
21 print("Eliminar un elemento mediante el método 'pop': ")
22 value = alejandro.pop("sueldo")
23 print(alejandro)
24 print(value)
25 |
26 print("\nAcceder a un elemento: ")
27 edad = alejandro["Edad"]
28 print(f" Tiene {edad} años")
29 print("Acceder un elemento mediante el método 'get': ")
30 alejandro.update({"sueldo": 2000})
31 print(alejandro)
```

Línea 11 Añadir un elemento → Alejandro["cargo"] = "CEO".

Línea 19 Eliminar un elemento → del Alejandro["cargo"]

Línea 22 Eliminar un elemento mediante el método 'pop' → value = Alejandro.pop("sueldo")

La variable value retorna el elemento eliminado.

Línea 27 Acceder a un elemento → edad =alejandro["Edad"]

Línea 30 Acceder a un elemento mediante el método 'get' → alejandro.update({"sueldo": 2000})

Diccionario original:
 {'Nombre': 'Alejandro', 'Edad': 30, 'Ciudad': 'Barcelona', 'Aficiones': ['videojuegos', 'lectura', 'cine']
 }

Añadir elemento:
 {'Nombre': 'Alejandro', 'Edad': 30, 'Ciudad': 'Barcelona', 'Aficiones': ['videojuegos', 'lectura', 'cine'],
 'cargo': 'CEO'}

Añadir un elemento mediante el método 'update':

{'Nombre': 'Alejandro', 'Edad': 30, 'Ciudad': 'Barcelona', 'Aficiones': ['videojuegos', 'lectura', 'cine'],
 'cargo': 'CEO', 'sueldo': 2000}

Eliminar un elemento

{'Nombre': 'Alejandro', 'Edad': 30, 'Ciudad': 'Barcelona', 'Aficiones': ['videojuegos', 'lectura', 'cine'],
 'sueldo': 2000}

Eliminar un elemento mediante el método 'pop':

{'Nombre': 'Alejandro', 'Edad': 30, 'Ciudad': 'Barcelona', 'Aficiones': ['videojuegos', 'lectura', 'cine']
 }
 2000

Acceder a un elemento:

Tiene 30 años

Acceder un elemento mediante el método 'get':

{'Nombre': 'Alejandro', 'Edad': 30, 'Ciudad': 'Barcelona', 'Aficiones': ['videojuegos', 'lectura', 'cine'],
 'sueldo': 2000}

PS C:\Users\pmver\Desktop\Ejer Python> █

```

1  import os
2
3  # Apartado 3: Listas de claves y valores
4  # Podemos obtener una lista tanto de las claves como de los valores
5  # que conforman un diccionario.
6
7  input("Presiona 'enter' para avanzar al apartado -> 3. Listas de claves y valores: ")
8  os.system("cls")
9
10 print("\nLista de claves: ")
11 alejandro = {"Nombre": "Alejandro", "Edad": 30, "Ciudad": "Barcelona",
12 |           "Aficiones": ["videojuegos", "lectura", "cine"]}
13 claves = alejandro.keys()
14 print(claves)
15
16 print("\nLista de valores: ")
17 valores = alejandro.values()
18 print(valores)
19
20 # Podemos iterar en el mismo bucle sobre ambas:
21
22 for clave, valor in zip(claves, valores):
23 |     print(f"{clave} --> {valor}")
24
25
26 for clave, valor in alejandro.items():
27 |     print(f"{clave} == {valor}")

```

Línea 13 Le asignamos a la variable clave todas las claves del diccionario alejandro.

Línea 14 las mostramos por consola.

Línea 17 Le asignamos a la variable valores los valores del diccionario alejandro.

Línea 18 las mostramos por consola.

Línea 22 Podemos iterar en el mismo bucle ejemplos líneas 22 y 26.

Para el ejemplo de la línea 22 previamente necesita las líneas seleccionadas, crear la lista claves y valores.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
Lista de claves:
dict_keys(['Nombre', 'Edad', 'Ciudad', 'Aficiones'])

Lista de valores:
dict_values(['Alejandro', 30, 'Barcelona', ['videojuegos', 'lectura', 'cine']])
Nombre --> Alejandro
Edad --> 30
Ciudad --> Barcelona
Aficiones --> ['videojuegos', 'lectura', 'cine']
Nombre == Alejandro
Edad == 30
Ciudad == Barcelona
Aficiones == ['videojuegos', 'lectura', 'cine']
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Ejercicio:

Buscador de productos. Tenemos varios productos, el usuario mediante el nombre del producto puede consultar su precio y sus unidades en stock.

Solución:

```
1  import os
2
3  pantalones = {"nombre": "pantalón", "precio": 39.99, "stock": 10}
4
5  chaquetas = {"nombre": "chaqueta", "precio": 59.99, "stock": 1}
6
7  bufandas = {"nombre": "bufanda", "precio": 10.99, "stock": 3}
8
9  productos = [pantalones, chaquetas, bufandas]
10
11 def askForInfo(nombreProducto):
12     for producto in productos:
13         if producto["nombre"] == nombreProducto:
14             return producto["precio"], producto["stock"]
15         break
16     else:
17         return 0, 0
18
19 nombre = input("¿Qué producto desea consultar? -> ")
20 precio, cantidad = askForInfo(nombre)
21 if precio == 0 and cantidad == 0:
22     print("Lo sentimos, el producto que buscas no existe")
23 else:
24     print(
25         f"El producto {nombre} vale {precio}€ y disponemos de {cantidad} unidades")
```

En las líneas 3, 5, 7 creamos los correspondientes diccionarios.

En la línea 9 las unificamos con el diccionario productos.

En la línea 11 creamos una función para localizar el producto que después de la función el usuario pedirá por teclado.

Esta función nos retornará dos valores, si no lo encuentra retornará los valores 0.

Este será el resultado:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe
"c:/Users/pmver/Desktop/Ejer Python/Capitulo3.py"
¿Qué producto desea consultar? -> pantalón
El producto pantalón vale 39.99€ y disponemos de 10 unidades
PS C:\Users\pmver\Desktop\Ejer Python> 
```

11.- ¿Qué es *ARGS y **KWARGS en Python? Funciones con argumentos opcionales

```
1  import os
2  #  Capítulos 11: Funciones - Capítulo dos: argumentos arbitrarios y keyword arguments
3  #
4  #  Apartado 1: Argumentos arbitrarios:
5  #  Los argumentos arbitrarios se utilizan cuando no sabemos
6  #  de forma exacta el número de parámetros que la función va a
7  #  recibir.
8  input("Presiona 'enter' para avanzar al apartado -> 1. Argumentos arbitrarios ")
9  os.system("cls")
10
11 def maximo(*args):
12     maximo = args[0]
13     for arg in args[1:]:
14         if arg > maximo:
15             maximo = arg
16     return maximo
17
18 print("El máximo es: ")
19 print(maximo(0, 90, 23, 11, 10, -5))
20 print("El máximo es: ")
21 print(maximo(0, 110, 23, 110.45, 10.23, -10, 33, 55))
```

En la línea 11 definimos una función pero no sabemos el número de argumentos que vamos a necesitar, para esto vamos a utilizar argumentos arbitrarios.

En la línea 12 a la variable máximo le asignamos el valor del primer argumento.

En la línea 13 con un ciclo for recorremos desde el segundo argumento hasta el último, la variable arg irá asumiendo los valores en cada iteración.

En la línea 14 si la variable arg es mayor a máximo.

En la línea 15 si se cumple la condición anterior a la variable máximo le asignamos el valor de la variable arg.

En la línea 16 Una vez terminado el bucle retornamos el valor de la variable máximo.

En la línea 19 y 21 llamamos a la función pero con distinto número de parámetros.

Este será el resultado:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

El máximo es:
90
El máximo es:
110.45
PS C:\Users\pmver\Desktop\Ejer Python> █
```

En el siguiente ejemplo queremos crear una función para poder pasar parámetros de vídeo o de audio, El problema está que si pasamos los parámetros de un audio, no necesita más parámetros , en cambio sí pasamos un parámetro de video, este nos puede pedir la resolución o bien la resolución más fps.

```

1 def formatoDescarga(tipo, *args):
2     numArgs = len(args)
3     if tipo == "video":
4         if numArgs == 0:
5             print(f"El formato seleccionado es:\n-Tipo de archivo: {tipo}")
6         elif numArgs == 1:
7             resolucion = args[0]
8             print(f"El formato seleccionado es:\n-Tipo de archivo: {tipo}\n-Resolución: {resolucion}")
9         elif numArgs == 2:
10            resolucion = args[0]
11            fps = args[1]
12            print(f"El formato seleccionado es:\n-Tipo de archivo: {tipo}\n-Resolución: {resolucion}\n-FPS: {fps}")
13    elif tipo == "audio":
14        print(f"El formato seleccionado es:\n-Tipo de archivo: {tipo}")
15    else:
16        print("Error: el formato que pides no existe.")
17
18 formatoDescarga("audio")
19 formatoDescarga("video", 720)
20 formatoDescarga("video", 1080, 60)

```

Línea 1 Vamos a crear una función llamada formatoDescarga que tendrá un argumento obligatorio y además puede tener otros argumentos opcionales (arbitrarios).

Línea 2 En la variable numArgumentos almacenaremos el número de argumentos opcionales.

Línea 3 Si tipo es igual a "vídeo"

Línea 4 Cumpliéndose la condición anterior y además si numArgs es igual a 0.

Línea 5 Si se cumplen las dos condiciones anteriores mostrará el texto con formato (f"El formato seleccionado es:\n-Tipo de archivo: {tipo}").

Línea 6 sino si numArgs es igual a 1.

Línea 7 a la variable resolución le pasamos el primer argumento opcional.

Línea 8 anteriores mostrará el texto con formato (f"El formato seleccionado es:\n-Tipo de archivo: {tipo}" \n-Resolución: {resolución}).

Línea 9 sino si numArgs es igual a 2, le pasamos el primer y segundo argumento opcional.

Línea 10 a la variable resolución le pasamos el primer argumento opcional.

Línea 11 a la variable fps le pasamos el segundo argumento opcional.

Línea 12 mostrará el texto con formato (f"El formato seleccionado es:\n-Tipo de archivo: {tipo}" \n-Resolución: {resolución} \n-FPS: {fps}).

Línea 13 comparamos si el argumento obligatorio es igual a "audio"

Línea 14 mostrará (f"El formato seleccionado es:\n-Tipo de archivo: {tipo}").

Línea 15 si no mostrará el mensaje "Error. El formato que pides no existe.")

Las líneas 18, 19 y 30 llamamos a la función fotmatoDescargar con distintos argumentos opcionales.

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Cap11.py"

```

```

El formato seleccionado es:
-Tipo de archivo: audio

```

```

El formato seleccionado es:
-Tipo de archivo: video
-Resolución: 720

```

```

El formato seleccionado es:
-Tipo de archivo: video
-Resolución: 1080
-FPS: 60

```

```

PS C:\Users\pmver\Desktop\Ejer Python>

```

```

1  import os
2  # Apartado 2: Keyword Arguments:
3
4  # Podemos pasar los argumentos de una función
5  # mediante keywords. De esta forma conseguimos
6  # que el orden de los argumentos sea indiferente.
7
8  input("Presiona 'enter' para avanzar al apartado -> 2. Keyword Arguments ")
9  os.system("cls")
10
11
12  def crearPersonaje(clase, raza, nombre):
13      print(f"{nombre} es un {clase} de raza {raza}")
14
15  crearPersonaje(clase="mago", nombre="Thorofin", raza="Elfo")
16  crearPersonaje(nombre="Askland", clase="Pícaro", raza="Humana")

```

Si queremos que el orden de los argumentos no coincidan con los de la función, tendremos que especificarlos cuando llamemos a la función como se muestra en la línea 15.

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

Thorofin es un mago de raza Elfo
Askland es un Pícaro de raza Humana
PS C:\Users\pmver\Desktop\Ejer Python> █

```

```

1  # Apartado 3: Keyword Arbitrary Arguments:
2
3  # Se utilizan cuando no sabemos cuántos argumentos
4  # de palabra clave vamos a recibir
5
6  def printKwargs(**kwargs):
7      print("\n")
8      print("Los atributos del personaje son:")
9      for clave, valor in kwargs.items():
10         print(f"{clave} ---> {valor}")
11
12
13  printKwargs(clase="mago", nombre="Thorofin", raza="Elfo")
14  printKwargs(clase="mago", nombre="Thorofin", raza="Elfo", mascota="Dragón", nivel="165", clan="ValenciaFC")

```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Cap11.py"

```

```

Los atributos del personaje son:
clase ---> mago
nombre ---> Thorofin
raza ---> Elfo

```

} Línea 13

```

Los atributos del personaje son:
clase ---> mago
nombre ---> Thorofin
raza ---> Elfo
mascota ---> Dragón
nivel ---> 165
clan ---> ValenciaFC
PS C:\Users\pmver\Desktop\Ejer Python>

```

Línea 14

```

1  # Apartado 4: Combinación de todos los anteriores
2  # Se pueden usar conjuntamente en una misma función
3
4
5  def crearPersonaje(nombre, *args, **kwargs):
6      descripcion = "#### " + nombre.upper() + " ####\n\n"
7      descripcion += "#### DESCRIPCIÓN ####\n\n"
8      for clave, valor in kwargs.items():
9          descripcion += f"- {clave} ---> {valor}\n"
10
11     descripcion += "\n#### HABILIDADES ####\n\n"
12
13     for skill in args:
14         descripcion += f"- {skill}\n"
15     return descripcion
16
17
18     pj = crearPersonaje("Dandelion", "ataque fuerte", "esquivar",
19         "bomba de humo", raza="Elfo", mascota="Dragón", nivel="165", clan="ValenciaFC")
20     print(pj)

```

Al crear una función con todos los argumentos anteriores, detallamos como se emprimen los bloques.

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Cap11.py"
#### DANDELION ####
#### DESCRIPCIÓN ####
- raza ---> Elfo
- mascota ---> Dragón
- nivel ---> 165
- clan ---> ValenciaFC
#### HABILIDADES ####
- ataque fuerte
- esquivar
- bomba de humo
PS C:\Users\pmver\Desktop\Ejer Python>

```

12.- LAS TUPLAS (Sintaxis Básica)

```
1  import os
2  #  Capítulos 12: Tuplas
3  #
4  #  Apartado 1: Tuplas - Conceptos básicos.
5  #  Las tuplas son conjuntos ordenados de datos que no se pueden modificar.
6
7
8  input("Presiona 'enter' para avanzar al apartado -> 1. Tuplas - Conceptos básicos ")
9  os.system("cls")
10
11  mitupla = ("BadBunny", "Anuel", "Daddy Yankee", "Wisin")
12  print("El valor de mitupla es: ")
13  print(mitupla)
14  print(f"La longitud de mitupla es: {len(mitupla)}")
15
16  print("Iterar sobre una tupla: ")
17  for cantante in mitupla:
18      print(f"Cantante número {mitupla.index(cantante)}: {cantante}")
```

Línea 11 creamos una tupla con el nombre de mitupla, que contendrá 4 elementos. La característica de las tuplas es que estas no pueden ser modificadas.

Línea 13 mostramos por consola mitupla.

Línea 17 Como en las listas hacemos una iteración con el ciclo for para mostrar todos los elementos, utilizamos mitupla.index(cantante) para que nos muestra la posición de cada elemento.

Este será el resultado:

PROBLEMAS	SALIDA	CONSOLA DE DEPURACIÓN	TERMINAL
-----------	--------	-----------------------	----------

```
El valor de mitupla es:
('BadBunny', 'Anuel', 'Daddy Yankee', 'Wisin')
La longitud de mitupla es: 4
Iterar sobre una tupla:
Cantante número 0: BadBunny
Cantante número 1: Anuel
Cantante número 2: Daddy Yankee
Cantante número 3: Wisin
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```

1  import os
2  # Apartado 2: Indexado de Tuplas
3
4  # Podemos indexar los elementos de una tupla de la misma
5  # forma que las listas.
6  input("Presiona 'enter' para avanzar al apartado -> 2. Indexado de Tuplas ")
7  os.system("cls")
8
9  mitupla = ("BadBunny", "Anuel", "Daddy Yankee", "Wisin")
10 print("El valor de mitupla es: ")
11 print(mitupla)
12
13 primero = mitupla[0]
14 print(f"Primer elemento de la tupla: {primero}")
15
16 ultimo = mitupla[-1]
17 print(f"Último elemento de la tupla: {ultimo}")
18
19 print("Subtupla")
20 subtupla = mitupla[1:3]
21 print(subtupla)

```

Línea 11 muestra los elementos de la tupla.

Línea 13 A la variable primero se le asigna el primer elemento de la tupla.

Línea 16 A la variable ultimo se le asigna el último elemento de la tupla con -1, igual que las listas.

Línea 20 a la variable subtupla se le asignan los elementos desde el segundo hasta el tercero, ya que el cuarto no se muestra.

Este será el resultado:

PROBLEMAS	SALIDA	CONSOLA DE DEPURACIÓN	TERMINAL
			<pre> El valor de mitupla es: ('BadBunny', 'Anuel', 'Daddy Yankee', 'Wisin') Primer elemento de la tupla: BadBunny Último elemento de la tupla: Wisin Subtupla ('Anuel', 'Daddy Yankee') PS C:\Users\pmver\Desktop\Ejer Python> </pre>

```

1  import os
2  import logging
3  # Apartado 3: Modificar tuplas
4
5  # No podemos modificar directamente una tupla. Veremos
6  # qué errores ocurren y cómo transformar una tupla a
7  # una lista y viceversa.
8  input("Presiona 'enter' para avanzar al apartado -> 3. Modificar Tuplas ")
9  os.system("cls")
10
11 mitupla = ("BadBunny", "Anuel", "Daddy Yankee", "Wisin")
12 print("El valor de mitupla es: ")
13 print(mitupla)
14
15 try:
16     del mitupla[2]
17 except BaseException:
18     logging.exception("El error cuando intentamos eliminar un elemento de una tupla es: ")
19
20 try:
21     mitupla.append("Yandel")
22 except BaseException:
23     logging.exception("El error cuando intentamos añadir un elemento nuevo a una tupla es: ")
24
25 try:
26     mitupla[3] = "Yandel"
27 except BaseException:
28     logging.exception("El error cuando intentamos modificar un elemento nuevo a una tupla es: ")
29
30
31 print("\n\nConvertir tupla -> lista: ")
32 milista = list(mitupla)
33 print(milista)
34 print("Añadimos un nuevo elemento a la lista: ")
35 milista.append("Yandel")
36 print(milista)
37 print("Convertimos la lista a una tupla: ")
38 mitupla = tuple(milista)
39 print(mitupla)
40 print("Unir varias tuplas: ")
41 mitupla2 = ("Don Omar", "Tego Calderon", "Kendo Kapone")
42 mitupla3 = mitupla + mitupla2
43 print(mitupla3)

```

Convierte una tupla el lista.

Convierte una lista en tupla.
Realiza un copia de tupla.

Unir varias tuplas en una nueva tupla.

Línea 11 creamos una tupla llamada mitupla con 4 elementos.

En la línea 15, 20 y 25 con la sentencia try: al provocar un error intentando modificar las tuplas y que el programa no se interrumpa agregamos la sentencia except BaseException: con su correspondiente mensaje seleccionado con un recuadro azul.

Línea 16 intentamos eliminar un elemento 3 de la tupla, provocando el correspondiente error.

Línea 21 intentamos agregar un elemento en la tupla, provocando el correspondiente error.

Línea 26 intentamos modificar el elemento 4 de la tupla, provocando el correspondiente error.

Línea 32 convertimos tupla a lista.

Línea 35 Añadimos un nuevo elemento a la nueva lista creada.

Línea 38 convertimos lista a tupla.

Línea 42 unimos dos tuplas en una tercera llamada mitupla3.

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

Primer elemento de la tupla: BadBunny

Último elemento de la tupla: Wisin

Subtupla

El valor de mitupla es:

('BadBunny', 'Anuel', 'Daddy Yankee', 'Wisin')

ERROR:root:El error cuando intentamos eliminar un elemento de una tupla es:

Traceback (most recent call last):

File "c:\Users\pmver\Desktop\Ejer Python\Cap12.py", line 16, in <module>

del mitupla[2]

~~~~~^^^

TypeError: 'tuple' object doesn't support item deletion

ERROR:root:El error cuando intentamos añadir un elemento nuevo a una tupla es:

Traceback (most recent call last):

File "c:\Users\pmver\Desktop\Ejer Python\Cap12.py", line 21, in <module>

mitupla.append("Yandel")

^^^^^^^^^^^^^^

AttributeError: 'tuple' object has no attribute 'append'

ERROR:root:El error cuando intentamos modificar un elemento nuevo a una tupla es:

Traceback (most recent call last):

File "c:\Users\pmver\Desktop\Ejer Python\Cap12.py", line 26, in <module>

mitupla[3] = "Yandel"

~~~~~^^^

TypeError: 'tuple' object does not support item assignment

Convertir tupla -> lista:

['BadBunny', 'Anuel', 'Daddy Yankee', 'Wisin']

Añadimos un nuevo elemento a la lista:

['BadBunny', 'Anuel', 'Daddy Yankee', 'Wisin', 'Yandel']

Convertimos la lista a una tupla:

('BadBunny', 'Anuel', 'Daddy Yankee', 'Wisin', 'Yandel')

Unir varias tuplas:

('BadBunny', 'Anuel', 'Daddy Yankee', 'Wisin', 'Yandel', 'Don Omar', 'Tego Calderon', 'Kendo Kapone')

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Cap12.py"

Presiona 'enter' para avanzar al apartado -> 3. Modificar Tuplas []

```

1 import os
2 # Apartado 4: Empaquetar y desempaquetar tuplas
3
4 # De la misma forma que podemos agrupar múltiples
5 # variables en una tupla (empaquetar), también podemos
6 # asignar cada elemento de una tupla a una variable distinta.
7
8
9 input("Presiona 'enter' para avanzar al apartado -> 4. Empaquetar y desempaquetar tuplas")
10 os.system("cls")
11
12 mitupla = ("BadBunny", "Anuel", "Daddy Yankee")
13 print("El valor de mitupla es: ")
14 print(mitupla)
15 print("Desempaquetar los elementos contenidos en una tupla")
16 (artista1, artista2, artista3) = mitupla
17 print("Artista 1: {}. Artista 2 {}. Artista 3: {}".format(artista1, artista2, artista3))
18
19 mitupla = ("BadBunny", "Anuel", "Daddy Yankee", "Myke Towers", "Farruko", "C.Tangana")
20 print("El valor de mitupla es: ")
21 print(mitupla)
22 print("Desempaquetar: ")
23
24 (primerArtista, segundoArtista, *variosArtistas) = mitupla
25 print(f"Primer artista: {primerArtista}. Segundo Artista {segundoArtista}. Varios artistas: ")
26 print(variosArtistas)
27
28
29 (primerArtista, *variosArtistas, ultimoArtista) = mitupla
30 print(f"Primer artista: {primerArtista}. Último Artista {ultimoArtista}. Varios artistas: ")
31 print(variosArtistas)

```

En este ejemplo vamos a ver como empaquetar y desempaquetar tuplas.

Línea 12 creamos una tupla de 3 elementos.

Línea 16 A las variables artista1, artista2 y artista3 le asignamos los elementos respectivos de la tupla.

Línea 19 creamos una tupla de 6 elementos.

Línea 24 La variable primerArtista asume el primer elemento de la tupla, segundoArtista asume el segundo valor de la lista y la lista variosArtistas asume el resto de valores.

Línea 29 primerArtista, para el primer elemento, ultimoArtista para el último elemento y los que se encuentran entre el primero y el último una lista con el nombre variosArtistas.

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

El valor de mitupla es:
('BadBunny', 'Anuel', 'Daddy Yankee')
Desempaquetar los elementos contenidos en una tupla
Artista 1: BadBunny. Artista 2 Anuel. Artista 3: Daddy Yankee
El valor de mitupla es:
('BadBunny', 'Anuel', 'Daddy Yankee', 'Myke Towers', 'Farruko', 'C.Tangana')
Desempaquetar:
Primer artista: BadBunny. Segundo Artista Anuel. Varios artistas:
['Daddy Yankee', 'Myke Towers', 'Farruko', 'C.Tangana']
Primer artista: BadBunny. Último Artista C.Tangana. Varios artistas:
['Anuel', 'Daddy Yankee', 'Myke Towers', 'Farruko']
PS C:\Users\pmver\Desktop\Ejer Python> █

```

```

1  # Apartado 5: *Extra* Listas de tuplas
2
3  # Podemos crear listas cuyos elementos
4  # sean tuplas. De esta forma, podemos
5  # iterar sobre las diversas tuplas
6  # y además desempaquetar sus elementos.
7
8  comida = [("primero", "macarrones"), ("segundo", "pollo"), ("postre", "flan")]
9  print("El menú de hoy es: ")
10
11  for plato, nombre in comida:
12      print(f"{plato} --> {nombre}")
13
14
15  # De hecho, esto es lo que realmente ocurre
16  # cuando utilizamos el método items sobre un diccionario.
17
18  artistas = {1: "C.Tangana", 2: "Duki", 3: "Khea"}
19  print("Artistas: ")
20  print(artistas)
21
22  listaTuplas = artistas.items()
23  print(listaTuplas)
24
25
26  for clave, valor in listaTuplas:
27      print(f"{clave}: {valor}")

```

Línea 8 creamos una lista llamada comida que contiene 3 tuplas.

Línea 11 Podemos crear un bucle for para iterar por las correspondientes tuplas como si se tratara de diccionarios.

Línea 18 ejemplo cuando utilizamos el método ítems sobre un diccionario.

Línea 22 A la variable listaTuplas le asignamos la posición de cada elemento y su elemento.

Línea 26 Con un bucle for realizamos una iteración para ir leyendo la clave y el valor de listasTuplas.

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Cap12.py"

```

```

El menú de hoy es:
primero --> macarrones
segundo --> pollo
postre --> flan

```

```

Artistas:
{1: 'C.Tangana', 2: 'Duki', 3: 'Khea'}
dict_items([(1, 'C.Tangana'), (2, 'Duki'), (3, 'Khea')])
1: C.Tangana
2: Duki
3: Khea
PS C:\Users\pmver\Desktop\Ejer Python>

```

13.- ¿Qué es la Programación Orientada a Objetos en Python?

¿Qué es la POO?

Es un paradigma de programación:

- Imita los objetos de la vida real, con sus propiedades y comportamientos, en los lenguajes de programación.

¿Qué nos aporta?

- Generar código mucho más modular. Significa que si nosotros estamos realizando una tarea mucho más compleja, lo vamos a poder dividir en pequeñas tareas que sean más sencillas y poderlas resolver de una a una de forma más simple. De este modo tendremos un código más estructurado y se visualiza con mejor claridad.
- Reutilización del código. Como ejemplo podríamos crear un objeto encargado de leer ficheros de Excel y este código lo podremos reutilizar para más de un fichero dependiendo de lo extenso que sea el fichero.
- Encapsulamiento y simplificación, que trataremos en el próximo capítulo.



¿Qué es un objeto?

Definición:

- Un objeto se forma principalmente por dos elementos:
 - Datos: Describen el estado o las características del objeto. A los datos de un objeto los llamaremos **propiedades** o **atributos**.
 - Código: Ejecuta ciertas funcionalidades o comportamientos de dicho objeto. Los llamaremos **métodos**.



Ejemplo:

- Objeto: Camiseta.
 - **Atributos:** Una camiseta puede tener un **precio**, una **marca**, un **color** y una **talla**.
 - **Métodos:** Podemos rebajar el precio o teñirla para cambiar su color.

En el ejemplo de las camisetas podemos ver que hay de diferente marca, precio, color y talla, tienen los mismos atributos pero con distintos valores, pero en el fondo todo son camisetas. Tenemos que utilizar un modo de agrupar estos objetos.

Las Clases

Definición:

- Una clase es el modelo para un grupo de objetos.
- La clase define los métodos y atributos comunes de todos los objetos que pertenecen a la misma clase.
- A un objeto que pertenece a una clase se le llama **instancia**.

Ejemplo:

- Clase: Camisetas. Todos los objetos de la clase camisetas tendrán un precio, una marca, un color y una talla.
- Cada instancia de Camisetas tendrá un precio, color y talla distintos.

Las Clases: Ejemplo



Pilares de la POO



Que comentaremos en los siguientes vídeos.

Vamos a crear un nuevo fichero de Python.

Para entender el procedimiento, primero vamos a realizar de una forma no correcta, pero con la finalidad de que se entienda este proceso hasta que al final se planteará correctamente.

Lo primero que vamos a realizar es crear la clase.

```
1 class Camiseta:
```

Vamos a crear los atributos.

```
1 class Camiseta:
2     talla = "XL"
3     color = "Rojo"
4     precio = 20.99
5     marca = "Adimas"
```

Ahora vamos a crear un objeto de esta clase.

```
1 class Camiseta:
2     talla = "XL"
3     color = "Rojo"
4     precio = 20.99
5     marca = "Adimas"
6
7 CamisetaRoja = Camiseta()
```

Hemos instanciado un objeto de la clase Camiseta().

¿Cómo podemos acceder a un atributo?

```
9 print(CamisetaRoja.color)
```

Para ello tenemos que escribir la instancia, seguido de un punto y a continuación escribir el atributo.

Si lo ejecutamos este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Poo.py"
```

Rojo

```
PS C:\Users\pmver\Desktop\Ejer Python>
```

Como podréis observar con este ejemplo no podemos reutilizar las instancias porque en la clase hemos agregado las propiedades, vamos a modificar el código para ver como se hace.

Lo que hemos explicado hasta ahora era un ejemplo académico.

Vamos a crear un método llamado `__init__()` que se ejecuta automáticamente al instanciar una clase.

```
1  class Camiseta:
2      def __init__(self):
```

El método `__init__()` se encarga de inicializar o de construir un objeto con unos determinados atributos.

El parámetro `self` la definición técnica es “Una referencia a la instancia actual de la clase” lo que hace es hacer referencia al objeto que estamos creando de esa clase.

```
1  class Camiseta:
2      def __init__(self):
3          self.marca = "Gicci"
4          self.precio = 100
5          self.talla = "M"
6          self.color = "negro"
7
8  camiseta = Camiseta()
9
10 print(camiseta.precio)
```

En la línea 8 creamos una instancia.

En la línea 10 con la nomenclatura del punto nos mostrará el precio de la nueva instancia `camiseta`.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Poo.py"
```

```
100
```

```
PS C:\Users\pmver\Desktop\Ejer Python> 
```

Pero aun no observamos como podemos hacer varias instancias con las mismas propiedades pero con valores distintos.

```

1  class Camiseta:
2      def __init__(self, marca, precio, talla, color):
3          self.marca = marca
4          self.precio = precio
5          self.talla = talla
6          self.color = color
7
8      camiseta1 = Camiseta("Nike", 19.99, "S", "Lila")
9      camiseta2 = Camiseta("Adidas", 30, "M", "Rojo")
10
11     print(camiseta1.precio)
12     print(camiseta2.marca)

```

En las líneas 8 y 9 al crear las instancias tenemos que pasarle las propiedades que tienen cada uno.

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Poo.py"

```

```

19.99
Adidas

```

```

PS C:\Users\pmver\Desktop\Ejer Python>

```

Ahora vamos a crear unos métodos para esta clase.

```

1  class Camiseta:
2      def __init__(self, marca, precio, talla, color):
3          self.marca = marca
4          self.precio = precio
5          self.talla = talla
6          self.color = color
7
8      def aplicarDescuento(self, porcentaje):
9          self.precio = self.precio - self.precio*porcentaje/100
10
11     camiseta1 = Camiseta("Nike", 19.99, "S", "Lila")
12     camiseta2 = Camiseta("Adidas", 30, "M", "Rojo")
13
14     print(camiseta2.precio)
15     camiseta2.aplicarDescuento(50)
16     print(camiseta2.precio)

```

En la línea 8 creamos un metodo para aplicar un descuento a una de las instancias.

En la línea 15 a la instancia camiseta2 llamamos al método aplicarDescuento(50) con la nomenclatura del punto y le aplicaremos un 50%, podrás observar que lo que valía 30 ahora vale 15.

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Poo.py"
```

```
30
```

```
15.0
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Ahora vamos a añadir un nuevo atributo de tipo boolean.

Creamos un segundo método llamado infoProducto(self).

Ejecutamos el método a las dos instancias.

```
1  class Camiseta:
2      def __init__(self, marca, precio, talla, color):
3          self.marca = marca
4          self.precio = precio
5          self.talla = talla
6          self.color = color
7          self.rebajada = False
8
9      def aplicarDescuento(self, porcentaje):
10         self.precio = self.precio - self.precio*porcentaje/100
11         if porcentaje < 100:
12             self.rebajada = True
13
14         def infoProducto(self):
15             info = f"Descripción de la camiseta:\nMarca: {self.marca}\nPrecio: {self.precio}\nTalla: {self.talla}\nColor: {self.color}\n"
16             if self.rebajada:
17                 info += "ESTE PRODUCTO ESTÁ REBAJADO"
18             return info
19
20     camiseta1 = Camiseta("Nike", 19.99, "S", "Lila")
21     camiseta2 = Camiseta("Adidas", 30, "M", "Rojo")
22
23     print(camiseta2.precio)
24     camiseta2.aplicarDescuento(50)
25     print(camiseta2.precio)
26
27     print(camiseta1.infoProducto())
28     print(camiseta2.infoProducto())
```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Poo.py"
30
15.0
Descripción de la camiseta:
Marca: Nike
Precio: 19.99
Talla: S
Color: Lila

Descripción de la camiseta:
Marca: Adidas
Precio: 15.0
Talla: M
Color: Rojo
ESTE PRODUCTO ESTÁ REBAJADO
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Ahora vamos a crear un método para cambiar el color a la camiseta.

```
20 def teñir(self, color):
21     self.color = color
```

Tenemos la instancia camiseta1 de color Lila, ahora la queremos de color verde.

```
33 camiseta1.teñir("Verde")
34 print(camiseta1.infoProducto())
```

Este será el resultado:

```
Descripción de la camiseta:
Marca: Nike
Precio: 19.99
Talla: S
Color: Verde
```

Si queremos que el campo del precio salga con 2 decimales y el símbolo del dólar.

```

def infoProducto(self):
    info = f"Descripción de la camiseta:\nMarca: {self.marca}\nPrecio: {self.precio:.2f}€\nTalla: {self.talla}\nColor: {self.color}\n"
    if self.rebajada:
        info += "ESTE PRODUCTO ESTÁ REBAJADO"
    return info

```

Resumen de todo el código:

class Camiseta:

def __init__(self, precio, marca, talla, color):

self.precio = precio

self.marca = marca

self.talla = talla

self.color = color

self.rebajada = False

def aplicarDescuento(self, porcentaje):

nuevoPrecio = self.precio - self.precio*porcentaje/100

self.precio = nuevoPrecio

if porcentaje < 100:

self.rebajada = True

def teñir(self, color):

self.color = color

def infoCamiseta(self):

info = f"Camiseta:\nTalla: {self.talla}\nPrecio: {self.precio:.2f}€\nMarca: {self.marca}\n"

if self.rebajada:

info += "Actualmente ésta camiseta está rebajada"

return info

camiseta = Camiseta(19.99, "Gucci", "XL", "Negro")

print(camiseta.infoCamiseta())

camiseta.aplicarDescuento(20)

print(camiseta.infoCamiseta())

print("\n#####\n")

camisetaAdimas = Camiseta(300, "Adimas", "M", "Rojo")

print(camisetaAdimas.color)

print(camisetaAdimas.marca)

camisetaAdimas.teñir("Verde")

print(camisetaAdimas.color)

14.- ¿Qué es la HERENCIA en Python?

¿Qué es la Herencia?

- La herencia es uno de los pilares de la programación orientada a objetos.
- Es un mecanismo o funcionalidad que permite que una clase reciba todas las características, métodos y atributos, de otra clase.
- La clase que **hereda**, tendrá todos los métodos y atributos de la clase de la cual está heredando. Además, tendrá también los **suyos propios**.

Nomenclatura de la herencia:

- **Superclase:** También conocida como clase padre. Es la clase de la cual se heredan las características. Suele ser más general y sirve de base para otras clases.
- **Subclase:** También conocida como clase hijo. Es la clase que recibe la herencia y amplía los métodos y atributos de la superclase con los suyos propios. Suele ser una clase más concreta o específica.



Superclase o
Clase Padre



Subclase o
Clase Hijo



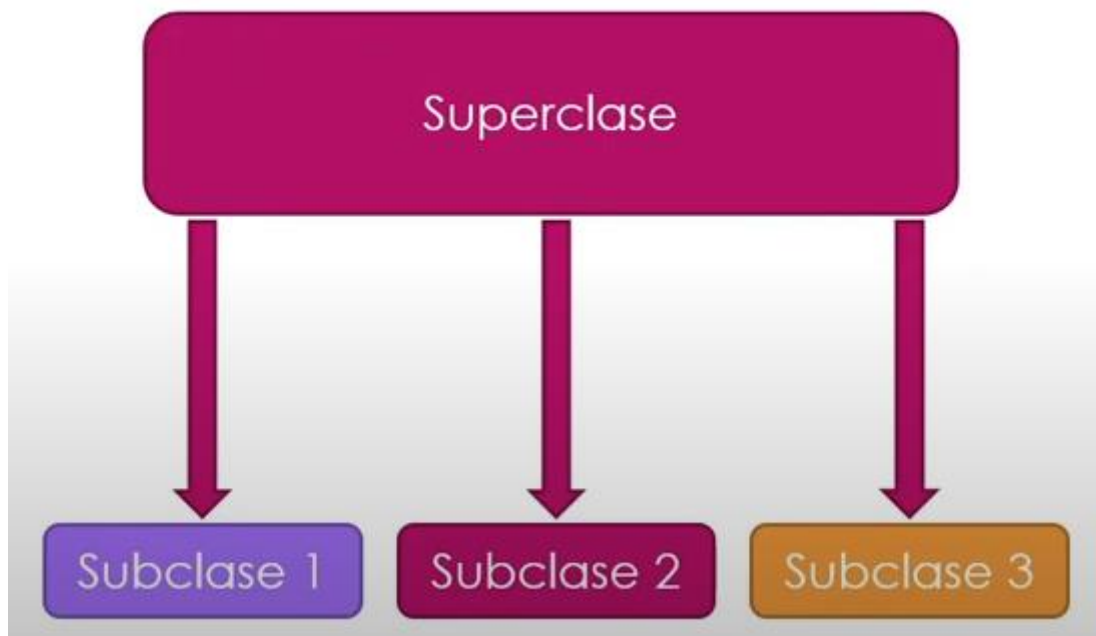
Tipos de Herencias: Herencias simples

- La herencia simple es la forma más básica de herencia que existe, como su nombre bien lo indica.
- Una subclase hereda de una superclase.



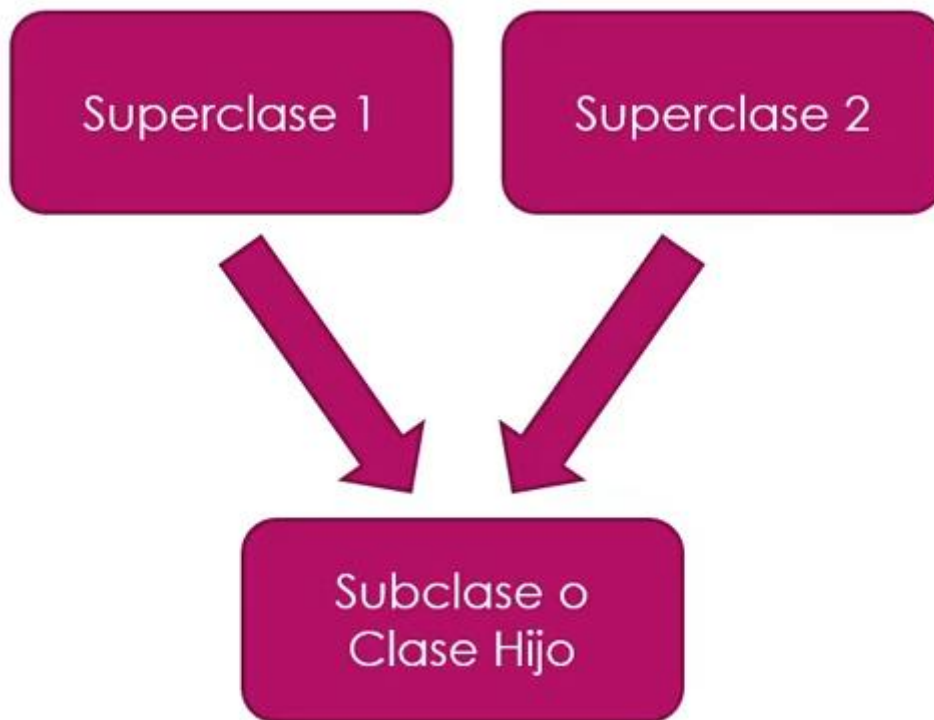
Tipos de Herencia: Herencia Jerárquica

- La herencia jerárquica se produce cuando múltiples subclases heredan de una única superclase.



Tipos de Herencia: Herencia múltiple

- Este tipo de herencia se produce cuando una subclase hereda de dos superclases al mismo tiempo.



Tipos de Herencia: Herencia Multinivel

- Una subclase hereda de una superclase, a su vez esta superclase hereda de otra superclase.



Vamos con un ejemplo práctico:

```

1  class Persona:
2      def __init__(self, nombre, edad, dni):
3          self.nombre = nombre
4          self.edad = edad
5          self.dni = dni
6
7      def presentarse(self):
8          print(f"¡Hola! Me llamo {self.nombre} y tengo {self.edad} años")
9
10 oscar = Persona("Oscar", 25, "12123123R")
11
12 oscar.presentarse()
13
14 print(oscar.dni)

```

En la línea 1 creamos la clase Persona.

En la línea 2 en el método constructor definimos los atributos como nombre, edad y dni.

En la línea 7 creamos un método llamada presentarse con la finalidad de que todas las instancias de la clase Persona puedan saludarse.

Línea 10 creamos una instancia llamada oscar, pasándoles las correspondientes propiedades.

Línea 12 a la instancia oscar llamamos al método presentarse.

Línea 14 vamos a mostrar el dni de oscar.

Este será el resultado:

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Herencias.py"

```

```

¡Hola! Me llamo Oscar y tengo 25 años
12123123R

```

```

PS C:\Users\pmver\Desktop\Ejer Python>

```

Ahora vamos a realizar nuestra primera herencia.

```

16  class Trabajador(Persona):
17      |      pass

```

Hemos creado la clase Trabajador que hereda de Persona, como esta clase no puede estar vacía ya que nos daría error, provisionalmente hemos puesto pass (no hacer nada).

```

19  trabajador = Trabajador("Juan", 33, "98987987H")
20  trabajador.presentarse()

```

Línea 19 creamos una instancia llamada trabajador de la clase Trabajador con sus correspondiente propiedades.

Línea 20 Ahora vamos a llamar el método .presentarse() ya que es un método que ha heredado.

Este será el resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Herencias.py"
¡Hola! Me llamo Oscar y tengo 25 años
12123123R
¡Hola! Me llamo Juan y tengo 33 años
PS C:\Users\pmver\Desktop\Ejer Python>
```

Nos genera un error por ahora la clase trabajador necesita los argumentos heredados más los suyos propios.

```
23 trabajador = Trabajador("Juan", 33, "98987987H", 1500, "Operario", "Manufacturas, S.A.")
```

Vamos a ejecutar de nuevo.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Herencias.py"
¡Hola! Me llamo Oscar y tengo 25 años
12123123R
¡Hola! Me llamo Juan y tengo 33 años
98987987H
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Ahora ya no muestra el error y la funciona el método heredado.

Ahora vamos a crear un nuevo método para la clase Trabajador.

```
16 class Trabajador(Persona):
17     def __init__(self, nombre, edad, dni, sueldo, cargo, empresa):
18         super().__init__(nombre, edad, dni)
19         self.sueldo = sueldo
20         self.cargo = cargo
21         self.empresa = empresa
22
23     def calcularSueldoAnual(self):
24         return self.sueldo * 12 + 2000
```

Calcula el sueldo mensual multiplicado por 12 más 2000 euros de pagas extras.

Ahora vamos a ejecutar este método con la instancia trabajador.

```
30 print(trabajador.calcularSueldoAnual())
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Herencias.py"
¡Hola! Me llamo Oscar y tengo 25 años
12123123R
¡Hola! Me llamo Juan y tengo 33 años
98987987H
20000
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Para ver la herencia jerárquica vamos a seguir con el ejemplo creando la clase Estudiantes.

```
26 class Estudiantes(Persona):
27     def __init__(self, nombre, edad, dni, universidad, curso, asignaturas):
28         super().__init__(nombre, edad, dni)
29         self.universidad = universidad
30         self.curso = curso
31         self.asignaturas = asignaturas
```

Vamos a crear un método para la clase Estudiantes.

```
26 class Estudiantes(Persona):
27     def __init__(self, nombre, edad, dni, universidad, curso, asignaturas):
28         super().__init__(nombre, edad, dni)
29         self.universidad = universidad
30         self.curso = curso
31         self.asignaturas = asignaturas
32
33     def describirse(self):
34         print(f"""¡Hola soy {self.nombre}!. Tengo {self.edad} años.
35         Estudio en la universidad de {self.universidad}.
36         Estoy realizando los estudios de {self.curso}.
37         Con las siguientes asignaturas: {self.asignaturas}
38         """)
```

Vamos a crear una instancia de tipo Estudiantes.

```
49 maria = Estudiantes("María", 20, "65123123G", "Universidad UAB", →
→ "3º Informática", ["Programación", "Redes", "Hardware"] )
```

Vamos a llamar al método.

```
51 maria.describirse()
```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "
c:/Users/pmver/Desktop/Ejer Python/Herencias.py"
```

```
¡Hola! Me llamo Oscar y tengo 25 años
```

```
12123123R
```

```
¡Hola! Me llamo Juan y tengo 33 años
```

```
98987987H
```

```
20000
```

```
¡Hola soy María!. Tengo 20 años.
  Estudio en la universidad de Universidad UAB.
  Estoy realizando los estudios de 3º Informática.
  Con las siguientes asignaturas: ['Programación', 'Redes', 'Hardware']
```

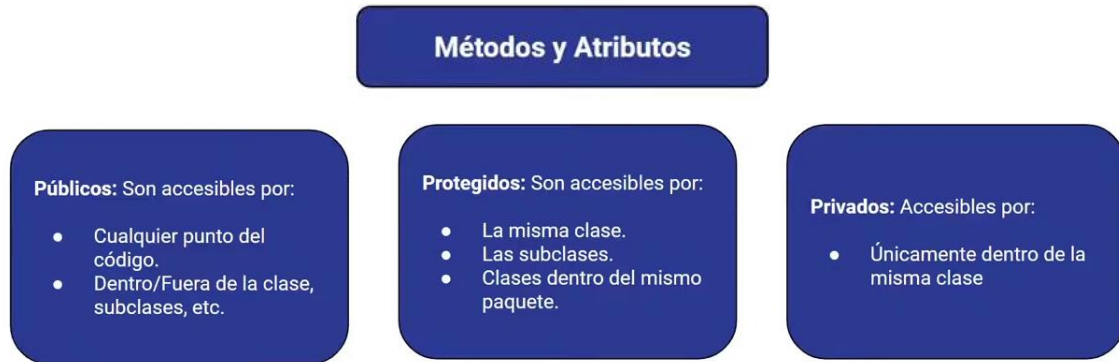
```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

15.- ¿Qué es el ENCAPSULAMIENTO en Python?

¿Qué es la encapsulación?

- Es uno de los pilares de la programación orientada a objetos.
- Permite regular el acceso a los métodos y atributos de una clase.
- En cierta manera, enmascara la complejidad de una clase.

Encapsulamiento: Modificadores de acceso



Utilidades del encapsulamiento

- Nos permite ocultar métodos y atributos fuera de la propia clase.
- Podemos regular la modificación. Crearemos métodos (públicos) para modificar los atributos de un objeto.
- Enmascarar la complejidad de algunos métodos haciéndolos privados y utilizarlos desde métodos públicos.

Encapsulamiento en Python

NO EXISTE

- Existe una convención, pero es solo sintaxis, no es funcional:
 - **`_atribulo`** = "**atributoProtegido**". Un guion bajo al principio del nombre de un atributo indica que es protegido.
 - **`__atribulo`** = **atributoPrivado**". Un doble guion bajo al principio del nombre de un atributo indica que es privado.
 - La **misma** sintaxis para los **métodos**.
 - Para Python, todos los atributos y métodos son **Públicos**. La convención simplemente se usa entre **programadores** para indicar cómo se debe usar los métodos y atributos de una clase
- En la práctica os explico más cosas con un ejemplo.

```

1  class Circulo:
2      def __init__(self, radio):
3          self.radio = radio
4          self.pi = 3.1415
5
6      def caldulaPerimetro(self):
7          return 2*self.pi*self.radio
8
9      def calcularArea(self):
10         return self.pi*self.radio**2

```

Línea 1 creamos una clase llamada Circulo.

Línea 2 Definimos el método constructor.

Línea 3 y 4 definimos los atributos.

Línea 6 definimos el método calcularPerímeto que retornará la siguiente fórmula:

$$\text{Perímetro} = 2 \times \pi \times r$$

Línea 9 definimos el método calcularArea que retornará la siguiente fórmula:

$$\text{Área} = \pi \times r^2$$

Ahora vamos a crear un instancia:

```
13  c1 = Circulo(2.5)
```

Vamos a llamar al método para que muestre en pantalla.

```

15  print(c1.calcularArea())
16  print(c1.caldulaPerimetro())

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python
/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Encapsulacion.py"
```

```
19.634375000000002
```

```
15.707500000000001
```

```
PS C:\Users\pmver\Desktop\Ejer Python>
```

Queremos limitar el resultado a 2 decimales.

```

15  print(f"Perímetro: {c1.calcularArea():.2f}")
16  print(f"Área: {c1.caldulaPerimetro():.2f}")

```

Este será el resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Encapsulacion.py"
```

```
Perímetro: 19.63
Área: 15.71
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Para mostrar la constante PI

```
17 print(f"La constnate PI es {c1.pi}")
```

Hasta ahora, todo esto ya lo sabíamos, a partir de ahora vamos a trabajar con el encapsulamiento.

```
1  class Circulo:
2      def __init__(self, radio):
3          self._radio = radio
4          self._pi = 3.1415
5
6      def caldulaPerimetro(self):
7          return 2*self._pi*self._radio
8
9      def calcularArea(self):
10         return self._pi*self._radio**2
11
12
13 c1 = Circulo(2.5)
14 print(f"Perímetro: {c1.calcularArea():.2f}")
15 print(f"Área: {c1.caldulaPerimetro():.2f}")
16 print(f"La constnate PI es {c1._pi}")
```

A todas las propiedades les hemos agregado un guion bajo al principio para proteger los datos, pero como podréis observar en lugar de producirse un error, muestra los datos, con este ejemplo confirmamos la no existencia de encapsulamiento en Python.

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Encapsulacion.py"
```

```
Perímetro: 19.63
```

```
Área: 15.71
```

```
La constnate PI es 3.1415
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Vamos a cambiar el guion bajo por dos guiones bajos.

```

1  class Circulo:
2      def __init__(self, radio):
3          self.__radio = radio
4          self.__pi = 3.1415
5
6      def caldulaPerimetro(self):
7          return 2*self.__pi*self.__radio
8
9      def calcularArea(self):
10         return self.__pi*self.__radio**2
11
12
13     c1 = Circulo(2.5)
14     print(f"Perímetro: {c1.calcularArea():.2f}")
15     print(f"Área: {c1.caldulaPerimetro():.2f}")
16     print(f"La constnate PI es {c1.__pi}")

```

Si ejecutamos:

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Encapsulacion.py"
Perímetro: 19.63
Área: 15.71
Traceback (most recent call last):
  File "c:\Users\pmver\Desktop\Ejer Python\Encapsulacion.py", line 16, in <module>
    print(f"La constnate PI es {c1.__pi}")
    ^^^^^^^
AttributeError: 'Circulo' object has no attribute '__pi'
PS C:\Users\pmver\Desktop\Ejer Python>

```

Como podrás observar nos sale el correspondiente error.

Lo que está pasando es que tu en Python cuando defines un método o un atributo que empieza con dos guiones bajos, lo que hace Python automáticamente es lo siguiente:

Si escribimos atributo: `__radio` (Lo convierte en) `→ _Circulo__radio`.

Lo cambia a guion bajo seguido del nombre de la clase doble guion bajo y el nombre de la propiedad o el método.

Vamos a cambiar el patrón:

```

16     print(f"La constnate PI es {c1._Circulo__pi}")

```

Ahora cuando ejecutemos podremos obtener su valor.

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Encapsulacion.py"
Perímetro: 19.63
Área: 15.71
La constnate PI es 3.1415
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Lo que está ocurriendo es que Python se encarga de que esto ocurra cuando nosotros definimos un método o un atributo dentro de una clase empezando por los dos guiones bajos, el automáticamente añade un guion bajo el nombre de la clase y el atributo o método que hayamos definido. Por tanto esto no es privado ahora la pregunta es ¿Por qué existe esta funcionalidad? Es una funcionalidad que Python implementa para evitar que métodos o propiedades con el mismo nombre en distintas clases puedan colisionar. Añadiendo el nombre de la clase ya las puede diferenciar de otras clases.

```
1  class Circulo:
2      def __init__(self, radio):
3          self.radio = radio
4          self.__pi = 3.1415
5
6      def caldulaPerimetro(self):
7          return 2*self.__pi*self.radio
8
9      def calcularArea(self):
10         return self.__pi*self.radio**2
11
12     def getPi(self):
13         return self.__pi
14
15     c1 = Circulo(2.5)
16     print(f"Perímetro: {c1.calcularArea():.2f}")
17     print(f"Área: {c1.caldulaPerimetro():.2f}")
18     print(f"La constnate PI es {c1.getPi()}")
```

Ahora solo dejaremos el atributo pi con dos guiones bajos, al resto se los quitamos.

En la línea 12 hacemos un método público que nos retorna el valor de pi.

En la línea 18 llamamos al método getPi() y de esta forma obtendremos su valor.

Este será el resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Encapsulacion.py"
```

```
Perímetro: 19.63
```

```
Área: 15.71
```

```
La constnate PI es 3.1415
```

```
PS C:\Users\pmver\Desktop\Ejer Python> []
```

Ahora vamos a hacer que el radio sea privado.

```
1  class Circulo:
2      def __init__(self, radio):
3          self.__radio = radio
4          self.__pi = 3.1415
5
6      def caldulaPerimetro(self):
7          return 2*self.__pi*self.__radio
8
9      def calcularArea(self):
10         return self.__pi*self.__radio**2
11
12     def getPi(self):
13         return self.__pi
14
15     c1 = Circulo(2.5)
16     print(f"Perímetro: {c1.calcularArea():.2f}")
17     print(f"Área: {c1.caldulaPerimetro():.2f}")
18     print(f"La constnate PI es {c1.getPi()}")
```

Ahora vamos a crear el método:

Para que no podamos asignarle valores que no se corresponden a esta propiedad.

```
14
15  def setRadio(self, nuevoValor):
16      if type(nuevoValor) == int or type(nuevoValor) == float:
17          if (nuevoValor > 0):
18              self.__radio = nuevoValor
19              print(f"El radio se ha modificado a {self.__radio}")
20          else:
21              print("Oye el radio no puede ser negativo")
22      else:
23          print("El radio tiene que ser un número positivo")
```

```

24
25
26  c1 = Circulo(2.5)
27  print(f"Perímetro: {c1.calcularArea():.2f}")
28  print(f"Área: {c1.calcularPerimetro():.2f}")
29  print(f"La constnate PI es {c1.getPi()}")
30
31  c1.setRadio(34)
32  c1.setRadio(-23)
33  c1.setRadio("Holaquetal")

```

En la línea 15 creamos el método publico setRadio pasándole el argumento nuevoValor.

En la línea 16 comparamos que el valor tiene que ser de tipo int o float.

En la línea 17 comprobamos que nuevoValor sea mayor a 0.

En la línea 18 a la propiedad __radio le asignamos el valor que le pasaremos cuando llamemos al método.

En la línea 19 nos notifica "El radio se ha modificado y muestra su nuevo valor.

En la línea 20 si no se a cumplido la condición de la línea 15 que nos diga "Oye el radio no puede ser negativo".

En la línea 22 si la condición de la línea 16 no se cumple que nos diga "El radio tiene que ser un número positivo".

En las líneas 31, 32 y 33 llamamos al método con distintos valores, el valor 34 es correcto, en cambio los valores -23 y "Holaquetal" no son correctos.

Cuando ejecutemos este será el resultado:

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Encapsulacion.py"
Perímetro: 19.63
Área: 15.71
La constnate PI es 3.1415
El radio se ha modificado a 34
Oye el radio no puede ser negativo
El radio tiene que ser un número positivo
PS C:\Users\pmver\Desktop\Ejer Python> █

```

En el primer método conseguimos cambiar el valor de Pi, en cambio en los dos siguientes no permitimos que se guarden los cambios ya que los datos introducidos no son correctos.

Adjunto el código completo para comprobar posibles errores que tengas.

```
class Circulo:
    def __init__(self, radio):
        self.__radio = radio
        self.__pi = 3.1415

    def calcularPerimetro(self):
        return 2*self.__pi*self.__radio

    def calcularArea(self):
        return self.__pi*self.__radio**2

    def getPi(self):
        return self.__pi

    def setRadio(self, nuevoValor):
        if type(nuevoValor) == int or type(nuevoValor) == float:
            if nuevoValor > 0:
                self.__radio = nuevoValor
                print(f"El radio se ha modificado correctamente: {self.__radio}")
            else:
                print("Oye el radio no puede ser negativo")
        else:
            print("El radio tiene que ser un número positivo")

c1 = Circulo(2.5)
print(c1.calcularArea())
print(c1.calcularPerimetro())
print(f"La constante PI es {c1.getPi()}")
c1.setRadio(34)
c1.setRadio(-23)
c1.setRadio("holaquetal")

# atributo: __radio -> _Circulo__radio . En general _NombreDeLaClase__nombreDelAtributo
```

16.- ¿Qué es el POLIMORFISMO en Python?

El Polimorfismo es simplemente una herramienta que nos va a permitir redefinir métodos que nosotros heredemos de una clase padre.

Por ejemplo si yo defino un método en una superclase y de esta superclase heredan tres subclases más en cada una de estas subclases bajo el mismo nombre voy a poder modificar, definir de nuevo el código que se ejecuta en estos métodos, por tanto la subclase1 si yo tengo un método que se llama saludar, puede ser distinto que en la subclase2 que tiene el mismo método que también se llama saludar y que pueda tener un código diferente. Porque, gracias al polimorfismo.

Vamos a por un ejemplo:

```
1 class Empleado:
2     def __init__(self, nombre, sueldoMensual):
3         self.nombre = nombre
4         self.sueldoMensual = sueldoMensual
5
6     def calcularSueldoAnual(self):
7         sueldo = 12 * self.sueldoMensual * (1 + 1/100)
8         print(f"El sueldo anual de {self.nombre}, empleado normal, es de {sueldo}€")
9
10 class Contable(Empleado):
11     def __init__(self, nombre, sueldoMensual):
12         super().__init__(nombre, sueldoMensual)
13
14     def calcularSueldoAnual(self):
15         sueldo = 12 * self.sueldoMensual * (1 + 4/100)
16         print(f"El sueldo anual de {self.nombre}, Contable, es de {sueldo}€")
17
18 class Publicista(Empleado):
19     def __init__(self, nombre, sueldoMensual):
20         super().__init__(nombre, sueldoMensual)
21
22     def calcularSueldoAnual(self):
23         sueldo = 12 * self.sueldoMensual * (1 + 5/100)
24         print(f"El sueldo anual de {self.nombre}, Publicista, es de {sueldo}€")
25
26 class Becario(Empleado):
27     def __init__(self, nombre, sueldoMensual):
28         super().__init__(nombre, sueldoMensual)
29
30     def calcularSueldoAnual(self):
31         sueldo = 12 * self.sueldoMensual
32         print(f"El sueldo anual de {self.nombre}, Becario, es de {sueldo}€")
33
34 empleados = [
35     Empleado("Juan", 1000),
36     Contable("Angela", 1100),
37     Publicista("Ryan", 1200),
38     Becario("Jose", 750)
39 ]
40
41 for empleado in empleados:
42     empleado.calcularSueldoAnual()
```

SuperClase Empleado

SubClase Contable

SubClase Publicista

SubClase Becario

En la línea 1 tenemos la clase Empleado es la superclase.

En la línea 6 tenemos el método de la clase Empleados.

En las líneas 10, 18 y 26 tenemos las subclases Contable, Publicista y Becario, con sus respectivos métodos, estos tienen el mismo nombre que el método de la SuperClase.

En la línea 34 tenemos una lista de empleados, esta lista realiza 4 instancias que se corresponden a las clases Empleado, Contable, Publicista y Becario, con su respectivo nombre y sueldo base.

En la línea 41 con un bucle for realizamos una iteración por las 4 instancias llamando el método .calcularSueldoAnual(), antes de llamar al método de la superclase si encuentra un método con el mismo nombre dentro de su clase, este es el que se llamará.

Este será el resultado al ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Polimorfismo.py"
El sueldo anual de Juan, empleado normal, es de 12120.0€
El sueldo anual de Angela, Contable, es de 13728.0€
El sueldo anual de Ryan, Publicista, es de 15120.0€
El sueldo anual de Jose, Becario, es de 9000€
PS C:\Users\pmver\Desktop\Ejer Python> █
```

17.- ABSTRACCIÓN en Python

La abstracción no es más que un mecanismo que nos permite crear una cosa abstracta, son clases nunca vamos a instanciar directamente, es decir yo tengo una clase abstracta que se llame Herramientas yo nunca voy a crear un objeto de la clase Herramientas, lo cual nos lleva a preguntar si yo tengo una clase que no puedo instanciar, ¿Para qué quiero esta clase?.

Queremos estas clases para utilizarlas de estructura principal de base para otras subclases que van a heredar de la clase abstracta.

Por ejemplo siguiendo con la clase Herramientas, una herramienta es un concepto muy genérico, yo puedo tener, un martillo, un destornillador, unos alicates, etc. Yo lo que puede hacer es crear mi clase Padre que sea Herramientas y hacerla abstracta. Luego crear otras subclases como las que conocemos que sean Martillo, Destornillador, Alicates, etc.

Todas las subclases van a heredar de la clase Herramientas y nosotros creamos crear un objeto crearemos un martillo, crearemos un destornillador, pero nunca crearemos una herramienta como tal porque no tiene mucho sentido.

Lo que sí tiene sentido es agrupar características comunes que van a tener muchas subclases en una superclase abstracta, de esta forma nos ahorramos mucho código y tener que reescribir muchos métodos.

Las clases abstractas como mínimo tienen que tener un método abstracto, esto nos lleva a preguntar que es un método abstracto, un método abstracto simplemente es un método que obligatoriamente vamos a tener que sobrescribir en las clases que hereden de la clase abstracta, es decir si la clase Herramientas tiene un método abstracto que se llame utilizar herramientas dentro de todas mis subclases voy que tener que redefinir (sobrescribir) este método en las subclases que heredan Herramientas.

```
8 from abc import ABC, abstractmethod
```

Con esta línea lo que estamos haciendo es importar una clase e importar un método que se llama abstractmethod del módulo ABC, esto nos permite utilizar clases que están en un archivo distinto al programa que estamos realizando.

En este ejemplo vamos a crear una clase abstracta llamada Personajes y de esta clase van a heredar la clase Mago como la clase Guerrero.

```
1 # CLASES ABSTRACTAS
2 # - NO LAS VAMOS A INSTANCIAR NUNCA DIRECTAMENTE.
3 # - CONTIENE POR LO MENOS UN MÉTODO ABSTRACTO.
4 # - LAS VAMOS A USAR DE BASE PARA SUBCLASES MÁS ESPECÍFICA.
5
6 # MÉTODOS ABSTRACTOS:
7 # - DEBEMOS SOBRESCRIBIRLOS EN CADA UNA DE LAS SUBCLASES.
8 from abc import ABC, abstractmethod
9
10 class Personajes(ABC):
```

La crear una clase y esta hereda de ABC, estamos realizando una clase abstracta.

```

10 class Personajes(ABC):
11     @abstractmethod # Decorador @abstractmethod: Contiene un conjunto de ficheros
12     def __init__(self, nombre):
13         self.nombre = nombre
14         self.nivel = 0
15         self.inventario = []
16         self.vida = 100

```

En la línea 11 el Decorador @abstractmethod lo que hace es envolver y modificar el comportamiento de la función que está decorando, este es un ejemplo de clase abstracta.

Ahora vamos a instanciar esta clase y observaremos que nos dará error.

```

17 pj = Personajes("Kike")

```

Ejecutamos:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Abstracion.py"

```

```

Traceback (most recent call last):
  File "c:\Users\pmver\Desktop\Ejer Python\Abstracion.py", line 17, in <module>
    pj = Personajes("Kike")
        ^^^^^^^^^^^^^^^^^^^
TypeError: Can't instantiate abstract class Personajes with abstract method __init__
PS C:\Users\pmver\Desktop\Ejer Python>

```

Ahora vamos a definir el método abstracto:

```

17     @abstractmethod
18     def atacar(self, objetivo):
19         pass

```

Ponemos pass porque cada clase tendrá su respectivo método atacar.

Vamos a crear otro método abstracto:

```

21     @abstractmethod
22     def getStatus(self):
23         print(f"Nombre: {self.nombre}. Nivel: {self.nivel}")

```

Para poder ver el nombre y su nivel. Lo podemos definir porque cuando la implementemos a los personajes este método si lo podremos utilizar.

```

26     def subirDeNivel(self):
27         self.nivel += 1

```

Este método no lo voy a hacer abstracto porque lo voy a utilizar tal como está, sin modificarlo.

```

29  ✓ |     def verInventario(self):
30      |         print(f"Inventario de {self.nombre}")
31  ✓ |         for objeto in self.inventario:
32      |             print(objeto)

```

Otro método que no hacemos abstracto.

Vamos a crear la clase Mago.

```

36  class Mago(Personajes):
37      def __init__(self, nombre):
38          super().__init__(nombre)
39          self.vida = 120
40          self.inteligencia = 95
41          self.inventario = ["Poción de Maná", "Grimorio"]

```

Ahora vamos a reescribir los métodos abstractos.

```

43      def getStatus(self):
44          print("Clase Mago")
45          super().getStatus()

```

Al llamar este método nos dirá el nombre y que soy de la clase Mago y además del mismo método de la clase abstracta nos dará el nombre y el nivel, de este modo estamos extendiendo el método getStatus(self).

Ahora vamos a crear el método vacío, recuerda que en la clase abstracta lo dejamos vacío, es porque cada personaje tiene su propio ataque.

```

47  ✓ |     def atacar(self, objetivo):
48      |         objetivo.vida -= self.inteligencia * 0.6
49      |         print(f"Vida actual del objetivo es: {objetivo.vida}")

```

Ya hemos redefinido los métodos abstractos, si queremos podemos hacer más.

Ahora vamos a repetir con el personaje Guerrero.

```

51  ✓ class Guerrero(Personajes):
52  ✓ |     def __init__(self, nombre):
53      |         super().__init__(nombre)
54      |         self.vida = 200
55      |         self.fuerza = 75
56      |         self.inventario = ["Poción de vida", "Maza", "Escudo"]
57
58  ✓ |     def getStatus(self):
59      |         print("Clase: Guerrero")
60      |         super().getStatus()
61

```

```

62  def atacar(self, objetivo):
63      objetivo.vida -= self.fuerza*0.85
64      print(f"Vida actual del objetivo: {objetivo.vida}")

```

Ahora vamos a instanciar un par de personajes:

```

66  guerrero = Guerrero("Kaladin")
67  mago = Mago("Yuno")

```

A cada personaje le vamos a llamar el método getStatus()

```

69  guerrero.getStatus()
70  mago.getStatus()

```

A cada personaje le vamos a llamar el método verInventario()

```

72  guerrero.verInventario()
73  mago.verInventario()

```

Ahora cada personaje tiene que atacar al otro.

```

75  mago.atacar(guerrero)
76  guerrero.atacar(mago)

```

Si ejecutamos este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

Clase: Guerrero
Nombre: Kaladin. Nivel: 0
Clase Mago
Nombre: Yuno. Nivel: 0

```

```

Inventario de Kaladin
Poción de vida
Maza
Escudo
Inventario de Yuno
Poción de Maná
Grimorio

```

```

Vida actual del objetivo es: 143.0
Vida actual del objetivo: 56.25

```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Código completo:

```
from abc import ABC, abstractmethod

class Personajes(ABC):
    @abstractmethod
    def __init__(self, nombre):
        self.nombre = nombre
        self.nivel = 0
        self.inventario = []

    @abstractmethod
    def atacar(self, objetivo):
        pass

    @abstractmethod
    def getStatus(self):
        print(f"Nombre: {self.nombre}. Nivel: {self.nivel}")

    def subirDeNivel(self):
        self.nivel += 1
        print(f"El nivel actual de {self.nombre} es {self.nivel}")

    def verInventario(self):
        print(f"Inventario de {self.nombre}: ")
        for objeto in self.inventario:
            print(objeto)

class Mago(Personajes):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.vida = 100
        self.inteligencia = 120
        self.inventario = ["Poción de maná", "Grimorio"]

    def getStatus(self):
        print("Clase: Mago")
        super().getStatus()

    def atacar(self, objetivo):
        objetivo.vida -= self.inteligencia*0.6
        print(f"Vida actual del objetivo: {objetivo.vida}")
```

```

class Guerrero(Personajes):
    def __init__(self, nombre):
        super().__init__(nombre)
        self.vida = 200
        self.fuerza = 75
        self.inventario = ["Poción de vida", "Maza", "Escudo"]

    def getStatus(self):
        print("Clase: Guerrero")
        super().getStatus()

    def atacar(self, objetivo):
        objetivo.vida -= self.fuerza*0.85
        print(f"Vida actual del objetivo: {objetivo.vida}")

guerrero = Guerrero("Kaladin")
mago = Mago("Yuno")

guerrero.getStatus()
mago.getStatus()

guerrero.verInventario()
mago.verInventario()

mago.atacar(guerrero)
guerrero.atacar(mago)

```

18.- Definir Clases con Ejemplos

Un objeto es un elemento va a estar compuesto por dos cosas:

Los atributos: Son como unas propiedades o características que definen al objeto o definen su estado, de esta forma si estamos intentando programar a una persona podemos tener varios atributos, nombre, dni, altura, edad, etc.

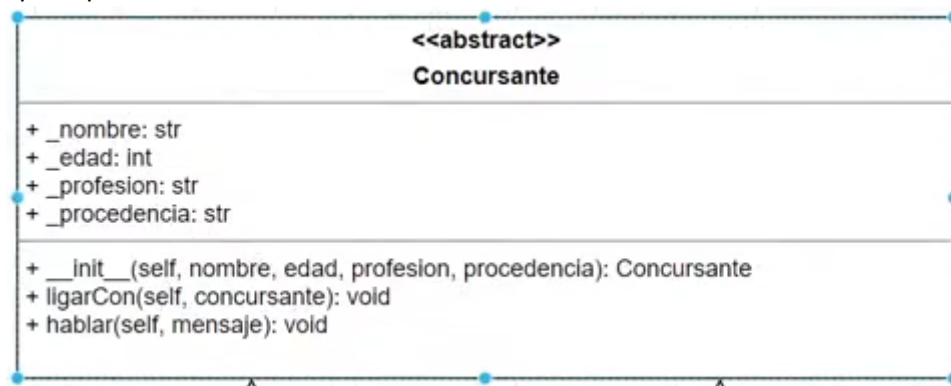
Los métodos: Son funcionalidades, labores que pueden desempeñar dichos objetos.

Las clases: Agrupan objetos que tienen métodos y atributos completamente iguales.

Todos los objetos que tienen los mismos atributos y métodos van a pertenecer a la misma clase.

Para este proyecto vamos a realizar un programa que simula la “Isla de las tentaciones”.

La clase principal son los concursantes de la isla de las tentaciones.



Independientemente sean solteros o tentados van a tener un nombre, edad, profesión y procedencia.

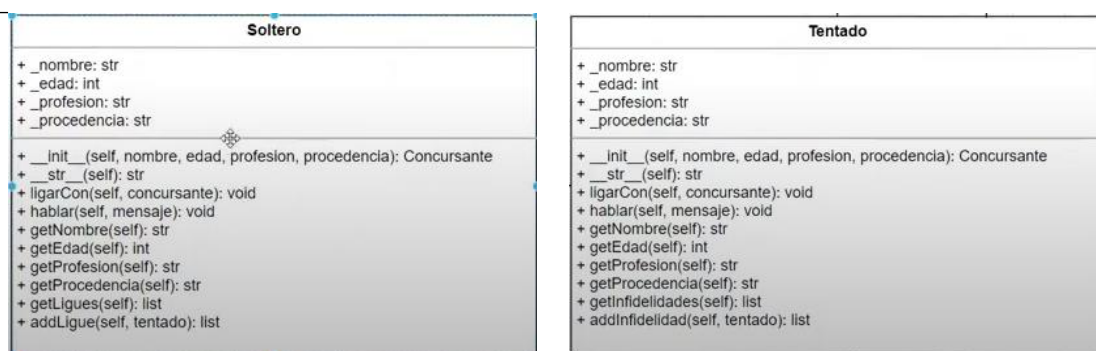
Y además podrán desempeñar ciertas tareas como ligar, hablar, etc.

Como todos nuestros concursantes van a compartir estas características van a pertenecer a la misma clase.

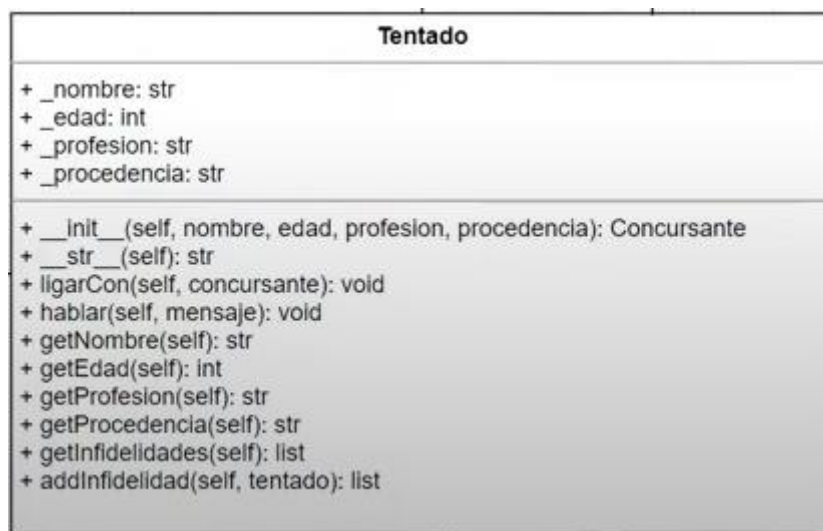
Como símil podemos ver las clases como plantillas y cuando creamos un objeto simplemente estamos rellenando estas plantillas con ciertos valores.

La herencia: Es un mecanismo que nos permite traspasar todas las propiedades y métodos de una clase a otra clase.

Usaremos la herencia para definir las clases Soltero y Tentado:



Tanto la clase Soltero como Tentado tienen que heredar de Concursante.



La clase Concursante la vamos a llamar clasePadre y las clases Soltero y Tentado serán las subclases o clases hijos.

La potencia de la herencia que además las clases Soltero y Tentado tengan las mismas características, también nos permite extender estos atributos y características.

La clase Padre tiene métodos que también tiene las clase hijos, además de otros métodos.

Polimorfismo: Viene como una consecuencia de la herencia, métodos que tenemos tanto el Soltero como en Tentado que tengan el mismo nombre, cuando sean llamadas realicen cosas distintas.

Por ejemplo con el método legarCon(sedlf, concursante) si lo llamados desde Tentado diremos que se ha hecho una infidelidad, en cambio si lo llamamos desde la clase Soltero diremos que se ha realizado una conquista.

El Polimorfismo nos permite dar una definición distinta al mismo método en clases distintas.

Abstracción: En la clase Concursante la vamos a definir como Abstracta. Significa que es una clase nunca vamos a instancia y que contiene como mínimo un método abstracto.

¿Qué significa que no vallamos a instanciarla? Significa que nunca vamos a crear un objeto de dicha clase.

Estas clases nos sirven con estructura para crear otras clases pero nunca vamos a crear objetos.

Nos permite crear métodos abstractos ya que cada clase tendrá su propio método con el mismo nombre y con acciones totalmente distintas (Polimorfismo). El método lo dejaremos en blanco.

Encapsulamiento: Teniendo en cuenta que en Python no existe el encapsulamiento, pero si en otros lenguajes como C++, JavaScript, etc.

En capítulos anteriores comentamos que hay una convención para poder realizar una simulación de que estamos trabajando en el encapsulamiento.

El encapsulamiento sirve para regular el acceso a los atributos y a los métodos de nuestras clases.

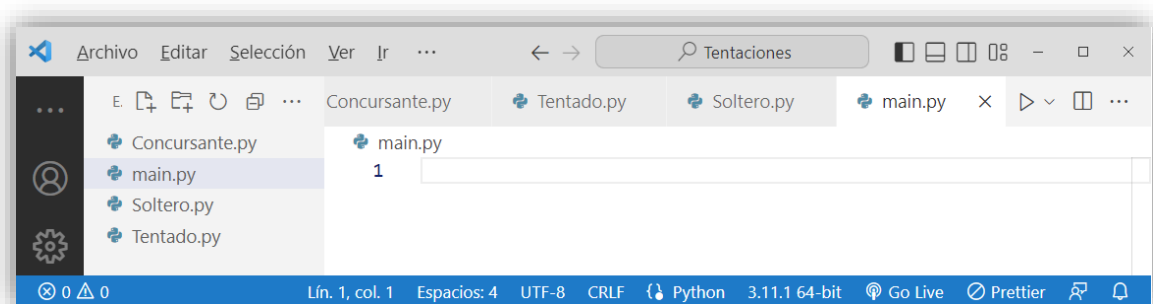
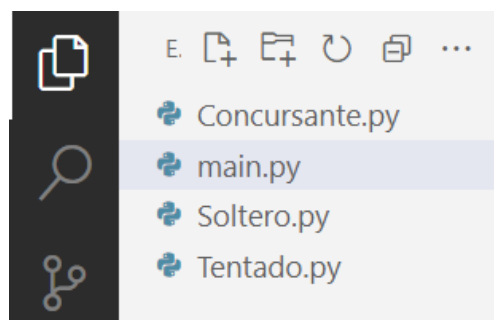
Por ejemplo creamos un objeto de la clase Soltera llamada Carla. Su nombre no va a cambiar y lo que no quiero que desde fuera de la clase Solteros se pueda modificar el nombre, para ello puedo regular su acceso y crearlo como privado. En cambio la edad si que puede cambiar pues lo podemos hacer público, siendo público desde cualquier instancia de Soltero voy a poder cambiar la edad, si es privado solo se podrán cambiar desde los métodos que existan en la propia clase.

Existe una tercera forma de regular el acceso protegido funciona igual que el privado, es decir solo puedo modificar los atributos protegidos dentro de los métodos de la misma clase, pero también los puedo modificar dentro de los métodos de las subclases.

Vamos con el ejemplo:

Para este proyecto vamos a crear una nueva carpeta.

Vamos a crear cada clase en un fichero distinto, los ficheros que contienen clase la primera letra irá en mayúsculas y el del propiamente para el código en minúsculas.



Vamos a comentar por el documento Concursante.py que es donde irá la superclase.

```
from abc import ABCMeta # Es una metaclass que nos va
                        # permitir hacer nuestras clases
                        # Abstractas.
from abc import abstractmethod # Decorador abstracto

class Concursante(metaclass=ABCMeta):
    @abstractmethod
    def __init__(self, nombre, edad, profesion, procedencia):
        self._nombre = nombre
        self._edad = edad
        self._profesion = profesion
        self._procedencia = procedencia
        # self.atributo -> publicco
        # self._atributo -> protegido
        # self.__atributo -> privado

    def setEdad(self, nuevaEdad):
        if type(nuevaEdad in int):
            self._edad = nuevaEdad

    def presentarse(self):
        print(self)

    @abstractmethod
    def ligarCon(self, consursante): ... # Para dejarlo en
                                         # blanco ponemos 3 puntos.

    @abstractmethod
    def hablar(self, mensaje): ...

    # GETTERS AND SETTERS
    def presentarse(self):
        print(self)

    def getNombre(self):
        return self._nombre

    def getEdad(self):
        return self._edad

    def getProfesion(self):
        return self._profesion

    def getProcedencia(self):
        return self._procedencia
```

Ahora vamos a Soltero.py

```
# Del fichero Concursante vamos a importar la clase Concursante
from Concursante import Concursante
import time

class Soltero(Concursante):
    def __init__(self, nombre, edad, profesion, procedencia):
        super().__init__(nombre, edad, profesion, procedencia)
        self._ligues = []

    # Métodos mágicos __str__, se llama automáticamente
    # hacemos un print de un objeto de la clase, este se
    # imprimira con el formato expecificado
    def __str__(self):
        return f"\n\n{self._nombre} ha venido a enamorarse en la isla de
las tentaciones. Tiene {self._edad} edad, viene de {self._procedencia} y
su trabajo es {self._profesion}\n"

    def addLigue(self, tentado):
        self._ligues.append(tentado)

    def ligarCon(self, tentado):
        self.addLigue(tentado)
        print(f"{self._nombre} ha conectado completamente con
{tentado._nombre} Esta es su {len(self._ligues)} a conquista")
        time.sleep(1) # Para que se detenga la ejecución 1 segundo.

    def hablar(self, mensaje):
        target = " "
        if len(self._ligues) > 0:
            target = f" (que ha conquistado a {self._ligues[-1]._nombre})"
        print(f"\n[{self._nombre.upper()}]{target}: {mensaje}\n")
        time.sleep(1)
```

Ahora vamos a Tentado.py

```
from Concursante import Concursante
import time

class Tentado(Concursante):
    def __init__(self, nombre, edad, profesion, procedencia):
        super().__init__(nombre, edad, profesion, procedencia)
        self._inifidelidades = []
        self._pareja = None

    def __str__(self):
        return f"\n{self._nombre} es la pareja de {self._pareja._nombre}.  
tiene {self._edad} años, viene de {self._procedencia} y tu trabajo es  
{self._profesion}\n"

    def setPareja(self, pareja):
        self._pareja = pareja

    def addInfidelidad(self, soltero):
        self._inifidelidades.append(soltero)

    def ligarCon(self, soltero):
        self.addInfidelidad(soltero)
        soltero.addLigue(self)
        print(f"¡NIIIIII NOOOOO NIIIIII NOOOOO NOOO! SUENA LA  
ALARMA.{self._nombre} ha sido infiel a {self._pareja._nombre}")
        time.sleep(1)

    def hablar(self, mensaje):
        print(f"\n[{self._nombre.upper()}] (pareja de  
{self._pareja._nombre}): {mensaje}\n")
```

Ahora vamos a main.py

```
from Soltero import Soltero
from Tentado import Tentado
import time, os

def emparejar(listaParejas):
    for novio, novia in listaParejas:
        novio.setPareja(novia)
        novia.setPareja(novio)

def presentaciones(parejas, solteros):
    print("[SANDRA]: ADELANTE LAS PAREJAS")
    for novio, novia in parejas:
        novio.presentarse()
        time.sleep(1)
        novia.presentarse()
        time.sleep(1)
    print("[SANDRA]: Adelante los tentadores y tentadoras:")
    for soltero in solteros:
        time.sleep(1)
        soltero.presentarse()
        print("")

if __name__ == "__main__": # Hilo principal de la ejecución

    diego = Tentado("Diego", 26, "Boxeador y James Lover", "León")
    lola = Tentado("Lola", 26, "llorar", "León")

    carla = Soltero("Carla", 28, "Influencer", "Barcelona")
    jenifer = Soltero("Jenifer", 26, "modelo", "Tenerife")
    simone = Soltero("Sione", 26, "empresario", "Milán")

    jesus = Tentado("Jesús", 26, "desconocido", "Sevilla")
    marina = Tentado("Marina", 26, "desconocida", "Sevilla")

    isaac = Soltero("Isaac", 25, "Influencer", "Barcelona")
    stefany = Soltero("Stefany", 25, "Psicóloga", "Barcelona")
    lara = Soltero("Lara", 24, "modelo", "Pontevedra")

    parejas = [(diego, lola), (jesus, marina)]
    solteros = [carla, jenifer, simone, isaac, stefany, lara]

    emparejar(parejas)
    presentaciones(parejas, solteros)
```



```
carla.hablar("Lo que ha hecho diego me ha dolido, se que no soy su
novia pero no me lo esperaba.")

print("\n[SANDRA BERNEDA]: Lola ¿Qué te han parecido estas
imágenes?.\n")
lola.hablar("Al perro me lo quedo yo Sandra. Es que parece una
persona.")

print("\n[SANDRA BERNEDA]: Marina es tu turno. Tenemos imágenes para
ti.\n")
jesus.ligarCon(stefany)
jesus.hablar("Es que Stefany físicamente es un 10")
jesus.ligarCon(lara)
jesus.hablar("Uf es que lara psicológicamente me gusta más")

marina.hablar("Esto es tonto")
print("FINAL DE LA HISTORIA")
```

Si ejecutamos este será el resultado:

[SANDRA]: ADELANTE LAS PAREJAS

Diego es la pareja de Lola. tiene 26 años, viene de León y tu trabajo es Boxeador y James Lover

Lola es la pareja de Diego. tiene 26 años, viene de León y tu trabajo es llorar

Jesús es la pareja de Marina. tiene 26 años, viene de Sevilla y tu trabajo es desconocido

Marina es la pareja de Jesús. tiene 26 años, viene de Sevilla y tu trabajo es desconocida

[SANDRA]: Adelante los tentadores y tentadoras:

Carla ha venido a enamorarse en la isla de las tentaciones. Tiene 28 edad, viene de Barcelona y su trabajo es Influencer

Jenifer ha venido a enamorarse en la isla de las tentaciones. Tiene 26 edad, viene de Tenerife y su trabajo es modelo

Sione ha venido a enamorarse en la isla de las tentaciones. Tiene 26 edad, viene de Milán y su trabajo es empresario

Isaac ha venido a enamorarse en la isla de las tentaciones. Tiene 25 edad, viene de Barcelona y su trabajo es Influencer

Stefany ha venido a enamorarse en la isla de las tentaciones. Tiene 25 edad, viene de Barcelona y su trabajo es Psicóloga

Lara ha venido a enamorarse en la isla de las tentaciones. Tiene 24 edad, viene de Pontevedra y su trabajo es modelo

[SYSTEM]: Pulsa 'enter' cuando quieras continuar con las historia:

[SANDRA BERNEDA]: ¡HOLAAA! Ya ha pasado una semana ¿Sabéis por qué estoy aquí?

[DIEGO] (pareja de Lola): ¿Has venido a ver Vikingos con nosotros?

[SANDRA BERNEDA]: No Diego. He venido porque hoy tenemos hoguera.

[JESÚS] (pareja de Marina): AI NO

[SANDRA BERNEDA]: Que de comienzo la hoguera de los chicos.

[SANDRA BERNEDA]: Diego empezamos contigo. Tenemos imágenes para ti.

¿NIIIIII NOOOOO NIIIIII NOOOOO NOOO! SUENA LA ALARMA.Lola ha sido infiel a Diego

[SIONE] (que ha conquistado a Lola) : Io sonno molto contento

[DIEGO] (pareja de Lola): A mi el choped ya no me gusta, prefiero el jamón de bellota

[SANDRA BERNEDA]: Jesús es tu turno. Tenemos un par de películas para ti.

¿NIIIIII NOOOOO NIIIIII NOOOOO NOOO! SUENA LA ALARMA.Marina ha sido infiel a Jesús

[JESÚS] (pareja de Marina): Aiii cuando la vea su madre Sandra. A ese su padre no le deja entrar en casa

[illegible]

[SYSTEM]: Pulsa 'enter' cuando quieras continuar con la hoguera de las chicas:

[SANDRA BERNEDA]: Que dé comienzo la hoguera de las chicas.

[SANDRA BERNEDA]: Lola empezamos contigo. Tenemos imágenes para ti.

{NIIIIII NOOOOO NIIIIII NOOOOO NOOO! SUENA LA ALARMA.Diego ha sido infiel a
Lola

[CARLA] (que ha conquistado a Diego) : Soy auténtico Jamón de bellota

{NIIIIII NOOOOO NIIIIII NOOOOO NOOO! SUENA LA ALARMA.Diego ha sido infiel a
Lola

[JENIFER] (que ha conquistado a Diego) : Después de esto, no voy a aparecer más en el programa

[CARLA] (que ha conquistado a Diego) : Lo que ha hecho diego me ha
dolido, se que no soy su novia pero no me lo esperaba.

[SANDRA BERNEDA]: Lola ¿Qué te han parecido estas imágenes?.

[LOLA] (pareja de Diego): Al perro me lo quedo yo Sandra. Es que parece una persona.

[SANDRA BERNEDA]: Marina es tu turno. Tenemos imágenes para ti.

{NIIIIII NOOOOO NIIIIII NOOOOO N000! SUENA LA ALARMA.Jesús ha sido infiel a Marina

[JESÚS] (pareja de Marina): Es que Stefany físicamente es un 10

¿NIIIIII NOOOOO NIIIIII NOOOOO NOOO! SUENA LA ALARMA.Jesús ha sido infiel a Marina

[JESÚS] (pareja de Marina): Uf es que lara psicológicamente me gusta más

[MARINA] (pareja de Jesús): Esto es tonto

FINAL DE LA HISTORIA

19.- Excepciones en Python

```
# Capítulo 19: EXCEPCIONES
# Las excepciones son errores que ocurren durante la ejecución del
# programa. Estos errores surgen a pesar de que la sintaxis es correcta.
# Ejemplos:
# - Acceder a una posición de una lista superior a la longitud de esta.
# - Intentar abrir un fichero no existente.
# - Convertir "skdjajf" a int.
# IMPORTANTE: Gestionar las excepciones nos permiten que el código se siga
# ejecutando a pesar de que ocurran errores.
```

Vamos a provocar un error dividiendo un número entre 0:

```
1 a = 10/0
```

Vamos a ejecutar:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Excepciones.py"

Traceback (most recent call last):
  File "c:\Users\pmver\Desktop\Ejer Python\Excepciones.py", line 1, in <module>
    a = 10/0
    ~~~~
ZeroDivisionError: division by zero
PS C:\Users\pmver\Desktop\Ejer Python>
```

El nombre de la excepción es ZeroDivisionError.

```
1 a = 10/0
2 print("Hola que tal")
```

Agregamos una segunda línea de código.

Al ejecutar de nuevo salta la excepción ZeroDivisionError y el programa se interrumpe, ya no vemos el mensaje "Hola que tal".

Vamos con otro ejemplo:

```
1 def division(a,b):
2     try:
3         resultado = a/b
4         print(resultado)
5     except ZeroDivisionError:
6         print("No se puede dividir por cero")
```

Línea 1 creamos una función llamada división a la que le pasaremos dos argumentos.

Línea 2 try: vamos a probar el código haber que pasa, en caso de error las líneas 3 y 4 no se ejecutarían.

Línea 5 con las sentencia except le decimos que si la excepción del error es ZeroDivisionError:

Línea 6 en caso afirmativo de la línea anterior, mostrará el siguiente mensaje “No se puede dividir por cero”.

Vamos a llamar la función un par de veces:

```
1  def division(a,b):
2      try:
3          resultado = a/b
4          print(resultado)
5      except ZeroDivisionError:
6          print("No se puede dividir por cero")
7
8  division(5,5)
9  division(5,0)
10
11 print("Fin del programa")
```

Vamos a ejecutar, la primera vez que llamamos a la función no se produciría en error ya que estamos dividiendo entre 5, en cambio en la segunda vez que llamamos a la función estamos realizando un división entre 0, esta tiene que generar un error, además agregamos una línea al final para indicarnos que hemos llegado al final del programa, para comprobar como el programa no se interrumpe.

Vamos a ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Excepciones.py"
```

```
1.0
```

```
No se puede dividir por cero
```

```
Fin del programa
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Nos muestra el mensaje que nosotros deseamos.

```

1  # Apartado 2: Gestión de distintos tipos de excepciones
2  frutas = ["0-Plátano", "1-Manzana", "2-Pomelo", "3-Melocotón"]
3
4  def elegirFruta(listaFrutas):
5      try:
6          print(listaFrutas)
7          index = int(input("Elige una fruta (pon el número): "))
8          print(f"Tu fruta favorita es {listaFrutas[index]}")
9      except IndexError:
10         print(f"Índice incorrecto entre 0 y {len(listaFrutas)-1}")
11     except ValueError:
12         print("Tienes que poner un número entero")

```

En este ejemplo vamos a crear un función donde tenemos que pasar como argumento una lista de frutas, haremos que el usuario de la lista seleccione al que desee.

Pueden pasar dos tipos de errores que el usuario ponga un número de fruta que no exista por ejemplo 6, como podréis observar la lista llega hasta el número 3.

Otro error que el usuario el lugar de un número ponga un texto esto generaría otro error, como queremos gestionar los dos errores vamos a poner dos excepciones.

Vamos a llamar a la función primero con la fruta 2, a continuación la la fruta 6 (recuerda que esta no existe y por último en lugar de un numero pondremos un texto.

```

14  elegirFruta(frutas)
15  elegirFruta(frutas)
16  elegirFruta(frutas)

```

Vamos a ejecutar:

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Excepciones.py"

['0-Plátano', '1-Manzana', '2-Pomelo', '3-Melocotón']
Elige una fruta (pon el número): 2
Tu fruta favorita es 2-Pomelo
['0-Plátano', '1-Manzana', '2-Pomelo', '3-Melocotón']
Elige una fruta (pon el número): 6
Índice incorrecto entre 0 y 3
['0-Plátano', '1-Manzana', '2-Pomelo', '3-Melocotón']
Elige una fruta (pon el número): Patata
Tienes que poner un número entero
PS C:\Users\pmver\Desktop\Ejer Python>

```

El programa no se ha interrumpido.

Vamos con el siguiente ejemplo.

```

1  # Apartado 3: Exception Exception..
2  # Las excepciones son objetos que heredan de la clase Exception.
3
4  frutas = ["0-Plátano", "1-Manzana", "2-Pomelo", "3-Melocotón"]
5
6  def elegirFruta(listaFrutas):
7      try:
8          print(listaFrutas)
9          index = int(input("Elige una fruta (pon el número): "))
10         print(f"Tu fruta favorita es {listaFrutas[index]}")
11     except Exception:
12         print("Ha ocurrido un error, algo ha salido mal")
13
14     elegirFruta(frutas)
15     elegirFruta(frutas)

```

En este caso queremos controlar la excepción sea la que sea sin tener en cuenta el tipo de excepción que es.

Tanto si ponemos un valor de fruta mayor de 3 o un texto en lugar de un número, siempre vamos a obtener el mismo mensaje de error.

Vamos a ejecutar.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Excepciones.py"
```

```
['0-Plátano', '1-Manzana', '2-Pomelo', '3-Melocotón']
Elige una fruta (pon el número): 6
Ha ocurrido un error, algo ha salido mal
['0-Plátano', '1-Manzana', '2-Pomelo', '3-Melocotón']
Elige una fruta (pon el número): Patata
Ha ocurrido un error, algo ha salido mal
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Si ponemos una Excepción que no es la que provoca el error, el programa se detendrá.

```

1  # Apartado 3: Exception Exception..
2  # Las excepciones son objetos que heredan de la clase Exception.
3
4  frutas = ["0-Plátano", "1-Manzana", "2-Pomelo", "3-Melocotón"]
5
6  def elegirFruta(listaFrutas):
7      try:
8          print(listaFrutas)
9          index = int(input("Elige una fruta (pon el número): "))
10         print(f"Tu fruta favorita es {listaFrutas[index]}")
11     except ZeroDivisionError: ←
12         print("Ha ocurrido un error, algo ha salido mal")

```

```

13
14  elegirFruta(frutas)

```

Vamos a ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Excepciones.py"

```

```

['0-Plátano', '1-Manzana', '2-Pomelo', '3-Melocotón']
Elige una fruta (pon el número): 6
Traceback (most recent call last):
  File "c:\Users\pmver\Desktop\Ejer Python\Excepciones.py", line 14, in <module>
    >
    elegirFruta(frutas)
  File "c:\Users\pmver\Desktop\Ejer Python\Excepciones.py", line 10, in elegirFruta
    print(f"Tu fruta favorita es {listaFrutas[index]}")
    ~~~~~^~~~~~

```

IndexError: list index out of range

```

PS C:\Users\pmver\Desktop\Ejer Python> █

```

Y nos muestra el tipo de error por si lo queremos controlar con una excepción.

Podemos renombrar las excepciones:

```

4  frutas = ["0-Plátano", "1-Manzana", "2-Pomelo", "3-Melocotón"]
5
6  def elegirFruta(listaFrutas):
7      try:
8          print(listaFrutas)
9          index = int(input("Elige una fruta (pon el número): "))
10         print(f"Tu fruta favorita es {listaFrutas[index]}")
11     except Exception as errorRandom:
12         print(errorRandom)
13
14  elegirFruta(frutas)

```

Podemos pasar el error a una variable errorRandom y a continuación mostrar el contenido de la variable.

Vamos a ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Excepciones.py"

```

```

['0-Plátano', '1-Manzana', '2-Pomelo', '3-Melocotón']
Elige una fruta (pon el número): 6

```

list index out of range

```

PS C:\Users\pmver\Desktop\Ejer Python> █

```

Nos indica el tipo de error.

Ahora vamos a introducir un texto.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Excepciones.py"

['0-Plátano', '1-Manzana', '2-Pomelo', '3-Melocotón']
Elige una fruta (pon el número): Patata
invalid literal for int() with base 10: 'Patata'
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Otro ejemplo:

```
3 import logging
4 frutas = ["0-Plátano", "1-Manzana", "2-Pomelo", "3-Melocotón"]
5
6 def elegirFruta(listaFrutas):
7     try:
8         print(listaFrutas)
9         index = int(input("Elige una fruta (pon el número): "))
10        print(f"Tu fruta favorita es {listaFrutas[index]}")
11    except Exception as errorRandom:
12        logging.exception("El error es el siguiente: ")
13
14 elegirFruta(frutas)
```

Vamos a importar el módulo logging y en la línea 12 hacemos la siguiente excepción.

Vamos a ejecutar y este será el resultado:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Excepciones.py"

['0-Plátano', '1-Manzana', '2-Pomelo', '3-Melocotón']
Elige una fruta (pon el número): 6
ERROR:root:El error es el siguiente:
Traceback (most recent call last):
  File "c:\Users\pmver\Desktop\Ejer Python\Excepciones.py", line 10, in elegirFruta
    print(f"Tu fruta favorita es {listaFrutas[index]}")
                                ~~~~~^~~~~~
IndexError: list index out of range
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Nos da toda esta información.

Gracias a la pila de errores es decir una excepción puede llamar a otra excepción y así poder que como han ido sucediendo los errores.

Vamos por último ejemplo:

```

1  # Apartado 4: Else, Finally, Raise
2  # Vamos a sumar los números que nos pase el usuario separados por espacios:
3
4  while True:
5      try:
6          total = 0
7          sumandos = input("Pon números separados por espacio: ")
8          sumandos = sumandos.split()
9          for num in sumandos:
10             if num.isnumeric():
11                 total += float(num)
12             else:
13                 # Lanzar un excepción
14                 raise ValueError("El valor no es un número")
15     except ValueError:
16         print("Los datos son incorrectos")
17         print("Vuelve a introducir los números")
18     else:
19         # Se ejecuta cuando el bloque try se ha ejecutado sin ninguna excepción.
20         print(f"El valor de la suma es {total}")
21         break
22     finally:
23         # Siempre ejecutará el bloque de código
24         print("Ha terminado la iteración")

```

Vamos a ejecutar poniendo correctamente los números.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Excepciones.py"

```

```

Pon números separados por espacio: 1 2 3 4 5 6 7 8 9 10
El valor de la suma es 55.0
Ha terminado la iteración
PS C:\Users\pmver\Desktop\Ejer Python> █

```

Ahora vamos a introducir una palabra por error.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Excepciones.py"

```

```

Pon números separados por espacio: 1 2 3 4 5 seis 7 8 9 10
Los datos son incorrectos
Vuelve a introducir los números
Ha terminado la iteración
Pon números separados por espacio: 1 2 3 4
El valor de la suma es 10.0
Ha terminado la iteración
PS C:\Users\pmver\Desktop\Ejer Python> █

```

20.- Funciones LAMBDA

Son funciones anónimas (no tienen nombre):

- Permiten múltiples argumentos.
- Solo puede tener una expresión.
- Estructura: lambda arg1, arg2 : expresión.

```
1  # Apartdo 1: Definir una función lambda
2
3  suma = lambda a, b: a+b
4  print(suma(5,5))
5  print(suma(3,2))
```

Vamos a ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Lambda.py"
10
5
PS C:\Users\pmver\Desktop\Ejer Python>
```

Como podréis ver los resultados será 10 y 5.

Otro ejemplo:

```
1  # Apartdo 1: Definir una función lambda
2
3  saludar = lambda nombre: print(f"Hola {nombre}!!")
4
5  saludar("Juan")
6  saludar("Ana")
```

Este será el resultado cuando ejecutemos:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Lambda.py"
Hola Juan!!
Hola Ana!!
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Estas funciones lambda las podemos asignar a funciones.

```

1  # Apartdo 2: Llamar fnciones dentro de una lambda
2  # Dentro de la exprtesión de una función lambda podemos
3  # llamar otras funciones:
4
5  maximo = lambda a, b, c: f"El máximo entre {a}, {b} y {c} es {max(a, b, c)}"
6  print(maximo(3, 5, 4))
7  print(maximo(10, 99, -25))

```

Vamos a ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Lambda.py"
El máximo entre 3, 5 y 4 es 5
El máximo entre 10, 99 y -25 es 99
PS C:\Users\pmver\Desktop\Ejer Python> 

```

```

# Apartdo 3: Funciones lambda dentro de funciones
# Podemos definir funciones lambda dentro de funciones
# convencionales.
#
# Esto nos permite generar funciones lambda con
# distintos parámetros.

```

Imaginaros al siguiente situación:

Yo tengo un nombre y quiero poner un prefijo, el prefijo puede ser Mr, Sr y Miss.

```

1  def ponerPrefijo(prefijo):
2      return lambda nombre: f"{prefijo} {nombre}"
3
4  addMr = ponerPrefijo("Mr")
5  addSr = ponerPrefijo("Sr")
6  addMiss = ponerPrefijo("Miss")
7
8  print(addMr("Juan"))
9  print(addSr("Julian"))
10 print(addMiss("Nerea"))

```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Lambda.py"

```

```

Mr Juan
Sr Julian
Miss Nerea

```

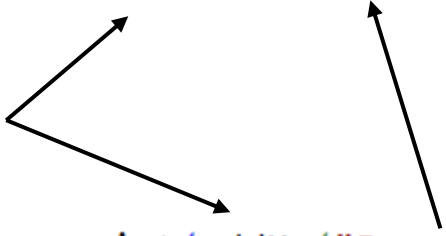
```

PS C:\Users\pmver\Desktop\Ejer Python> 

```

¿Qué es lo que ocurre? Cuando utilizamos la variable `addMr` le estamos diciendo que vamos a utilizar el prefijo "Mr", según la línea 4 además del nombre que agregamos como parámetro.

```
def ponerPrefijo(prefijo):  
    return lambda nombre: f"{prefijo} {nombre}"  
  
addMr = ponerPrefijo("Mr")  
  
print(addMr("Juan"))
```



Vamos a ver otro ejemplo:

```
1 def elevarA(exponente):  
2     return lambda base: base**exponente  
3  
4     elevarCuadrado = elevarA(2)  
5     elevarCubo = elevarA(3)  
6  
7     print(elevarCuadrado(3))  
8     print(elevarCubo(2))
```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/  
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/Lambda.py"
```

9
8

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

21.- Usar las Funciones MAP y FILTER en Python

Capítulo 21: Funciones MPA y FILTER

Son funciones que sirven para trabajar con objetos iterables.

Vienen incorporadas por defecto en Python.

Apartado 1: Función filter

filter(function, iterable). Devuelve un iterador

con los valores del iterable que cumplan la función.

```
1  emails = [  
2      'juan@hotmail.com',  
3      'javier@gmail.com',  
4      'mandarinas',  
5      'wikipedia.com'  
6  ]  
7  
8  for email in emails:  
9      if '@' in email:  
10         print(f"El email {email} es válido")  
11     else:  
12         print(f"El email {email} es inválido")
```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/map_filter.py"  
El email juan@hotmail.com es válido  
El email javier@gmail.com es válido  
El email mandarinas es inválido  
El email wikipedia.com es inválido  
PS C:\Users\pmver\Desktop\Ejer Python>
```

Seguimos con este ejemplo:

```
1  ✓ emails = [  
2      'juan@hotmail.com',  
3      'javier@gmail.com',  
4      'mandarinas',  
5      'wikipedia.com'  
6  ]
```

```

7
8 emailsValidos = []
9 for email in emails:
10     if '@' in email:
11         print(f"El email {email} es válido")
12         emailsValidos.append(email)
13     else:
14         print(f"El email {email} es inválido")
15
16 print(emailsValidos)

```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/map_filter.py"
El email juan@hotmail.com es válido
El email javier@gmail.com es válido
El email mandarinas es inválido
El email wikipedia.com es inválido
['juan@hotmail.com', 'javier@gmail.com']
PS C:\Users\pmver\Desktop\Ejer Python>

```

Ahora lo vamos a realizar utilizando la función filter.

```

1 emails = [
2     'juan@hotmail.com',
3     'javier@gmail.com',
4     'mandarinas',
5     'wikipedia.com'
6 ]
7
8 def evaluarEmail(email):
9     if '@' in email:
10         return True
11     else:
12         return False
13
14 emailsValidos = filter(evaluarEmail, emails)
15
16 # Al ser un objeto de tipo filter es un iterador.
17 # tenemos que usar next.

```

```

18  ✓ for i in range(2):
19      |     print(next(emailsValidos))

```

Como sabemos en este ejemplo que solo dos cumplen la condición hacemos un bucle de 2 iteraciones.

Este será el resultado:

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

StopIteration
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/map_filter.py"
juan@hotmail.com
javier@gmail.com
PS C:\Users\pmver\Desktop\Ejer Python> 

```

Si queremos leer más elementos de los que hay.

```

18  ✓ for i in range(3):
19      |     print(next(emailsValidos))

```

Este será el resultado:

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/map_filter.py"
juan@hotmail.com
javier@gmail.com
Traceback (most recent call last):
  File "c:\Users\pmver\Desktop\Ejer Python\map_filter.py", line 19, in <module>
    print(next(emailsValidos))
          ^^^^^^^^^^^^^^^^^^^^^
StopIteration
PS C:\Users\pmver\Desktop\Ejer Python> 

```

Estamos intentando leer más datos de los que hay.

Para conseguir resultados más rápidos vamos a convertir un iterador en una lista.

```

8  def evaluarEmail(email):
9      |     if '@' in email:
10         |         return True
11         |     else:
12         |         return False
13
14  emailsValidos = list(filter(evaluarEmail, emails))
15
16  print(emailsValidos)

```

Este será el resultado:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/map_filter.py"
['juan@hotmail.com', 'javier@gmail.com']
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Ahora vamos a simplificar la función evaluarEmail() y verás que funciona igual.

```
8  def evaluarEmail(email):
9      return '@' in email
```

Si ejecutas podrás comprobar que el resultado es el mismo.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/map_filter.py"
['juan@hotmail.com', 'javier@gmail.com']
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Os recuerdo que esto es una función lambda.

A continuación vamos a comentar esta función y modificar la siguiente.

```
8  # def evaluarEmail(email):
9  #      return '@' in email
10
11  emailsValidos = list(filter(lambda email: '@' in email, emails))
12  print(emailsValidos)
```

Este será el resultado:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/map_filter.py"
['juan@hotmail.com', 'javier@gmail.com']
PS C:\Users\pmver\Desktop\Ejer Python> █
```

La función map lo que hace es ejecutar la función usando como parámetros cada elemento del iterable, es decir aquí ya no estoy evaluando la condición, simplemente estoy cogiendo el valor iterable, la paso por una función.

```
1  # Apartado 2: Función map
2  # map(function, iterable)
3  # Ejecuta la función usando com parámetro
4  # cada elemento del iterable.
5
```

```
-
6  # Creamos una tupla
7  palabras = ('dale', 'un', 'buen', 'like', 'y', 'suscribete')
8  longitudes = list(map(lambda palabra: len(palabra), palabras))
9  print(longitudes)
```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/map_filter.py"
```

```
[4, 2, 4, 4, 1, 10]
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Vamos a ver otro ejemplo:

```
1  import math # Nos sirve para ejecutar funciones matemáticas.
2
3  eje_x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4  eje_y = [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
5  eje_z = list(map(lambda x, y: round(math.cos(x) + math.exp(y), 2), eje_x, eje_y))
6  print(eje_z)
```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/map_filter.py"
```

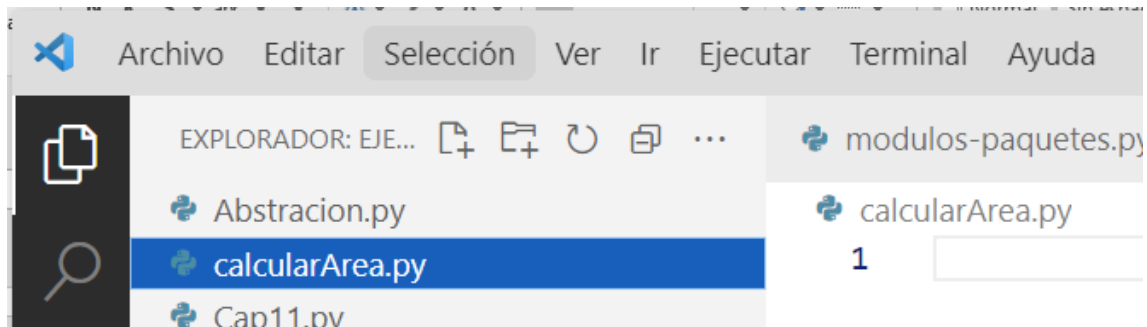
```
[21.09, 55.14, 148.0, 402.44, 1095.98, 2981.24, 8104.04, 22027.22, 59874.0, 162753.88, 442412.55]
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

22.- ¿Qué son los MÓDULOS y PAQUETES en Python?

```
# Capítulo 22: Módulos y Paquetes
# Apartado 1: Módulos y Paquetes
# Los módulos son archivos de Python (o de Cpython o de C).
# Estos archivos pueden contener funciones, clases y variables.
# Podemos importarlas a nuestros archivos para hacer uso de estas
# sin tener que volver a escribir el código.
```

Para este capítulo vamos a crear un nuevo archivo llamado `calcularArea.py` además del archivo que estamos trabajando.



En este módulo vamos a definir una funciones.

```
1  PI = 3.1415
2
3  def areaCirculo(radio):
4      |   return PI*radio**2
5
6  def areaTriangulo(base, altura):
7      |   return base*altura/2
8
9  def areaCuadrado(lado):
10     |   return lado*lado
```

Ahora nos vamos al otro documento.

```
1  import calcularArea
2
3  print(calcularArea.PI)
4  print(calcularArea.areaCirculo(5))
```

Vamos a ejecutar:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/modulos-paquetes.py"
3.1415
78.53750000000001
PS C:\Users\pmver\Desktop\Ejer Python>
```

Cada vez que llamamos a una función tenemos que hacer referencia al nombre del archivo, para evitarlo.

```
1 import calcularArea
2 from calcularArea import areaCuadrado
3
4 print(calcularArea.PI)
5 print(calcularArea.areaCirculo(5))
6
7 print(areaCuadrado(10))
```

Si ejecutamos de nuevo:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/modulos-paquetes.py"
3.1415
78.53750000000001
100
PS C:\Users\pmver\Desktop\Ejer Python>
```

Si queremos ejecutar la función areaTriangulo sin calcularArea. nos dará error para ello la tendríamos que importar.

```
1 import calcularArea
2 from calcularArea import areaCuadrado, areaTriangulo
3
4 print(calcularArea.PI)
5 print(calcularArea.areaCirculo(5))
6
7 print(areaCuadrado(10))
8 print(areaTriangulo(2,3))
```

Este será el resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/modulos-paquete s.py"
3.1415
78.53750000000001
100
3.0
PS C:\Users\pmver\Desktop\Ejer Python>
```

¿Qué pasa si un módulo tiene un montón de funciones y variables.

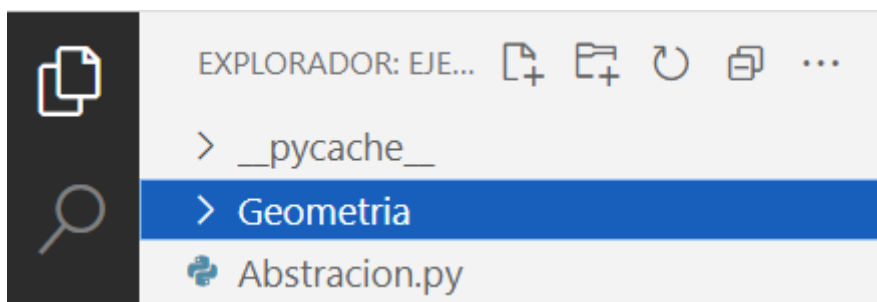
```
1 import calcularArea
2 from calcularArea import *
3
4 print(PI)
5 print(areaCirculo(5))
6
7 print(areaCuadrado(10))
8 print(areaTriangulo(2,3))
```

Utilizaremos el asterisco, este será el resultado:

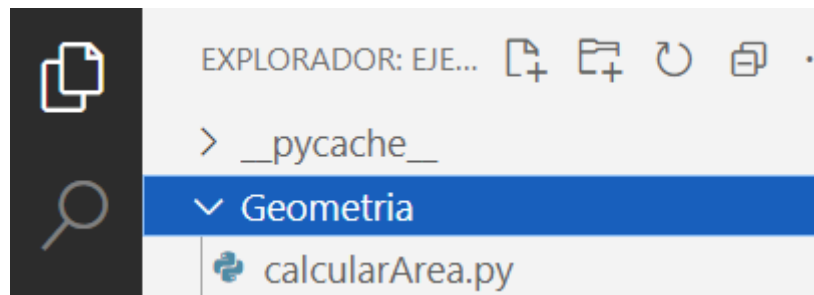
```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/modulos-paquete s.py"
3.1415
78.53750000000001
100
3.0
PS C:\Users\pmver\Desktop\Ejer Python>
```

Ahora vamos a ver que es un paquete.

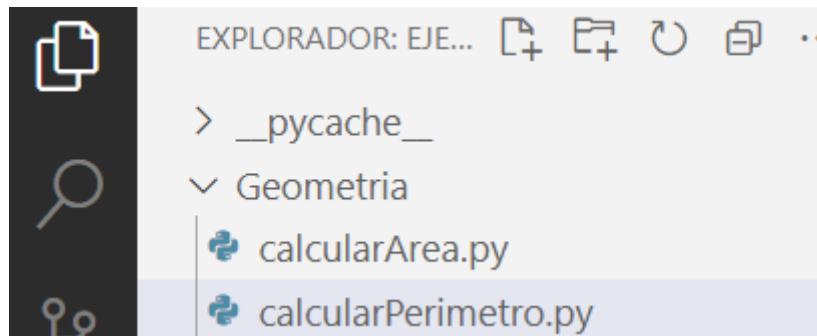
Vamos a crear una carpeta llamada geometría.



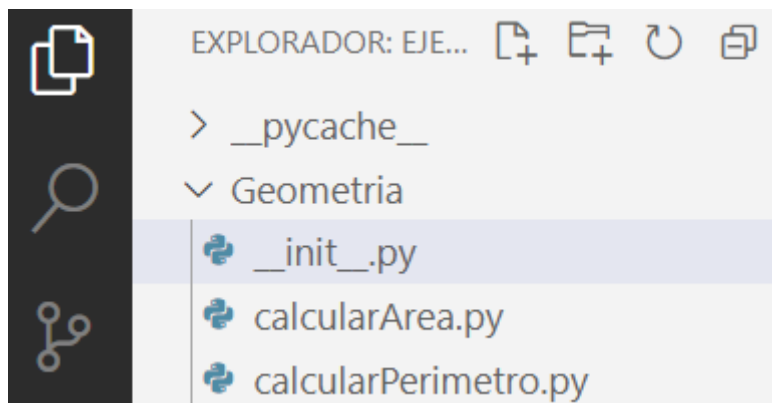
Dentro de esta carpeta vamos a mover el archivo calcularArea.py.



Dentro de la carpeta Geometria vamos a crear otro fichero llamado calculaPerimetro.py



Ahora vamos a crear un fichero __init__.py que dejaremos vacío.



Este archivo se ejecuta cuando cargamos un paquete.

Nos vamos al fichero calculaPerimetro.py

```
1  import math
2
3  class CalculaPerimetros:
4      def __init__(self):
5          ...
6
7      def perimetroCuadrado(self, lado):
8          return lado*4
9
```

```

10         def perimetroCirculo(self, radio):
11             return math.pi*radio*2

```

Ahora nos vamos al fichero modulos-paquetes.py

```

1 from Geometria.calcularPerimetro import calculaPerimetros as CP

```

Del paquete Geometria el módulo calcularPerimetro me importas la clase calculaPerimetros y me lo vas a renombrar con CP.

```

2
3 # Los paquetes son conjuntos de módulos, relacionados entre si,
4 # en un mismo directorio.
5
6 cp = CP()
7 print(cp.perimetroCirculo(2))

```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/modulos-paquete s.py"

```

```

12.566370614359172

```

```

PS C:\Users\pmver\Desktop\Ejer Python> 

```

```

1 from Geometria.calcularPerimetro import CalculaPerimetros as CP
2 from Geometria.calcularArea import *
3
4 # Los paquetes son conjuntos de módulos, relacionados entre si,
5 # en un mismo directorio.
6
7 cp = CP()
8 print(cp.perimetroCirculo(2))
9 print(areaCirculo(3))

```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/modulos-paquete s.py"

```

```

12.566370614359172

```

```

28.273500000000002

```

```

PS C:\Users\pmver\Desktop\Ejer Python> 

```

23.- Archivos Externos [LEER y ESCRIBIR ficheros.txt] + Ejercicio resuelto

```
# Capítulo 23: Archivos externos
# Apartado 1: Abrir y leer un documento existente.

# Por defecto, el modo de abrir un fichero es lectura.
# como texte. Por tanto para leer su contenido no
# es necesario especificar el segundo argumento.

# r -> read
# w -> write
# a -> append
# x -> create

# t -> text-mode
# b -> byte - para archivos como fotos
```

Para este capitulo vamos a crear nuestro fichero demo.txt.

≡ demo.txt

```
1  Podréis darle un buen LIKE al vídeo.
2  También os podéis SUSCRIBIR, que me da una alegría.
3  Y si os apetece podéis COMENTAR qué os ha parecido.
```

Volvemos al archivo que hemos creado para este proyecto.

```
1  fichero = open('./demo.txt', 'rt', encoding='utf-8')
```

Vamos a abrir un fichero llamado demo.txt que se encuentra en el mismo directorio donde esta el archivo del proyecto.

r -> En modo lectura.

T -> En modo de texto.

encoding = 'utf-8' -> Para que reconozca las vocales con acento.

```
2  primeraLinea = fichero.readline()
3  print(primerLinea)
```

El objeto fichero con el método readline() nos permite leer la primera línea que lo asignaremos a la variable primeraLinea que a continuación se mostrará en consola.

Este será el resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-externos.py"
```

Podréis darle un buen LIKE al vídeo.

```
PS C:\Users\pmver\Desktop\Ejer Python>
```

Podréis observar que a continuación hay una línea en blanco, esto es porque también ha leído el salto de línea de la primera frase.

```
5 print(fichero.readline())
```

Esto nos mostrará la segunda línea.

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-externos.py"
```

Podréis darle un buen LIKE al vídeo.

También os podéis SUSCRIBIR, que me da una alegría.

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```
1 fichero = open('./demo.txt', 'rt', encoding='utf-8')
2 primeraLinea = fichero.readline()
3 print(primeraLinea)
4 print(fichero.readline())
5 print(fichero.readline())
```

Ahora ya podemos imprimir la tercera línea.

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-externos.py"
```

Podréis darle un buen LIKE al vídeo.

También os podéis SUSCRIBIR, que me da una alegría.

Y si os apetece podéis COMENTAR qué os ha parecido.

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Como en la tercera línea no hemos dado a enter no aparece el espacio en blanco.

```
# Para leer todas las líneas (restantes) de un fichero,
# podemos utilizar el método reeaddlines. Nos
# retorna una lista cuyos elementos son líneas
# del fichero.
```

```

1 fichero = open('./demo.txt', 'rt', encoding='utf-8')
2 todasLasLineas = fichero.readlines()
3 print(todasLasLineas)

```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-externos.py"
['Podréis darle un buen LIKE al vídeo.\n', 'También os podéis SUSCRIBIR, que me da una alegría.\n', 'Y si os apetece podéis COMENTAR qué os ha parecido.']
PS C:\Users\pmver\Desktop\Ejer Python> █

```

```

1 fichero = open('./demo.txt', 'rt', encoding='utf-8')
2 print(fichero.readline())
3 todasLasLineas = fichero.readlines()
4 print(todasLasLineas)

```

Si en la línea 2 mostramos la primera línea cuando quiere mostrar toda las líneas de nuevo, la primera ya no nos la mostrará.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-externos.py"
Podréis darle un buen LIKE al vídeo.

['También os podéis SUSCRIBIR, que me da una alegría.\n', 'Y si os apetece podéis COMENTAR qué os ha parecido.']
PS C:\Users\pmver\Desktop\Ejer Python> █

```

El objeto fichero tienen un cursor (puntero) que lo que va haciendo es desplazarse y va contando caracteres y cuando llamamos `fichero.readlines()` muestra desde donde está el cursor hasta el final del fichero.

Debemos cerrar el fichero una vez hayamos terminado

```
6 fichero.close()
```

Apartado 2: Escribir en un fichero existente

Si un fichero ya tiene contenido, tenemos dos opciones:

Sobreescribir el contenido, o añadir el contenido

nuevo al final del que ya existe. Modo write:

```

6 fichero = open('./demo.txt', 'w', encoding='utf-8')
7 fichero.write("He sobrescrito el fichero.\n")
8 fichero.close

```

Vamos a ejecutar.

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-extern
os.py"
PS C:\Users\pmver\Desktop\Ejer Python> 

```

Por la consola no se muestra nada, vamos a abrir el fichero demo.txt.

```

1 He sobrescrito el fichero.
2 |

```

Ahora el fichero contiene el nuevo texto.

```

5 fichero = open('./demo.txt', 'w', encoding='utf-8')
6 fichero.write("He sobrescrito el fichero.\n")
7 listaContenido = [
8     'Dima es el mejor Youtuber.\n',
9     'Me turboflica su',
10    ' curso d ePython.\n'
11    'Adios muy buenas.\n'
12 ]
13 fichero.writelines(listaContenido)
14 fichero.close

```

Vamos a ejecutar y ver el contenido del archivo demo.txt.

```

1 He sobrescrito el fichero.
2 Dima es el mejor Youtuber.
3 Me turboflica su curso d ePython.
4 Adios muy buenas.

```

En la línea 5 o crea el fichero demo.txt o reemplaza todo su texto por el nuevo texto.

En la línea 13 agregamos en contenido de la lista al fichero al final del mismo, al no haberse cerrado el puntero del mismo está al final de la primera línea.

El \n provoca el salto de línea.

Vamos a realizar una modificaciones:

```

5 fichero = open('./demo.txt', 'w', encoding='utf-8')
6 fichero.write("He sobrescrito el fichero.\n")
7 listaContenido = [
8     'Dima es el mejor Youtuber.',
9     'Me turboflica su curso d ePython.',
10    'Adios muy buenas.'
11 ]
12 listaContenido = list(map(lambda line: line + '\n', listaContenido))
13 print(listaContenido)
14 fichero.writelines(listaContenido)
15 fichero.close

```

En este ejemplo para evitar tener que escribir en cada línea '\n' en la línea 12 con el método lambda hacemos que se incorpore el '\n' automáticamente.

En la línea 13 mostrará el resultado por consola cuando ejecutemos.

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-externos.py"
['Dima es el mejor Youtuber.\n', 'Me turboflica su curso d ePython.\n', 'Adios muy buenas.\n']
PS C:\Users\pmver\Desktop\Ejer Python> █

```

Y este será el contenido del fichero demo.txt.

```

1  He sobrescrito el fichero.
2  Dima es el mejor Youtuber.
3  Me turboflica su curso d ePython.
4  Adios muy buenas.
5
# Apartado 3: Escribir en un fichero existente.
# Modo APPEND. Nos sirve para añadir contenido
# sin aliminar el existente:

1  fichero = open('./demo.txt', 'a', encoding='utf-8')
2  fichero.write('\n\nEsto es una nueva linea.')
3  fichero.close()

```

Vamos a ejecutar y visualizar el contenido del fichero demo.txt.

```

1  He sobrescrito el fichero.
2  Dima es el mejor Youtuber.
3  Me turboflica su curso d ePython.

```

```
4   Adios muy buenas.  
5  
6  
7  
8   Esto es una nueva linea.
```

Línea que se a agregado al fichero demo.txt.

```
# Apartado 4: Crear un nuevo fichero.  
# Cuando usamos open en modo create,  
# podemos escribir en el ficheros tras crearlo  
# pero no leer.
```

```
1   fichero = open('./demo2.txt', 'x', encoding='utf-8')  
2   fichero.write("Soy un fichero nuevo.")  
3   fichero.close()
```

Vamos a ejecutar:

```
≡ demo2.txt  
1   Soy un fichero nuevo.
```

Vamos a agregar un método llamado `.readable()` y nos dará un valor booleano de si se puede leer o no.

```
1   fichero = open('./demo2.txt', 'x', encoding='utf-8')  
2   fichero.write("Soy un fichero nuevo.")  
3   print(fichero.readable())  
4   fichero.close()
```

Borramos el fichero demo2.txt y vamos a ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/  
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-extern  
os.py"
```

```
False
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Vamos a agregar el método `.writable()` y nos dará un valor booleano de si se puede escribir o no.

```

1 fichero = open('./demo2.txt', 'x', encoding='utf-8')
2 fichero.write("Soy un fichero nuevo.")
3 print(fichero.readable())
4 print(fichero.writable())
5 fichero.close()

```

Borramos el fichero demo2.txt y vamos a ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-extern
os.py"
False
True
PS C:\Users\pmver\Desktop\Ejer Python> █

```

Si intentamos crear un archivo que ya existe es nos dará un error y el programa se detendrá, vamos a controlarlo con las excepciones.

```

1 try:
2     fichero = open('./demo2.txt', 'x', encoding='utf-8')
3     fichero.write("Soy un fichero nuevo.")
4     print(fichero.readable())
5     print(fichero.writable())
6     fichero.close()
7 except FileExistsError:
8     print("No se puede crear porque ya existe")

```

Este será el resultado si intentamos crear un fichero que ya existe:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-extern
os.py"
No se puede crear porque ya existe
PS C:\Users\pmver\Desktop\Ejer Python> █

```

Apartado 5: Método seek. Podemos controlar
la posición desde la cual empezamos a leer.

```

1 # La lectura viene por defecto.
2 fichero = open('./demo.txt', encoding='utf-8')
3 fichero.seek(10) # Desplazamos el cursor 10 caracteres
4 print("Estamos leyendo desde el décimo caracter:\n")
5 print(fichero.read())

```

6 fichero.close()

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

Estamos leyendo desde el décimo caracter:

rito el fichero.
Dima es el mejor Youtuber.
Me turboflica su curso d ePython.
Adios muy buenas.

La frase original es:

He sobrescrito el fichero.

Esto es una nueva línea.

PS C:\Users\pmver\Desktop\Ejer Python> █

Apartado 6: Lectura y escritura simultáneas.

```
1 fichero = open('./demo.txt', 'r+', encoding='utf-8')
2 lineas = fichero.readlines()
3 print(lineas)
4
5 fichero.write('\nEsta línea es nueva.')
6 fichero.close()
```

Este es el fichero antes de ejecutar.

≡ demo.txt

```
1 He sobrescrito el fichero.
2 Dima es el mejor Youtuber.
3 Me turboflica su curso d ePython.
4 Adios muy buenas.
```

Una vez ejecutado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-externos.py"

```
['He sobrescrito el fichero.\n', 'Dima es el mejor Youtuber.\n', 'Me turboflica su curso d ePython.\n', 'Adios muy buenas.']
```

PS C:\Users\pmver\Desktop\Ejer Python> █

Lee el fichero y el cursor se sitúa al final.

demo.txt

```
1 He sobrescrito el fichero.  
2 Dima es el mejor Youtuber.  
3 Me turboflica su curso de Python.  
4 Adios muy buenas.  
5 Esta línea es nueva.
```

Escribe la última línea.

Ejercicio resuelto:

Queremos comparar el tiempo que tarda en sumar todos los elementos de una lista un bucle for y un bucle while.

Para ello, debemos calcular el tiempo empleado en cada bucle varias veces y luego promediarlo.

Debemos cronometrar lo que tarda cada bucle 100 veces y guardar cada resultado en un fichero.

```
port time, random  
lista = []  
for x in range(50000):  
    lista.append(random.randint(0, 150000))  
  
fichero = open('./times.txt', 'wt', encoding='utf-8')  
  
for x in range(100):  
    accFor = 0  
    starTime = time.time()  
    for num in lista:  
        accFor = accFor + num  
    elapsedTimeFor = time.time() - starTime  
  
    pos = 0  
    accWhile = 0  
    starTime = time.time()  
    while pos < len(lista):  
        accWhile = accWhile + lista[pos]  
        pos = pos + 1  
    elapsedTimeWhile = time.time() - starTime  
    fichero.write(f"{elapsedTimeFor};{elapsedTimeWhile}\n")  
  
fichero.close()  
fichero = open('./times.txt', encoding='utf-8')  
  
meanFor = 0  
meanWhile = 0  
samples = 0
```

```

for line in fichero.readlines():
    line.replace('\n', '')
    timeFor, timeWhile = line.split(";")
    meanFor += float(timeFor)
    meanWhile += float(timeWhile)
    samples +=1

print(f"Tiempo for: {(meanFor/samples)*(10**3)} ms. Tiempo While:
{(meanWhile/samples)*(10**3)} ms")
fichero.close()

```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```

PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/
Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/archivos-extern
os.py"
Tiempo for: 3.5311102867126465 ms. Tiempo While: 9.903161525726318 ms
PS C:\Users\pmver\Desktop\Ejer Python>

```

El contenido del fichero times.txt.

```

1      0.002991914749145508;0.008975744247436523
2      0.003989696502685547;0.008975744247436523
3      0.0029921531677246094;0.007978439331054688
4      0.0029921531677246094;0.008975982666015625
5      0.003989219665527344;0.007978677749633789
6      0.002994537353515625;0.00897359848022461
7      0.0029921531677246094;0.008975744247436523
8      0.0029921531677246094;0.009973287582397461

...

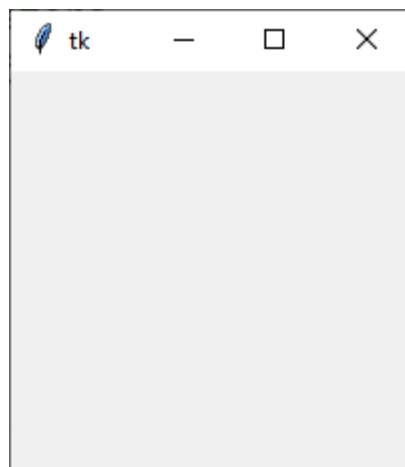
93     0.003988742828369141;0.008970022201538086
94     0.0029926300048828125;0.008976459503173828
95     0.002990245819091797;0.008976459503173828
96     0.002992391586303711;0.008975744247436523
97     0.0029921531677246094;0.008975505828857422
98     0.0029921531677246094;0.008975505828857422
99     0.002992391586303711;0.008976221084594727
100    0.003988981246948242;0.008975982666015625

```

24.- Interfaces gráficas con TKINTER

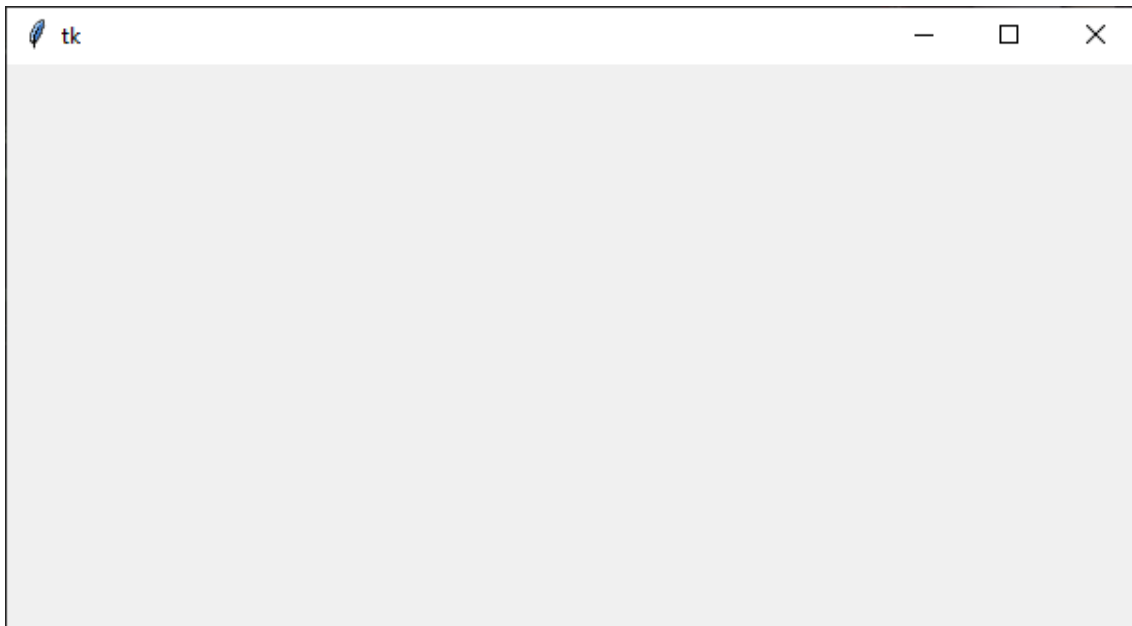
```
1  # Capítulo 24: Interfaces gráficas con tkinter
2  # Apartado 1: Conceptos básicos de tkinter.
3
4  import tkinter as tk
5
6  # Tkinter funciona mediante Widgets. Los Widgets son
7  # elementos predefinidos que podemos ir incrustando
8  # en nuestra aplicación.
9  # Por ejemplo, un botón, un input de texto, etcétera.
10 # De hecho, la ventana principal de la aplicación
11 # es un widget.
12
13 # Colores: '#00a8e8'
14 # Fuente: Courier, 25
15
16 app = tk.Tk()
17 app.mainloop()
```

Vamos a ejecutar:

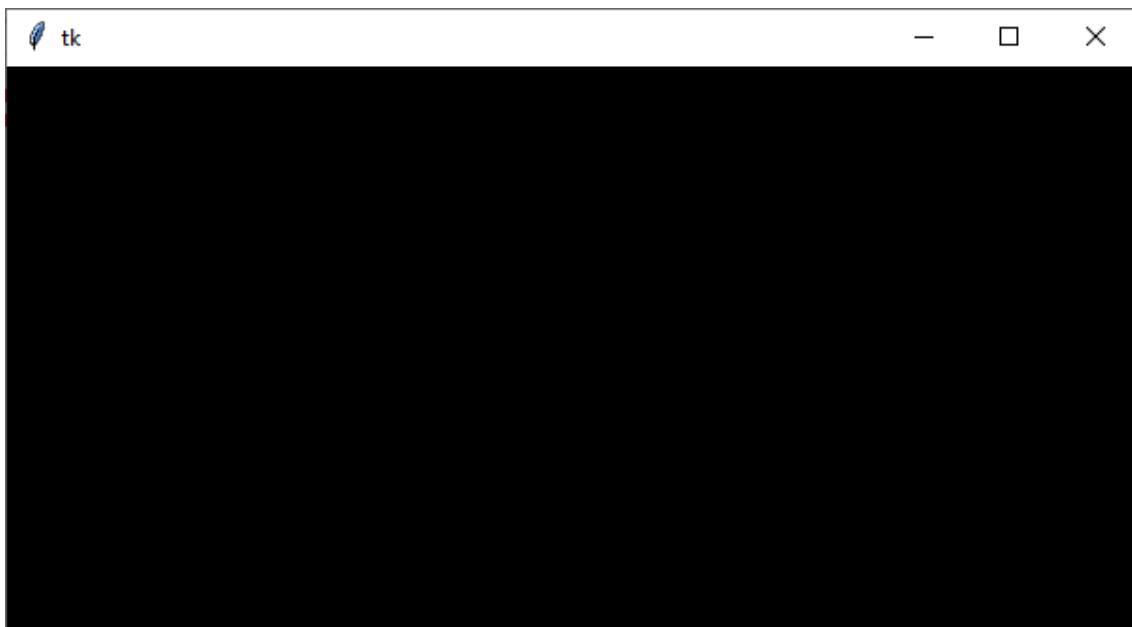


```
16 app = tk.Tk()
17 # Dimensiones: Anchura x Altura
18 app.geometry("600x300")
19 app.mainloop()
```

Definimos las dimensiones de la ventana.

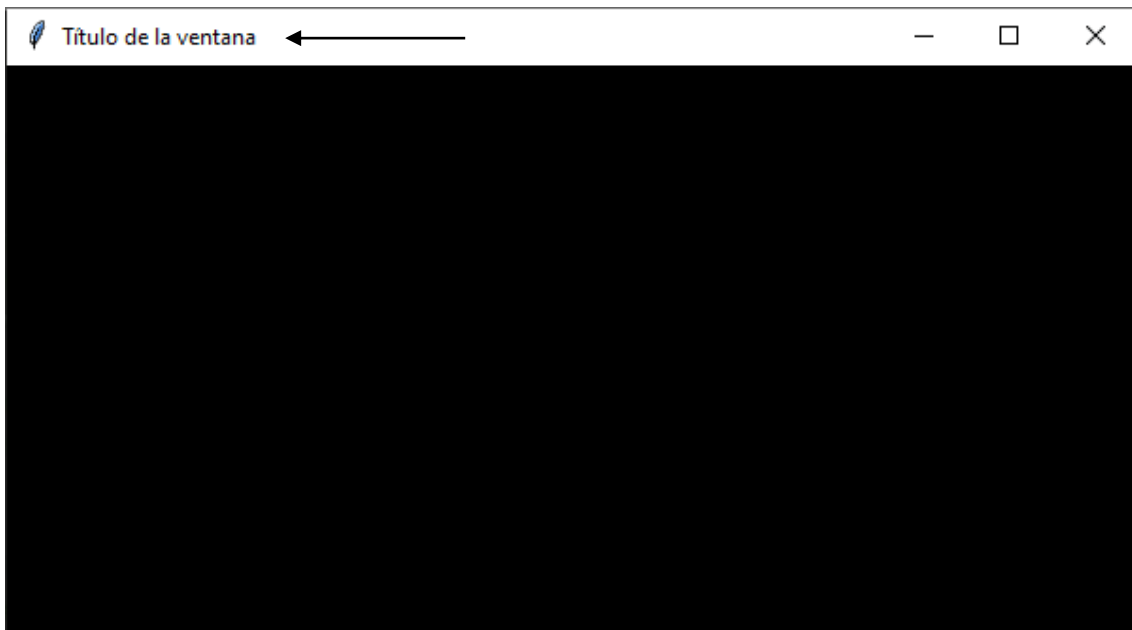


```
16 app = tk.Tk()
17 # Dimensiones: Anchura x Altura
18 app.geometry("600x300")
19 app.configure(background="black")
20 app.mainloop()
```

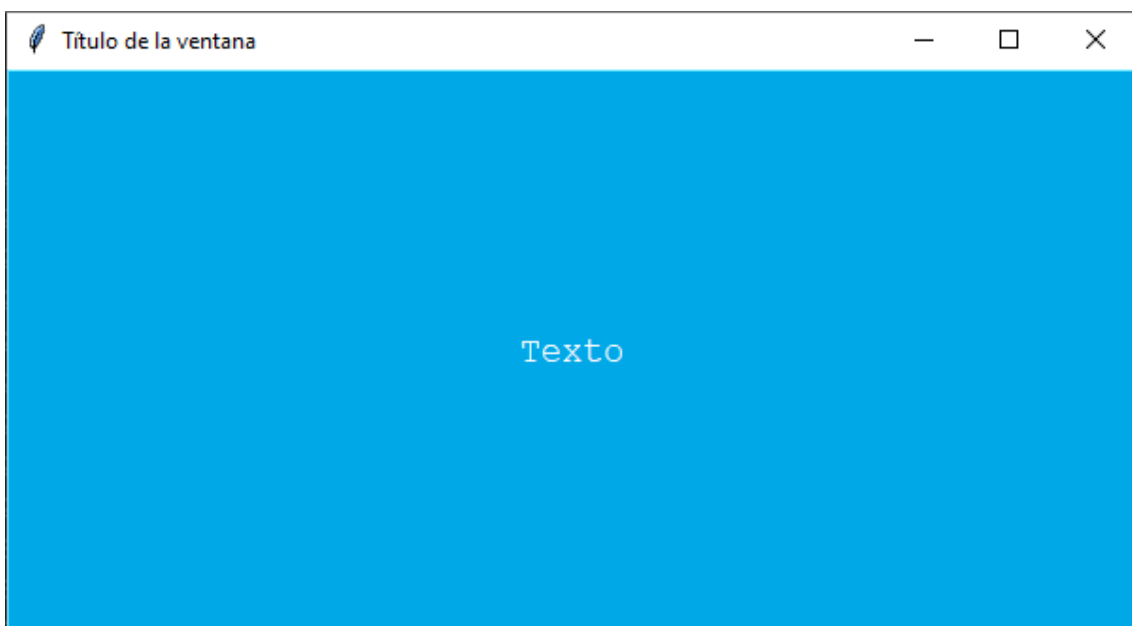


```
16 app = tk.Tk()
17 # Dimensiones: Anchura x Altura
18 app.geometry("600x300")
19 app.configure(background="black")
20 tk.Wm.wm_title( app, "Título de la ventana")
```

21 app.mainloop()



```
20 tk.Wm.wm_title( app, "Título de la ventana")
21
22 tk.Button(
23     app,
24     text="Texto",
25     font=("Courier", 14),
26     bg="#00a8e8",
27     fg="white"
28 ).pack(
29     fill=tk.BOTH,
30     expand=True,
31 )
32 app.mainloop()
```



Ahora vamos a hacer que al presionar el botón diga por consola "Has hecho clic en el botón".

```
22 def click():
23     print("Has hecho clic en el botón.")
24 tk.Button(
25     app,
26     text="Texto",
27     font=("Courier", 14),
28     bg="#00a8e8",
29     fg="white",
30     command=click, # Aquí no ponemos los paréntesis.
31 ).pack(
32     fill=tk.BOTH,
33     expand=True,
34 )
35 app.mainloop()
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/tkinter-1.py"
```

```
Has hecho clic en el botón.
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Otra forma de hacerlo:

Borramos la función.

```
22 tk.Button(
23     app,
24     text="Texto",
25     font=("Courier", 14),
26     bg="#00a8e8",
27     fg="white",
28     command=lambda: print("Has hecho clic en el botón.")
29 ).pack(
30     fill=tk.BOTH,
31     expand=True,
32 )
```

Vamos a agregar una etiqueta:

```
34 tk.Label(
35     app,
36     text="Etiqueta",
```

```

37         fg="white",
38         bg="black",
39         justify="center"
40     ).pack(
41         fill=tk.BOTH,
42         expand=True,
43     )
44     app.mainloop()

```

Este será el resultado:

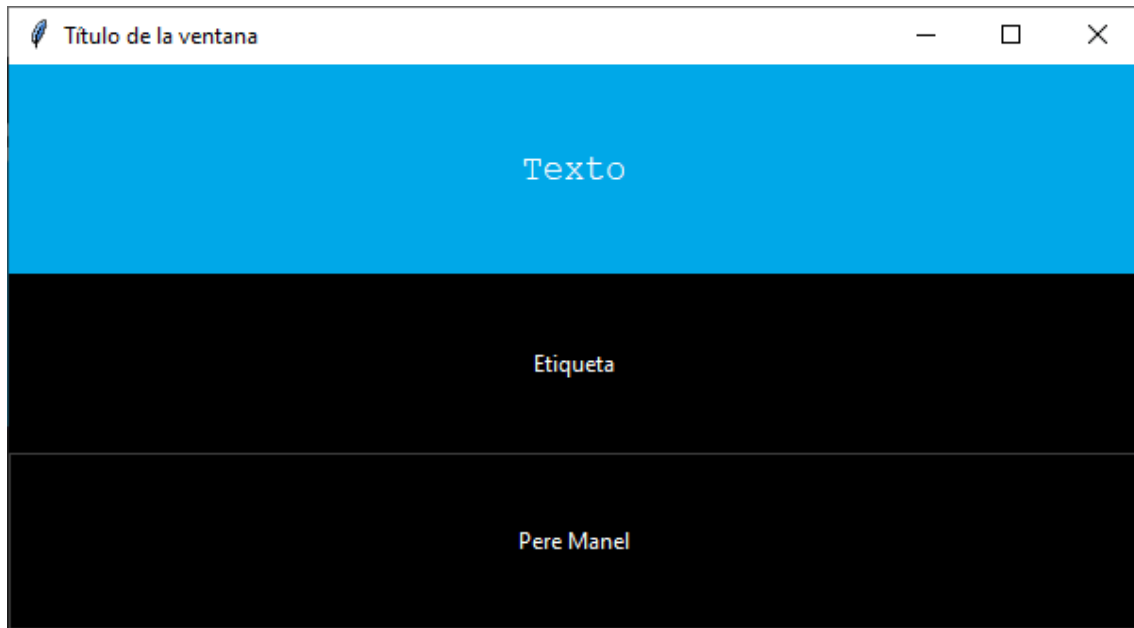


Vamos a crear una entrada de texto:

```

46  tk.Entry(
47      app,
48      fg="white",
49      bg="black",
50      justify="center"
51  ).pack(
52      fill=tk.BOTH,
53      expand=True,
54  )
55
56  app.mainloop()

```



Pero este dato que he introducido no lo puedo recoger.

```
app = tk.Tk()
entrada = tk.StringVar(app)
# Dimensiones: Anchura x Altura
app.geometry("600x300")
app.configure(background="black")
tk.Wm.wm_title( app, "Título de la ventana")
```

Agregamos una variable

```
tk.Button(
    app,
    text="Texto",
    font=("Courier", 14),
    bg="#00a8e8",
    fg="white",
    command=lambda: print("Has escrito la palabra " + entrada.get()),
    relief="flat" # El botón no tenga relieve
).pack(
    fill=tk.BOTH,
    expand=True,
)

tk.Label(
    app,
    text="Etiqueta",
    fg="white",
    bg="black",
    justify="center"
).pack(
    fill=tk.BOTH,
    expand=True,
)
```

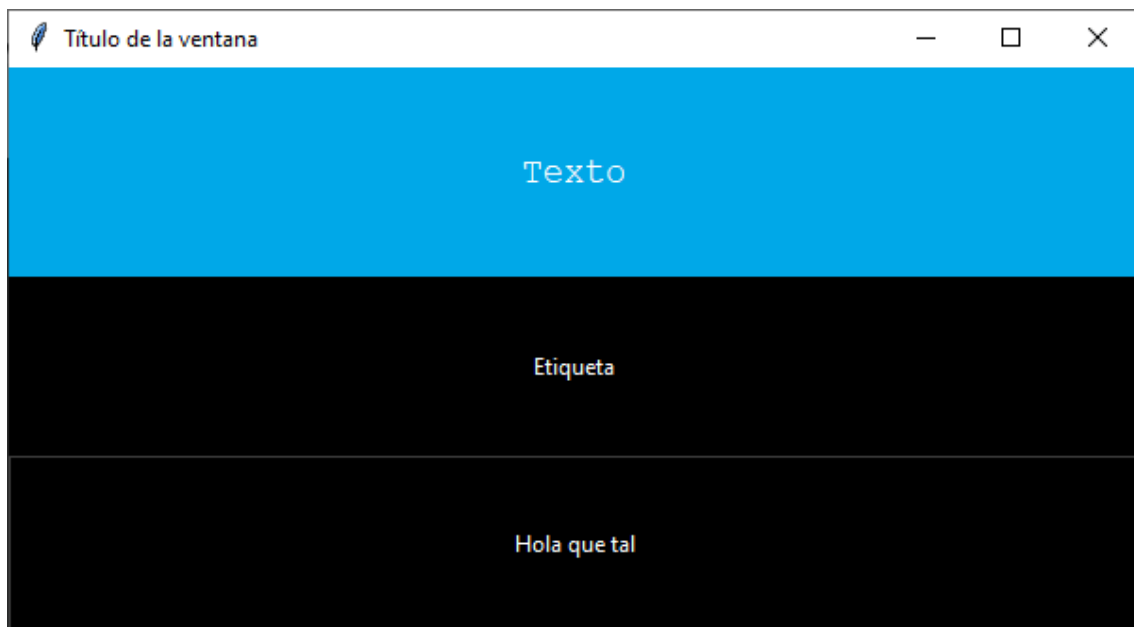
Mostramos la variable por consola.

```
tk.Entry(
    app,
    fg="white",
    bg="black",
    justify="center",
    textvariable=entrada
).pack(
    fill=tk.BOTH,
    expand=True,
)
```

Definimos la variable en la entrada.

```
app.mainloop()
```

Ejecutamos:



Escribimos el texto "Hola que tal".

Pulsamos el botón.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/tkinter-1.py"
Has escrito la palabra Hola que tal
```

Ahora queremos que cuando introduzcas un texto en la entrada la etiqueta muestre un texto más la palabra que has introducido.

```
app = tk.Tk()
palabra = tk.StringVar(app)
entrada = tk.StringVar(app)
# Dimensiones: Anchura x Altura
app.geometry("600x300")
```

Definimos la variable palabra.

```
app.configure(background="black")
tk.Wm.wm_title( app, "Título de la ventana")
```

```
def cambiarPalabra():
    palabra.set("Suscribete " + entrada.get())
```

Creamos la función
cambiarPalabra()

```
tk.Button(
    app,
    text="Texto",
    font=("Courier", 14),
    bg="#00a8e8",
    fg="white",
    command=cambiarPalabra,
    relief="flat" # El botón no tenga relieve
).pack(
    fill=tk.BOTH,
    expand=True,
)
```

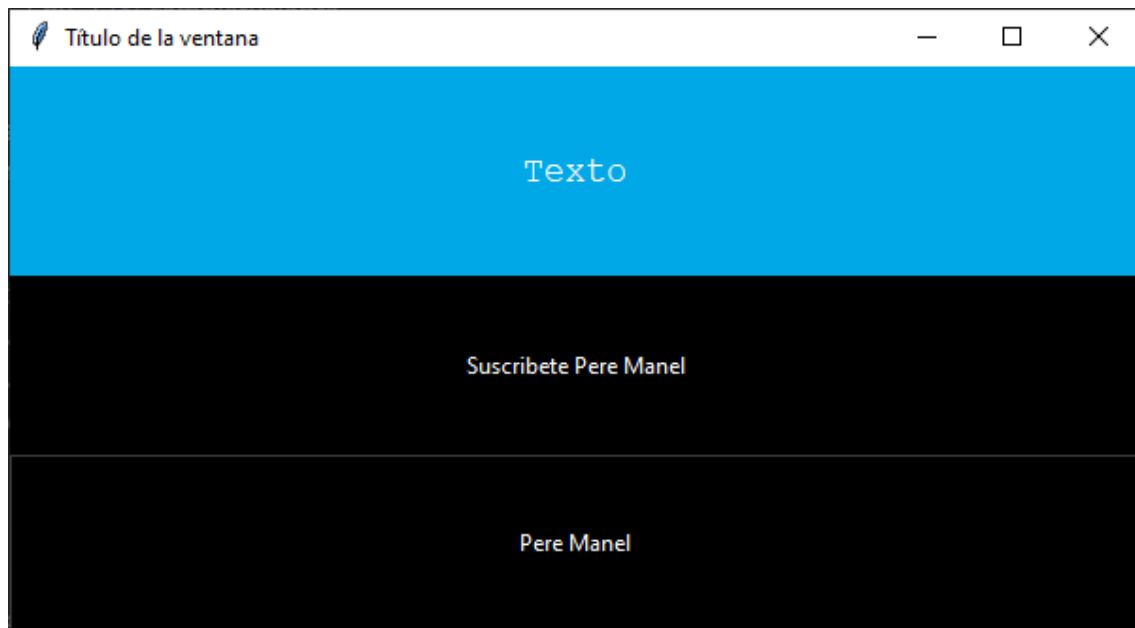
Al hacer clic en el botón ejecutamos
la función cambiarPalabra(), no lleva
paréntesis por que lo pasamos como
un objeto.

```
tk.Label(
    app,
    text="Etiqueta",
    textvariable=palabra,
    fg="white",
    bg="black",
    justify="center"
).pack(
    fill=tk.BOTH,
    expand=True,
)
```

```
tk.Entry(
    app,
    fg="white",
    bg="black",
    justify="center",
    textvariable=entrada
).pack(
    fill=tk.BOTH,
    expand=True,
)
```

```
app.mainloop()
```

Vamos a ejecutar e introduciremos nuestro nombre y a continuación haremos clic en el botón.



Para profundizar en tkinder, puedes acceder al siguiente enalce:

http://www.peremanelv.com/Pro_Python/Tkinter.pdf



25.- ¿Qué son los DECORADORES en Python?

Los decoradores sirven para modificar el comportamiento de una función.

Apartado 1: Las funciones son objetos por tanto:

- Es una instancia de tipo Objeto.
- Podemos almacenar una función en una variable.
- Podemos pasar una función como parámetro de otra función.
- Una función puede retornar otra función.
- Se puede almacenar en estructuras de datos, tablas, listas...

```
1 def funcionExterna(nombre):
2     def funcionEnvoltorio():
3         print("\nEmpieza la función envoltorio")
4         print(nombre)
5         print("Final de la función envoltorio\n")
6     return funcionEnvoltorio
```

8 `funcionExterna("Dimas")`

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

```
PS C:\Users\pmver\Desktop\Ejer Python>
```

Como podréis ver no imprime nada, porque retorna un objeto ya que la función no tiene paréntesis.

```
1  ✓ def funcionExterna(nombre):
2  ✓     def funcionEnvoltorio():
3         print("\nEmpieza la función envoltorio")
4         print(nombre)
5         print("Final de la función envoltorio\n")
6     return funcionEnvoltorio()
```

En la línea 6 hemos agregado los paréntesis, ahora retorna la ejecución de la función, no el objeto, vamos a ejecutar:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

```
Empieza la función envoltorio
Dimas
Final de la función envoltorio
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Le vamos a quitar los paréntesis para que retorne un objeto.

```
1  def funcionExterna(nombre):
2      def funcionEnvoltorio():
3          print("\nEmpieza la función envoltorio")
4          print(nombre)
5          print("Final de la función envoltorio\n")
6      return funcionEnvoltorio
7
8  instanciaFuncion = funcionExterna("Dimas")
9  instanciaFuncion()
```

La variable instanciaFuncion le asignamos la funciónExterna("Dimas") como un objeto.

Si esta variable le agregamos paréntesis al final esta funcionará igual que una función, si ejecutamos este será el resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

```
Empieza la función envoltorio
Dimas
Final de la función envoltorio
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

```
1  def funcionExterna(nombre):
2      def funcionEnvoltorio():
3          print("\nEmpieza la función envoltorio")
4          print(nombre)
5          print("Final de la función envoltorio\n")
6      return funcionEnvoltorio
7
```

```

8     instanciaFuncion = funcionExterna("Dimas")
9     instanciaFuncion()
10
11     otraFuncion = funcionExterna("María")
12     otraFuncion()

```

Ahora creamos otra variable asignándole como objeto la funciónExterna("María"), vamos a ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

```

Empieza la función envoltorio
Dimas
Final de la función envoltorio

```

```

Empieza la función envoltorio
María
Final de la función envoltorio

```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Primero ejecuta instanciaFuncion() y a continuación otraFuncion().

```

1     def funcionExterna(nombre):
2         def funcionEnvoltorio():
3             print("\nEmpieza la función envoltorio")
4             print(nombre)
5             print("Final de la función envoltorio\n")
6         return funcionEnvoltorio
7
8     otraFuncion = funcionExterna("María")
9     otraFuncion()
10
11     print(otraFuncion.__name__)

```

Este objeto también tiene atributos, vamos a ejecutar:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

```
Empieza la función envoltorio
María
Final de la función envoltorio
```

```
funcionEnvoltorio
```

```
PS C:\Users\pmver\Desktop\Ejer Python> 
```

Es la función que retorna.

Apartado 2: Función como argumento de otra función, en vez de pasar la variable “nombre”, pasamos otra función.

```
1  # Para manipular fechas y horas
2  from datetime import datetime
3
4  def fecha():
5      print(datetime.today().strftime("%d-%m-%Y"))
6
7  fecha()
```

Creamos una función fecha() y a continuación comprobamos su resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

```
01-02-2023
```

```
PS C:\Users\pmver\Desktop\Ejer Python> 
```

Borramos la llamada a la función.

```
1  # Para manipular fechas y horas
2  from datetime import datetime
3
4  def fecha():
5      print(datetime.today().strftime("%d-%m-%Y"))
6
7  def hora():
8      print(datetime.now().strftime("%H:%M:%S"))
9
10  hora()
```

Creamos la función hora() y comprobamos su resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

```
06:33:15
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Borramos la llamada a la función.

```
# Para manipular fechas y horas
from datetime import datetime

def fecha():
    print(datetime.today().strftime("%d-%m-%Y"))

def hora():
    print(datetime.now().strftime("%H:%M:%S"))

def funcionExterna(funcionInterna):
    def funcionEnvoltorio():
        print("\nEmpieza la función")
        funcionInterna()
        print("Fin de la función\n")
    return funcionEnvoltorio

mostrarHora = funcionExterna(hora)
mostrarFecha = funcionExterna(fecha)
```

Este será el resultado:

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

```
Empieza la función
06:44:03
Fin de la función
```

```
Empieza la función
01-02-2023
Fin de la función
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Apartado 3: Sintaxis de los Decoradores.

Vamos a realizar el código para que realice lo mismo que el ejercicio anterior pero con una sintaxis más cómoda.

```
def funcionExterna(funcionInterna):
    def funcionEnvoltorio():
        print("\nEmpieza la función")
        funcionInterna()
        print("Fin de la función\n")
    return funcionEnvoltorio
```

```
@funcionExterna
def saludar():
    print("Hola crash!")
```

```
@funcionExterna
def despedirse():
    print("Adios máquina")
```

```
saludar()
despedirse()
```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
Empieza la función
Hola crash!
Fin de la función
```

```
Empieza la función
Adios máquina
Fin de la función
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Esta es la sintaxis correcta de Python.

Apartado 4: Funciones con argumentos, veremos cómo podemos usar decoradores con funciones que reciban diferentes números de argumentos.

```
1  ✓ def sumarNumeros(*args, **kwargs):
2      |     acc = 0
3  ✓     |     for num in args:
4      |         |     acc += num
5      |         |     return acc
6
7      print(sumarNumeros(1,2,3,4,5,6,7,8,9,10))
```

Hemos creado una función para diferentes argumentos, lo ejecutamos:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

55

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

A continuación borramos la línea 7 donde llamamos a la función.

```
def sumarNumeros(*args, **kwargs):
    acc = 0
    for num in args:
        acc += num
    return acc

def operarConPares(operacion):
    def wrapper(*args, **kwargs):
        soloPares = list(filter(lambda num: num%2 == 0, args))
        resultado = operacion(*soloPares, **kwargs)
        print(f"El resultado de la operación es: {resultado}")
        return resultado
    return wrapper
```

```
sumarPares = operarConPares(sumarNumeros)
sumarPares(1,2,3,4,5,6)
```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

```
El resultado de la operación es: 12
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

Vamos a poner un ejemplo con la arroba.

```
@operarConPares
def sumarNumeros(*args, **kwargs):
    acc = 0
    for num in args:
        acc += num
    return acc
```

```
sumarNumeros(1,2,3,4,5,6)
```

El resultado es el mismo.

Reto:

Tenemos que en la función de multiplicar tenemos que devolver el resultado de la multiplicación, siempre y cuando no supere un valor máximo.

El valor máximo no puede estar fijo, tiene que ser variable.

```
def operarConPares(operacion):
    def wrapper(*args, **kwargs):
        soloPares = list(filter(lambda num: num%2 == 0, args))
        resultado = operacion(*soloPares, **kwargs)
        print(f"El resultado de la operación es: {resultado}")
        return resultado
    return wrapper

@operarConPares
def multiplicacion(*args, **kwargs):
    acc = 1
    for num in args:
        acc *= num
    if "max" in kwargs.keys():
        return min(kwargs["max"], acc)
    return acc
```

```
multiplicacion(1,2,3,4, max=6)
```

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
PS C:\Users\pmver\Desktop\Ejer Python> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/decoradores.py"
```

```
El resultado de la operación es: 6
```

```
PS C:\Users\pmver\Desktop\Ejer Python> █
```

26.- ¿Cómo usar SQLite3 en Python?

BASE DE DATOS SQL

- Una base de datos puede contener múltiples tablas.
- Cada tabla, guarda elementos con un mismo modelo.



Las tablas

| Nombre | Apellido | Edad |
|---------|-----------|------|
| Alberto | García | 28 |
| María | Parra | 33 |
| Jota | Pelirrojo | 40 |

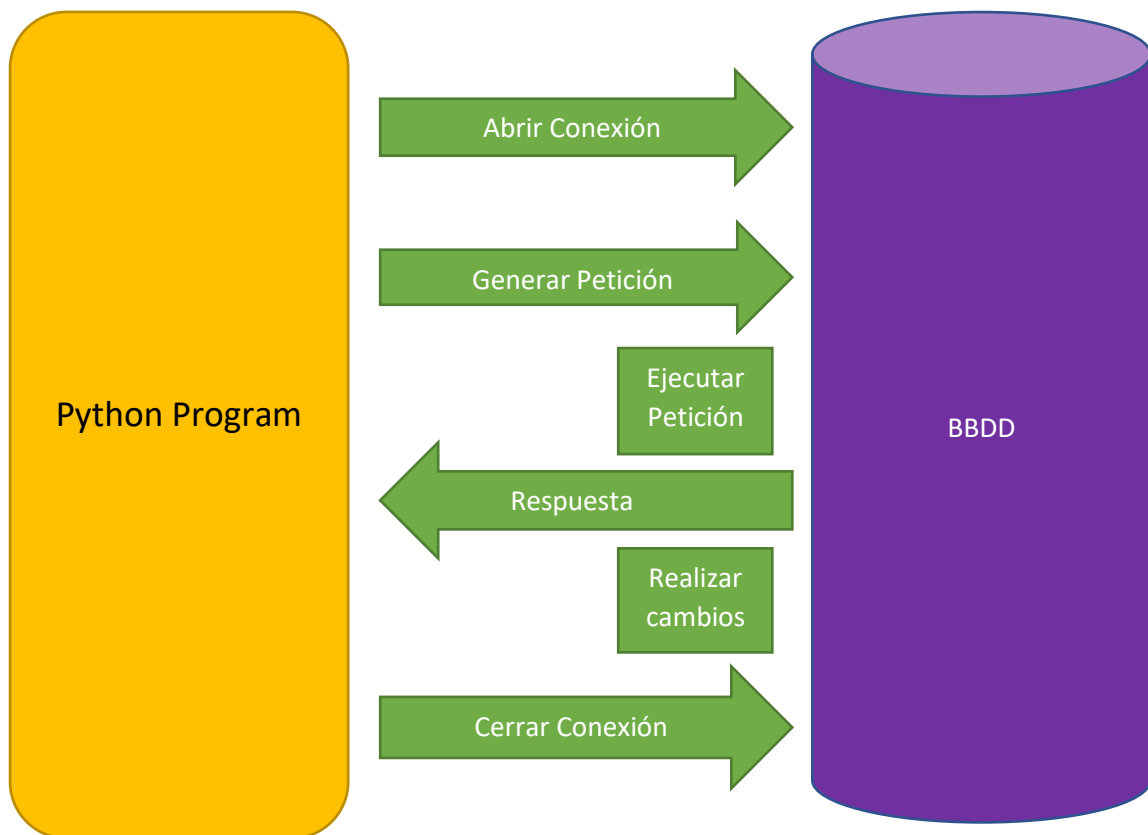
- Una tabla tiene filas y columnas.
- Las columnas son los campos de los elementos.
- Las filas contienen los datos de cada elemento.

El cursor

| Nombre | Apellido | Edad |
|---------|-----------|------|
| Alberto | García | 28 |
| María | Parra | 33 |
| Jota | Pelirrojo | 40 |

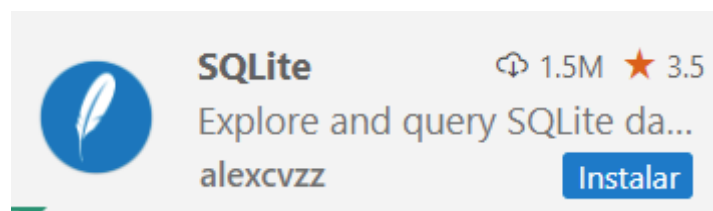
- El cursor selecciona los elementos de una tabla.
- Contiene los datos seleccionados.
- Es el "encargado" de leer o escribir en la tabla.

Comunicación con la base de datos



Vamos con la práctica:

Vamos a agregar la siguiente extensión:



Podremos ver de forma visual las tablas de la base de datos.

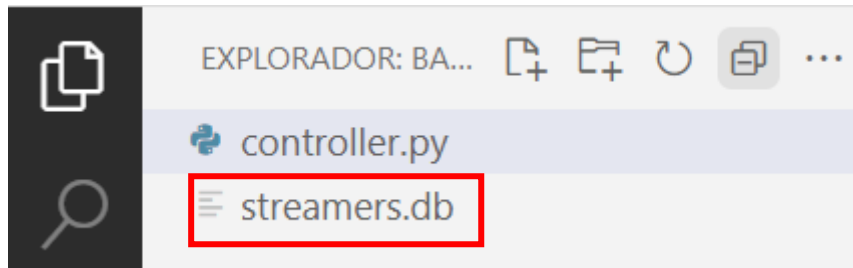
```
1  import sqlite3 as sql
2
3  def createDB():
4      conn = sql.connect("streamers.db")
5      conn.commit()
6      conn.close()
7
```

```

8
9  ✓ if __name__ == "__main__":
10     |     createDB()

```

Vamos a ejecutar:



Ha creado la base de datos.

```

1  import sqlite3 as sql
2
3  def createDB():
4      |     conn = sql.connect("streamers.db")
5      |     conn.commit()
6      |     conn.close()
7
8  def createTable():
9      |     conn = sql.connect("streamers.db")
10     |     cursor = conn.cursor()
11     |     cursor.execute(
12     |         |     """CREATE TABLE streamers (
13     |         |         |     name text,
14     |         |         |     followers integer,
15     |         |         |     subs integer
16     |         |     )"""
17     |     )
18     |     conn.commit()
19     |     conn.close()
20
21
22  if __name__ == "__main__":
23     |     # createDB()
24     |     createTable()

```

Creamos una función para crear una tabla y la ejecutamos:

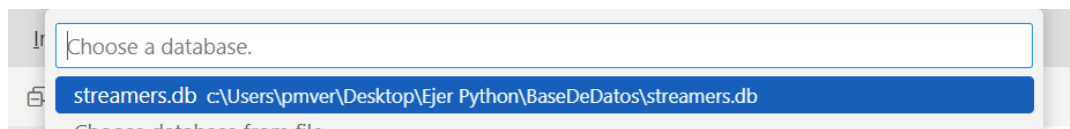
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL P3

```
hon311/python.exe "c:/Users/pmver/Desktop/Ejer Python/BaseDeDatos/controller.py"
PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos>
```

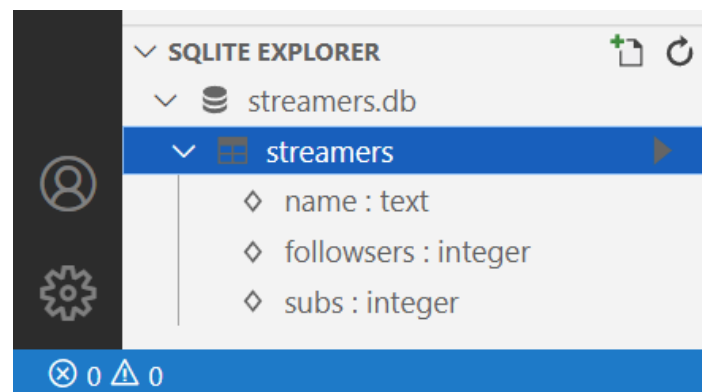
No muestra nada, os recordáis que nos instalamos una extensión al principio de la práctica.

Ahora vamos a seleccionar el código y pulsamos las teclas Ctrl + Shift y P y escribimos sqlite.

Saldrá la opción Open Database.



Seleccionamos la base de datos.



En la parte inferior izquierda podremos ver la estructura de nuestra base de datos, tenemos una base de datos llamada streamers.db con una tabla llamada streamers que contiene los campos name, followers sub.

Insertar un registro:

```
import sqlite3 as sql
def createDB():
    conn = sql.connect("streamers.db")
    conn.commit()
    conn.close()

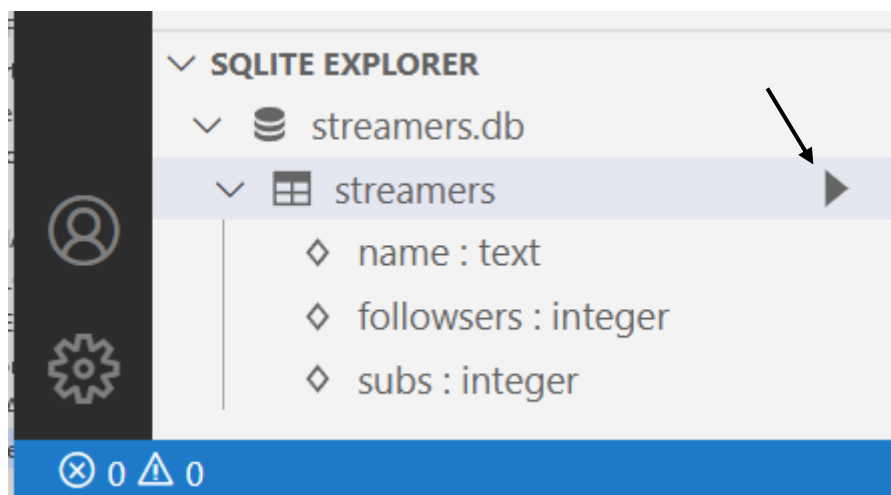
def createTable():
    conn = sql.connect("streamers.db")
    cursor = conn.cursor()
    cursor.execute(
        """CREATE TABLE streamers (
            name text,
            followers integer,
            subs integer
        )"""
    )
```

```
conn.commit()
conn.close()
```

```
def insertRow(nombre, followers, subs):
    conn = sql.connect("streamers.db")
    cursor = conn.cursor()
    instruccion = f"INSERT INTO streamers VALUES ('{nombre}',
{followers}, {subs})"
    cursor.execute(instruccion)
    conn.commit()
    conn.close()
```

```
if __name__ == "__main__":
    # createDB()
    # createTable()
    insertRow("Ibai", 700000, 25000)
```

Vamos a ejecutar:



Le damos el play.



SQL ▾ < 1 / 1 > 1 - 1 of 1   



| name | followers | subs |
|------|-----------|-------|
| Ibai | 700000 | 25000 |

Nos mostrará el primer registro que hemos introducido.

```
30 if __name__ == "__main__":
31     # createDB()
32     # createTable()
33     # insertRow("Ibai", 700000, 25000)
34     insertRow("AlexElCapo", 800000, 10000)
```

Agregamos un nuevo registro.



SQL ▾ < 1 / 1 > 1 - 2 of 2





| name | followers | subs |
|------------|-----------|-------|
| Ibai | 700000 | 25000 |
| AlexElCapo | 800000 | 10000 |

Leer columnas:

```
29 def readRows():
30     conn = sql.connect("streamers.db")
31     cursor = conn.cursor()
32     instruccion = f"SELECT * FROM streamers"
33     cursor.execute(instruccion)
34     datos = cursor.fetchall()
35     conn.commit()
36     conn.close()
37     print(datos)
```

```
39 ▾ if __name__ == "__main__":
40     # createDB()
41     # createTable()
42     # insertRow("Ibai", 700000, 25000)
43     # insertRow("AlexElCapo", 800000, 10000)
44     readRows()
```

Ejecutamos:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  Python + v [ ] [ ] ^ x

PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos> & C:/Users/pmver/AppData/
Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Pyth
on/BaseDeDatos/controller.py"
[('Ibai', 700000, 25000), ('AlexElCapo', 800000, 10000)]
PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos> █
```

Esto son los dos registros que tiene la tabla streamers. Obtenemos una lista y dentro las tuplas 1 por cada registro.

Añadir varios registros a la vez

```
39 def insertRows(streamerList):
40     conn = sql.connect("streamers.db")
41     cursor = conn.cursor()
42     instruccion = f"INSERT INTO streamers VALUES (?, ?, ?)"
43     cursor.executemany(instruccion, streamerList)
44     datos = cursor.fetchall()
45     conn.commit()
46     conn.close()
47
48 if __name__ == "__main__":
49     # createDB()
50     # createTable()
51     # insertRow("Ibai", 700000, 25000)
52     # insertRow("AlexElCapo", 800000, 10000)
53     # readRows()
54     streamers = [
55         ("ElXokas", 1000000, 9500),
56         ("Cristinini", 3000000, 5500),
57         ("Auronplay", 8000000, 20000)
58     ]
59     insertRows(streamers)
```

Vamos a ejecutar.

Vamos a la base de datos:

| name | followers | subs |
|------------|-----------|-------|
| Ibai | 700000 | 25000 |
| AlexElCapo | 800000 | 10000 |
| ElXokas | 1000000 | 9500 |
| Cristinini | 3000000 | 5500 |
| Auronplay | 8000000 | 20000 |

```

48  if __name__ == "__main__":
49      # createDB()
50      # createTable()
51      # insertRow("Ibai", 700000, 25000)
52      # insertRow("AlexElCapo", 800000, 10000)
53      #readRows()
54      streamers = [
55          ("ElXokas", 1000000, 9500),
56          ("Cristinini", 3000000, 5500),
57          ("Auronplay", 8000000, 20000)
58      ]
59      #insertRows(streamers)
60      readRows()

```

Vamos a leer todos los registros

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python + - [] [X] ^

```

PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/BaseDeDatos/controller.py"
[('Ibai', 700000, 25000), ('AlexElCapo', 800000, 10000), ('ElXokas', 1000000, 9500), ('Cristinini', 3000000, 5500), ('Auronplay', 8000000, 20000)]
PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos>

```

Quiero leer la base de datos ordenada por el número de subscriptores, vamos a crear otra función.

```

48  def readOrded(field):
49      conn = sql.connect("streamers.db")
50      cursor = conn.cursor()
51      instruccion = f"SELECT * FROM streamers ORDER BY {field}"
52      cursor.execute(instruccion)
53      datos = cursor.fetchall()
54      conn.commit()
55      conn.close()
56      print(datos)
57
58  if __name__ == "__main__":
59      # createDB()
60      # createTable()
61      # insertRow("Ibai", 700000, 25000)
62      # insertRow("AlexElCapo", 800000, 10000)
63      #readRows()

```

```

64     streamers = [
65         ("ElXokas", 1000000, 9500),
66         ("Cristinini", 3000000, 5500),
67         ("Auronplay", 8000000, 20000)
68     ]
69     #insertRows(streamers)
70     #readRows()
71     readOrdered("subs")

```

Vamos a ejecutar:

```

PROBLEMAS  SALIDA  TERMINAL  ...  Python  + v  [ ]  [X]  ^  X
PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/BaseDeDatos/controller.py"
[('Cristinini', 3000000, 5500), ('ElXokas', 1000000, 9500), ('AlexElCapo', 800000, 10000), ('Auronplay', 8000000, 20000), ('Ibai', 700000, 25000)]
PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos>

```

Vamos a ver el número de suscriptores.

Estos están ordenados de menor a mayor número de suscriptores.

Si queremos ordenar en modo decreciente.

```

48 def readOrdered(field):
49     conn = sql.connect("streamers.db")
50     cursor = conn.cursor()
51     instruccion = f"SELECT * FROM streamers ORDER BY {field} DESC"
52     cursor.execute(instruccion)
53     datos = cursor.fetchall()
54     conn.commit()
55     conn.close()
56     print(datos)

```

Vamos a ejecutar:

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  Python  + v  [ ]  [X]  ^  X
PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/BaseDeDatos/controller.py"
[('Ibai', 700000, 25000), ('Auronplay', 8000000, 20000), ('AlexElCapo', 800000, 10000), ('ElXokas', 1000000, 9500), ('Cristinini', 3000000, 5500)]
PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos>

```

Vamos a buscar un dato en la base de datos

```
58 def search():
59     conn = sql.connect("streamers.db")
60     cursor = conn.cursor()
61     instruccion = f"SELECT * FROM streamers WHERE name='AlexElCapo' "
62     cursor.execute(instruccion)
63     datos = cursor.fetchall()
64     conn.commit()
65     conn.close()
66     print(datos)
```

```
68 if __name__ == "__main__":
69     # createDB()
70     # createTable()
71     # insertRow("Ibai", 700000, 25000)
72     # insertRow("AlexElCapo", 800000, 10000)
73     # readRows()
74     streamers = [
75         ("ElXokas", 1000000, 9500),
76         ("Cristinini", 3000000, 5500),
77         ("Auronplay", 8000000, 20000)
78     ]
79     #insertRows(streamers)
80     #readRows()
81     #readOrded("subs")
82     search()
```

Ejecutamos:

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  Python + - [ ] [X] ^ X

PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos> & C:/Users/pmver/AppData/
Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Pyth
on/BaseDeDatos/controller.py"
[('AlexElCapo', 800000, 10000)]
PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos> █
```

Muestra el correspondiente registro.

```
58 def search(name):
59     conn = sql.connect("streamers.db")
60     cursor = conn.cursor()
61     instruccion = f"SELECT * FROM streamers WHERE name='{name}' "
62     cursor.execute(instruccion)
63     datos = cursor.fetchall()
64     conn.commit()
65     conn.close()
66     print(datos)
```

Como parámetro el nombre a buscar.

```

68 if __name__ == "__main__":
69     # createDB()
70     # createTable()
71     # insertRow("Ibai", 700000, 25000)
72     # insertRow("AlexElCapo", 800000, 10000)
73     #readRows()
74     streamers = [
75         ("ElXokas", 1000000, 9500),
76         ("Cristinini", 3000000, 5500),
77         ("Auronplay", 8000000, 20000)
78     ]
79     #insertRows(streamers)
80     #readRows()
81     #readOrded("subs")
82     search("ElXokas")

```

Como argumento le pasamos "ElXokas"

Este será el resultado:

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python + - [] [] [] [] [] []

```

PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/BaseDeDatos/controller.py"
[('ElXokas', 1000000, 9500)]
PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos>

```

Ten en cuenta que si alguna letra la cambiamos a mayúsculas o minúsculas no lo va a encontrar, para evitar esto vamos a modificar la función:

```

58 def search(name):
59     conn = sql.connect("streamers.db")
60     cursor = conn.cursor()
61     instruccion = f"SELECT * FROM streamers WHERE name like '{name}' "
62     cursor.execute(instruccion)
63     datos = cursor.fetchall()
64     conn.commit()
65     conn.close()
66     print(datos)

```

Llamaremos a la función pasándoles los argumentos cambiados de mayúsculas a minúsculas.

```

82 search("elxokas")

```

Vamos a ejecutar.

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python + - [] [] [] [] [] []

```

PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/pmver/Desktop/Ejer Python/BaseDeDatos/controller.py"
[('ElXokas', 1000000, 9500)]
PS C:\Users\pmver\Desktop\Ejer Python\BaseDeDatos>

```

Lo ha encontrado. Like no tiene distinción entre mayúsculas y minúsculas.

Si queremos encontrar un nombre que empiece por Alex lo escribiremos de la siguiente forma:

"SELECT * FROM streamers WHERE name like 'Alex%'".

Si queremos encontrar valores mayores a una cantidad:

"SELECT * FROM streamers WHERE subs > 6000" para menor es signo <.

Podemos concatenar condición y ordenar:

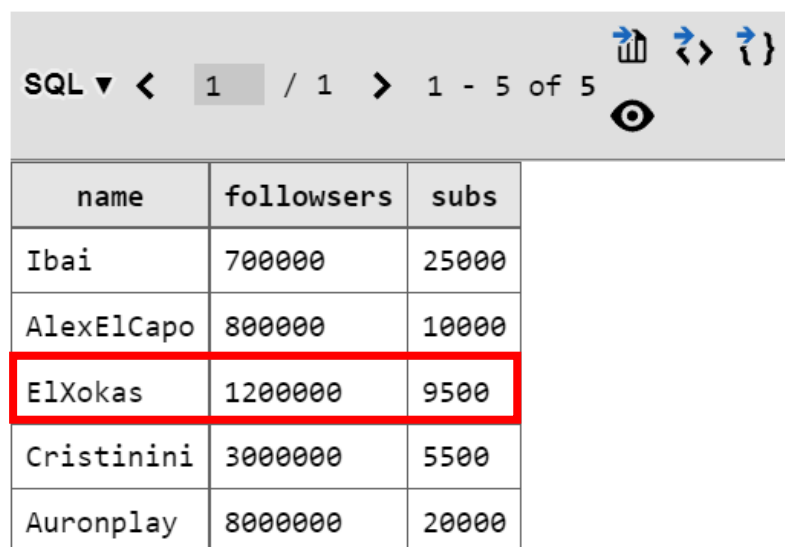
"SELECT * FROM streamers WHERE subs > 6000 ORDER BY subs DESC"

Ahora vamos a actualizar un registro

```
68 def updateFields():
69     conn = sql.connect("streamers.db")
70     cursor = conn.cursor()
71     instruccion = f"UPDATE streamers SET followers=1200000 WHERE name like 'elxokas'"
72     cursor.execute(instruccion)
73     conn.commit()
74     conn.close()

77 if __name__ == "__main__":
78     # createDB()
79     # createTable()
80     # insertRow("Ibai", 700000, 25000)
81     # insertRow("AlexElCapo", 800000, 10000)
82     #readRows()
83     streamers = [
84         ("ElXokas", 1000000, 9500),
85         ("Cristinini", 3000000, 5500),
86         ("Auronplay", 8000000, 20000)
87     ]
88     #insertRows(streamers)
89     #readRows()
90     #readOrded("subs")
91     #search("elxokas")
92     updateFields()
```

Ejecutamos:



SQL ▾ < 1 / 1 > 1 - 5 of 5

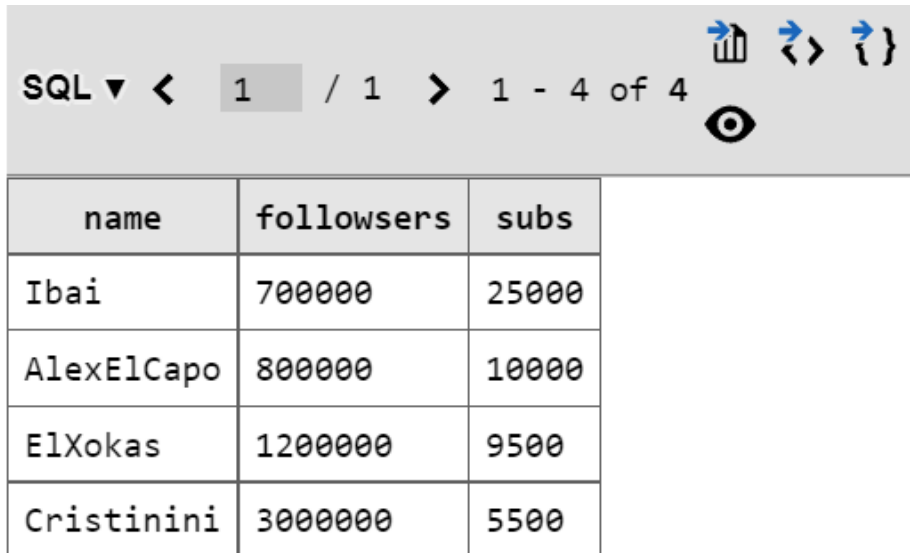
| name | followers | subs |
|------------|-----------|-------|
| Ibai | 700000 | 25000 |
| AlexElCapo | 800000 | 10000 |
| ElXokas | 1200000 | 9500 |
| Cristinini | 3000000 | 5500 |
| Auronplay | 8000000 | 20000 |

Vamos a borrar registros

```
76 def deleteRow():
77     conn = sql.connect("streamers.db")
78     cursor = conn.cursor()
79     instruccion = f"DELETE FROM streamers WHERE name ='Auronplay'"
80     cursor.execute(instruccion)
81     conn.commit()
82     conn.close()

84 if __name__ == "__main__":
85     # createDB()
86     # createTable()
87     # insertRow("Ibai", 700000, 25000)
88     # insertRow("AlexElCapo", 800000, 10000)
89     #readRows()
90     streamers = [
91         ("ElXokas", 1000000, 9500),
92         ("Cristinini", 3000000, 5500),
93         ("Auronplay", 8000000, 20000)
94     ]
95     #insertRows(streamers)
96     #readRows()
97     #readOrded("subs")
98     #search("elxokas")
99     #updateFields()
100    deleteRow()
```

Ejecutamos:



| name | followers | subs |
|------------|-----------|-------|
| Ibai | 700000 | 25000 |
| AlexElCapo | 800000 | 10000 |
| ElXokas | 1200000 | 9500 |
| Cristinini | 3000000 | 5500 |

El registro de Auronplay ya se ha eliminado.

Contenido

| | |
|--|-----|
| 1.- Introducción | 1 |
| 1.- ¿Qué es Python? | 1 |
| 2.- ¿Para qué sirve Python?..... | 1 |
| 3.- ¿Cómo tomar este curso? | 1 |
| Instalar Python con la última versión..... | 1 |
| 2.- Tipo de DATOS, VARIABLES y Operaciones Aritméticas en Python. | 8 |
| Variables..... | 8 |
| Tipos de datos | 8 |
| 3.- ¿Cómo usar CONDICIONALES en Python? | 13 |
| 4.- Funciones en Python con ejercicios resueltos. | 17 |
| 5.- ¿Cómo hacer un BUCLE en PYTHON? (Con ejemplos incluidos) | 21 |
| 6.- Listas en Python | 33 |
| 7.- Ejercicio | 38 |
| 8.- ¿Qué es una CADENA en Python?..... | 40 |
| 9.- ¿Qué son los MÉTODOS de CADENAS en Python? | 43 |
| 10.- ¿Cómo funciona un DICCIONARIO en Python? | 48 |
| 11.- ¿Qué es *ARGS y **KWARGS en Python? Funciones con argumentos opcionales | 53 |
| 12.- LAS TUPLAS (Sintaxis Básica)..... | 57 |
| 13.- ¿Qué es la Programación Orientada a Objetos en Python? | 63 |
| 14.- ¿Qué es la HERENCIA en Python? | 71 |
| 15.- ¿Qué es el ENCAPSULAMIENTO en Python?..... | 78 |
| 16.- ¿Qué es el POLIMORFISMO en Python? | 86 |
| 17.- ABSTRACCIÓN en Python | 88 |
| 18.- Definir Clases con Ejemplos | 94 |
| 19.- Excepciones en Python | 106 |
| 20.- Funciones LAMBDA..... | 113 |
| 21.- Usar las Funciones MAP y FILTER en Python | 116 |
| 22.- ¿Qué son los MÓDULOS y PAQUETES en Python?..... | 121 |
| 23.- Archivos Externos [LEER y ESCRIBIR ficheros.txt] + Ejercicio resuelto..... | 126 |
| 24.- Interfaces gráficas con TKINTER..... | 136 |
| 25.- ¿Qué son los DECORADORES en Python? | 145 |
| 26.- ¿Cómo usar SQLite3 en Python?..... | 153 |