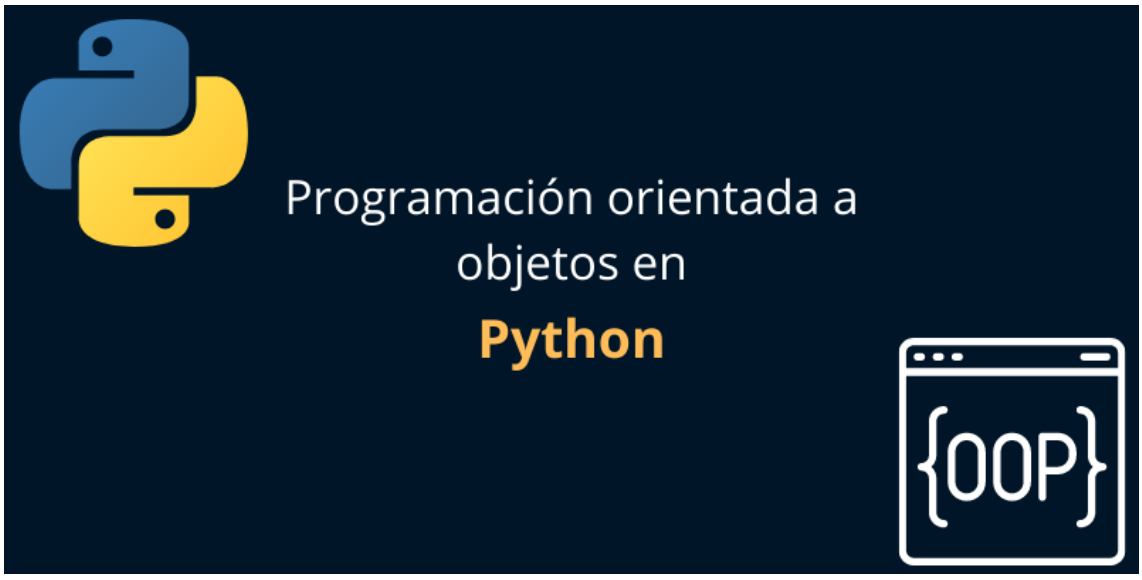


PYTHON POO

PROGRAMACIÓN ORIENTADO A OBJETO

PERE MANEL VERDUGO ZAMORA

www.peremanelv.com

Contenido

1.- Clases y objetos

Programación orientada a objetos:

Clases -> Objetos

- ➔ Atributos
- ➔ Métodos

Herencias

Polimorfismo

Encapsulamiento

Instancia de clase

Variables de clase

Variables de instancia

__init__ __str__

Super() __var

- ➔ Mucho menos código de escribir
- ➔ Mucho más organizado el código
- ➔ Mucho más fácil modificarlo o ampliarlo

```
La ciudad se ha llenado de zombis.
Estás en la calle 1 y has de llegar
a la calle 40 para poder salvarte.

Los zombis avanzan 1, 2 ó 3 calles.
Tú puedes avanzar 1, 2 ó 3 calles.

¿Estás preparado? Cuál es tu nombre:
```

```
Jose, estás en la calle:  1

Hay zombis en las calles:

12 12 12 15 16 16 17 17 20 20

Cuánto quieres correr (1/2/3):
```

```
Jose, estás en la calle:  4

Hay zombis en las calles:

9 13 14 14 15 16 16 18 18 22

Cuánto quieres correr (1/2/3):
```

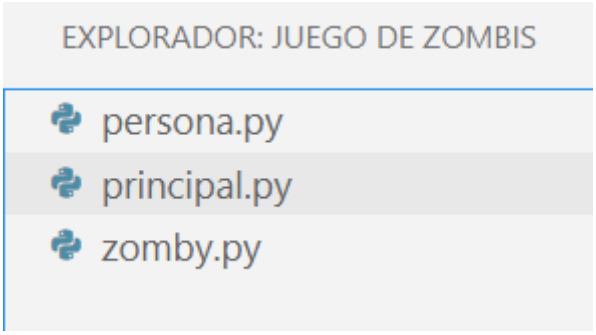
```
Jose, estás en la calle:  7

Hay zombis en las calles:

6 11 13 14 14 16 16 18 19 24

Cuánto quieres correr (1/2/3):
```

```
Jose, estás en la calle: 10

Hay zombis en las calles:

5 10 12 13 15 16 19 20 21 27

Un zombi te acaba de ver.
Te va a comer. Se acabó el juego.
```

Vamos a crear una carpeta llamada “Juego de zombis”

Vamos al visual Studio Code para empezar a programar.

Crearemos tres archivos perona.py, principal.py y zombi.py.



Empezamos con la codificación clase persona, para ello iremos al archivo persona.py

```python
class Persona:
    pass

jose = Persona()

print(jose)
```

Ejecutamos:

```
PS F:\Juego de zombis> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "f:/Juego de zombis/persona.py"
<__main__.Persona object at 0x0000026887A6B590>
PS F:\Juego de zombis>
```

Vemos un objeto de tipo persona, alojado en la siguiente dirección de memoria.

Esta clase no nos sirve para mucho porque no contiene ninguna información.

Las clases tienen dos tipos de atributos, por una parte las características, los datos y por otra parte las funcionalidades que son los comportamientos de los objetos que pertenece a esta clase que son los métodos.

```
1  class Persona:
2      pass
3
4  jose = Persona()
5  jose.nombre = "Jose"
6  print(jose.nombre)
```

Si a jose le damos un nombre, cuando ejecutemos:

```
PS F:\Juego de zombis> & C:/Users/pmver/AppData/Local/Pr
ograms/Python/Python311/python.exe "f:/Juego de zombis/p
ersona.py"
Jose
PS F:\Juego de zombis>
```

Nos muestro el nombre el objeto Jose.

```
1  class Persona:
2      pass
3
4  jose = Persona()
5  juan = Persona()
6
7  jose.nombre = "Jose"
8  juan.calle = 5
9
10 print(jose.nombre)
11 print(juan.calle)
```

Si ejecutamos:

```
PS F:\Juego de zombis> & C:/Users/pmver/AppData/Local/Pr
ograms/Python/Python311/python.exe "f:/Juego de zombis/p
ersona.py"
Jose
5
```

```
PS F:\Juego de zombis>
```

Esta forma de realizarlo no es muy eficiente, hay otra forma de llevarlo a cabo.

```python
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

jose = Persona("Jose")
juan = Persona("Juan")

print(jose.nombre)
print(juan.nombre)
```

Ejecutamos:

```
PS F:\Juego de zombis> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "f:/Juego de zombis/persona.py"
Jose
Juan
PS F:\Juego de zombis>
```

```python
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre
        self.calle = 1

jose = Persona("Jose")
juan = Persona("Juan")

print(jose.calle)
print(juan.calle)
```

Si ejecutamos:

```
PS F:\Juego de zombis> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "f:/Juego de zombis/persona.py"
```

```
1
1
PS F:\Juego de zombis>
```

Los dos son de la calle 1.

```python
class Persona:
    def __init__(self, nombre, calle=1):
        self.nombre = nombre
        self.calle = calle

jose = Persona("Jose")
juan = Persona("Juan",2)

print(jose.calle)
print(juan.calle)
```

La mayoría son de la calle 1 pero se puede dar el caso de que alguno sea de otra calle.

Vamos a ejecutar:

```
PS F:\Juego de zombis> & C:/Users/pmver/AppData/Local/Programs/Python/Python311/python.exe "f:/Juego de zombis/persona.py"
1
2
PS F:\Juego de zombis>
```

Podemos necesitar la acciones.

```python
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre
        self.calle = 1

    def moverse(self, velocidad):
        if velocidad == "1":
            self.calle += 1
        elif velocidad == "2":
```

```
10              self.calle += 2
11          else:
12              self.calle += 3
13
14
15  jose = Persona("Jose")
16  print(jose.calle)
17  jose.moverse(2)
18  print(jose.calle)
```

Vamos a ejecutar:

```
PS F:\Juego de zombis> & C:/Users/pmver/AppData/Local/Pr
ograms/Python/Python311/python.exe "f:/Juego de zombis/p
ersona.py"
1
4
PS F:\Juego de zombis>
```

Y se puede mover por más calles.

2.- Atributos y métodos

```
1    class Persona:
2        def __init__(self, nombre):
3            self.nombre = nombre
4            self.calle = 1
5
6        def moverse(self, velocidad):
7            if velocidad == "1":
8                self.calle += 1
9            elif velocidad == "2":
10               self.calle += 2
11           else:
12               self.calle += 3
```

Así ha de quedar la clase Persona.

Ahora vamos a zombi.py

```
1    import random
2
3    class Zombi:
4        def __init__(self):
5            self.calle = random.randint(10,20)
6            self.direccion = random.choice(["izquierda", "derecha"])
7
8        def moverse(self):
9            if self.direccion == "izquierda":
10               self.calle -= random.randint(1,3)
11           else:
12               self.calle += random.randint(1,3)
13
14       def no_visible(self):
15           if self.calle < 0 or self.calle >40:
16               return True
17           else:
18               return False
```

Creamos un objeto:

```
20   z = Zombi()
21   print(z.calle)
22   print(z.direccion)
```

```
23    z.moverse()
24    print(z.calle)
25    print(z.no_visible())
```

Vamos a ejecutar:

```
PS F:\Juego de zombis> & C:/Users/pmver/AppData/Local/Pr
ograms/Python/Python311/python.exe "f:/Juego de zombis/z
omby.py"
10
derecha
11
False
PS F:\Juego de zombis>
```

Inicialmente se situado en la calle 10, va hacia la derecha, después a saltado 1 calle y al no visible False.

El zombi cada vez cambia de velocidad ¿Cómo podemos hacer para que la velocidad sea fija?.

```
1   import random
2
3   class Zombi:
4       def __init__(self):
5           self.calle = random.randint(10,20)
6           self.direccion = random.choice(["izquierda", "derecha"])
7           self.velocidad = random.randint(1,3)
8
9       def moverse(self):
10          if self.direccion == "izquierda":
11              self.calle -= self.velocidad
12          else:
13              self.calle += self.velocidad
14
15      def no_visible(self):
16          if self.calle < 0 or self.calle >40:
17              return True
18          else:
19              return False
20
```

3.- Creando objetos

Ahora vamos a trabajar con archivo persona.py, que es donde irá el flujo del programa.

```
1   class Persona:
2       def __init__(self, nombre):
3           self.nombre = nombre
4           self.calle = 1
5
6       def situacion(self):
7           return "{}, estás en la calle {}".format(self.nombre, self.calle)
8
9       def moverse(self, velocidad):
10          if velocidad == "1":
11              self.calle += 1
12          elif velocidad == "2":
13              self.calle += 2
14          else:
15              self.calle += 3
```

Agregamos el método seleccionado.

Vamos a principal.py

```
1   from persona import Persona
2   from zombi import Zombi
3   import os
4
5   os.system("cls")
6
7   print()
8   print("  La ciudad se ha llenado de zombis.")
9   print("  Estás en la calle 1 y has de llegar")
10  print("  a la calle 40 para poder salvarte.")
11  print()
12  print("  Los zombis avanzan 1, 2, ó 3 calles.")
13  print("  Tú puedes avanzar 1, 2, ó 3 calles.")
14  print()
15  nombre = input("  ¿Estás preparado? Cúal es tu nombre: ").capitalize()
16
17  jugador = Persona(nombre)
18
19  horda = []
20  for i in range(10):
21      z = Zombi()
22      horda.append(z)
23
24  while True:
25      os.system("cls")
26      print(jugador.situacion())
27      calles =[]
```

Se crean 10 objetos de tipo Zombi().

Las añadimos en una lista llamada horda.

```
28      for zombi in horda:
29          calles.append(zombi.calle)
30      calles.sort()
31      print("Hay zombis en las calles: ")
32      for elemento in calles:
33          print(elemento, end=" ")
34      print()
35      print()
```

Con otro ciclo for los agregamos a la lista calle, que a su ver la ordenamos.

Con un ciclo for mostramos la lista calles sin el salto de línea end=" ".

4.- Llamando a métodos

```
37      if jugador.calle > 40:
38          print("   No te ha visto ningún zombi.")
39          print("   Te has librado de ser comido.")
40          print()
41          break
```

Si el jugador supera la calle 40 ha ganado.

```
43      comido = False
44
45      for zombi in horda:
46          if zombi.calle == jugador.calle:
47              comido = True
```

Aun no te ha comido.

En el siguiente for recorremos todas las calles donde están los zombis y comprobamos si en alguna de ellas nos encontramos nosotros, si es así te han comido.

```
48      if comido:
49          print("   Un zombi te acaba de ver.")
50          print("   Te va a comer. Se acabó el juego")
51          print()
52          break
```

Te dice que te ha comido y al interrumpirse el bucle se termia el juego.

```
54      velocidad = ""
55      while velocidad not in("1", "2", "3"):
56          velocidad = input("   Cuánto quieres corres (1/2/3): ")
57      jugador.moverse(velocidad)
```

Eliminamos el valor de velocidad.

Con este while controlamos que solo contestemos con el valor 1, 2 o 3 de lo contrario te lo volverá a preguntar.

Si la respuesta es correcta, llamaremos al método moverse de la clase persona con el parámetro velocidad.

```
59      for z in list(horda):
60          z.moverse()
61          if z.no_visible():
62              horda.remove(z)
```

Utilizamos el list porque al ser las listas dinámicas y eliminar algún elemento, algún elemento que queramos eliminar se nos escapará.

Con este ciclo for hacemos que todos los zombis llamen al método moverse para que se puedan desplazar.

Unos hacia la derecha y otros hacia la izquierda un salto continuo.

Si el zombi se encuentra fuera de las calles 1 a la 40 vamos a eliminarlo.

Vamos a ejecutar:

```
La ciudad se ha llenado de zombis.
Estás en la calle 1 y has de llegar
a la calle 40 para poder salvarte.

Los zombis avanzan 1, 2, ó 3 calles.
Tú puedes avanzar 1, 2, ó 3 calles.

¿Estás preparado? Cúal es tu nombre: Jose
```

Contestamos por Jose.

```
Jose, estás en la calle 1
Hay zombis en las calles:
10 12 12 16 16 16 18 19 19 20

   Cuánto quieres corres (1/2/3): 3
```

Contestamos por 3.

```
Jose, estás en la calle 4
Hay zombis en las calles:
10 11 13 15 16 17 17 19 20 21

   Cuánto quieres corres (1/2/3): 3
```

Contestamos de nuevo por 3.

```
Jose, estás en la calle 7
Hay zombis en las calles:
8 10 12 14 18 18 18 18 21 23

   Cuánto quieres corres (1/2/3): 3
```

Contestamos de nuevo por 3.

```
Jose, estás en la calle 10
Hay zombis en las calles:
6 7 12 13 17 19 19 21 22 25

   Cuánto quieres corres (1/2/3): 2
```

Contestamos por 2.

```
Jose, estás en la calle 12
Hay zombis en las calles:
4 4 10 14 16 20 20 23 24 27

   Cuánto quieres corres (1/2/3): 3
```

Contestamos por 3.

```
Jose, estás en la calle 15
Hay zombis en las calles:
1 2 8 15 15 21 21 24 27 29

   Un zombi te acaba de ver.
   Te va a comer. Se acabó el juego
```

5.- Más objetos interactuando

En este capítulo vamos a agregar más jugadores, es donde verdaderamente se ve la ventaja de trabajar en Programación Orientada a objetos.

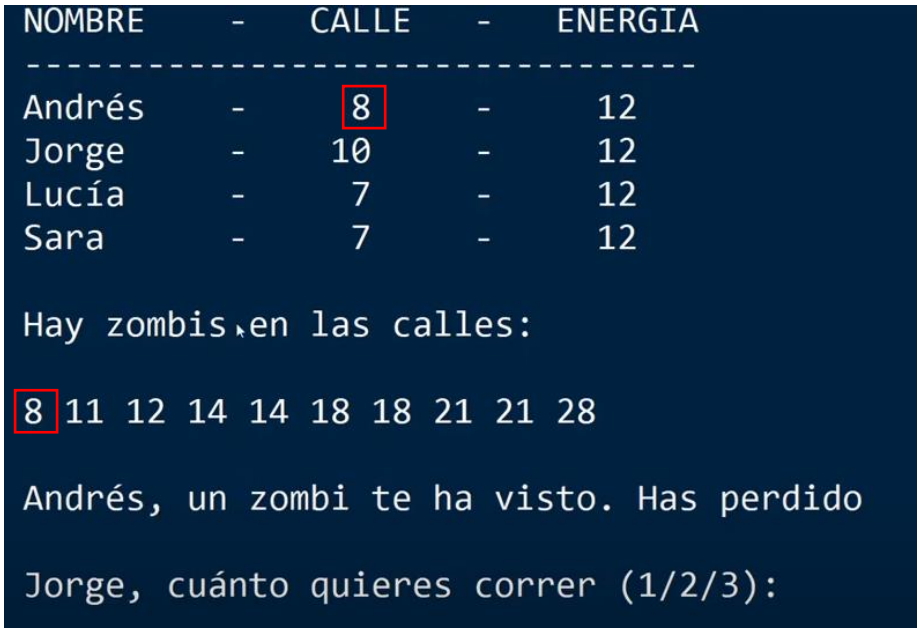
```
La ciudad se ha llenado de zombis.
Estás en la calle 1 y has de llegar
a la calle 40 para poder salvarte.

Los zombis avanzan 1, 2 ó 3 calles.
Tú puedes avanzar 1, 2 ó 3 calles.

Número de jugadores (1/4): 4
```

```
La ciudad se ha llenado de zombis.
Estás en la calle 1 y has de llegar
a la calle 40 para poder salvarte.

Los zombis avanzan 1, 2 ó 3 calles.
Tú puedes avanzar 1, 2 ó 3 calles.

Número de jugadores (1/4): 4
Nombre del 1º jugador: Andrés
Nombre del 2º jugador: Jorge
Nombre del 3º jugador: Lucía
Nombre del 4º jugador: Sara
```

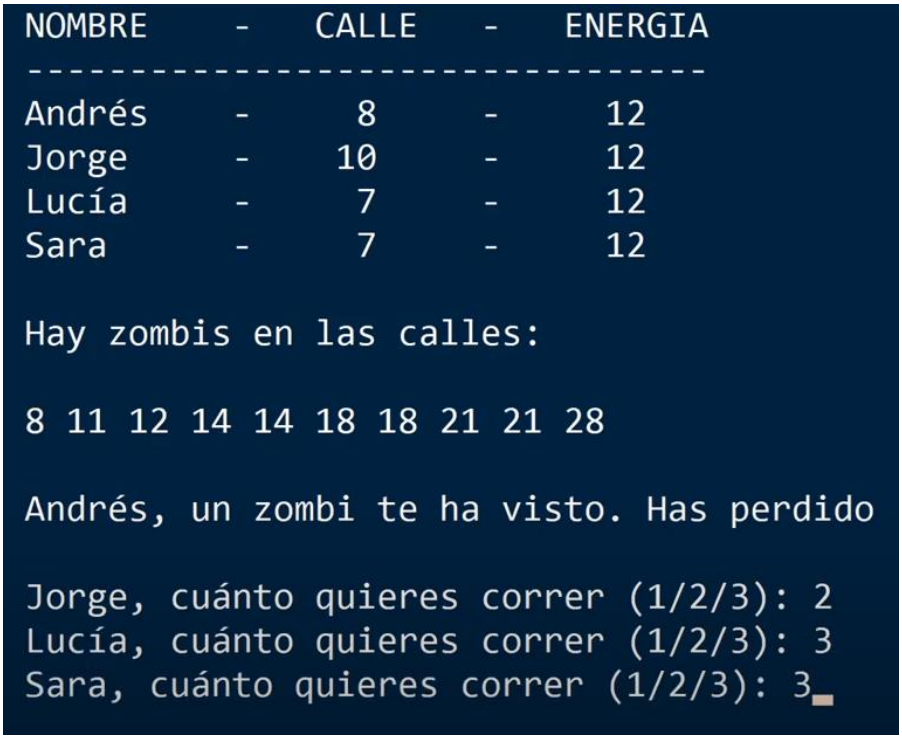
```
NOMBRE    -   CALLE   -   ENERGIA
--------------------------------
Andrés    -     6     -     13
Jorge     -     7     -     13
Lucía     -     5     -     13
Sara      -     6     -     13

Hay zombis en las calles:

11 12 13 14 16 16 16 19 19 25


Andrés, cuánto quieres correr (1/2/3): 2
Jorge, cuánto quieres correr (1/2/3): 3
Lucía, cuánto quieres correr (1/2/3): 2
Sara, cuánto quieres correr (1/2/3): _
```

Preguntará la velocidad de cada uno de los jugadores.

Hemos introducido otra característica que es la energía, por cada movimiento se perderá un punto de energía.

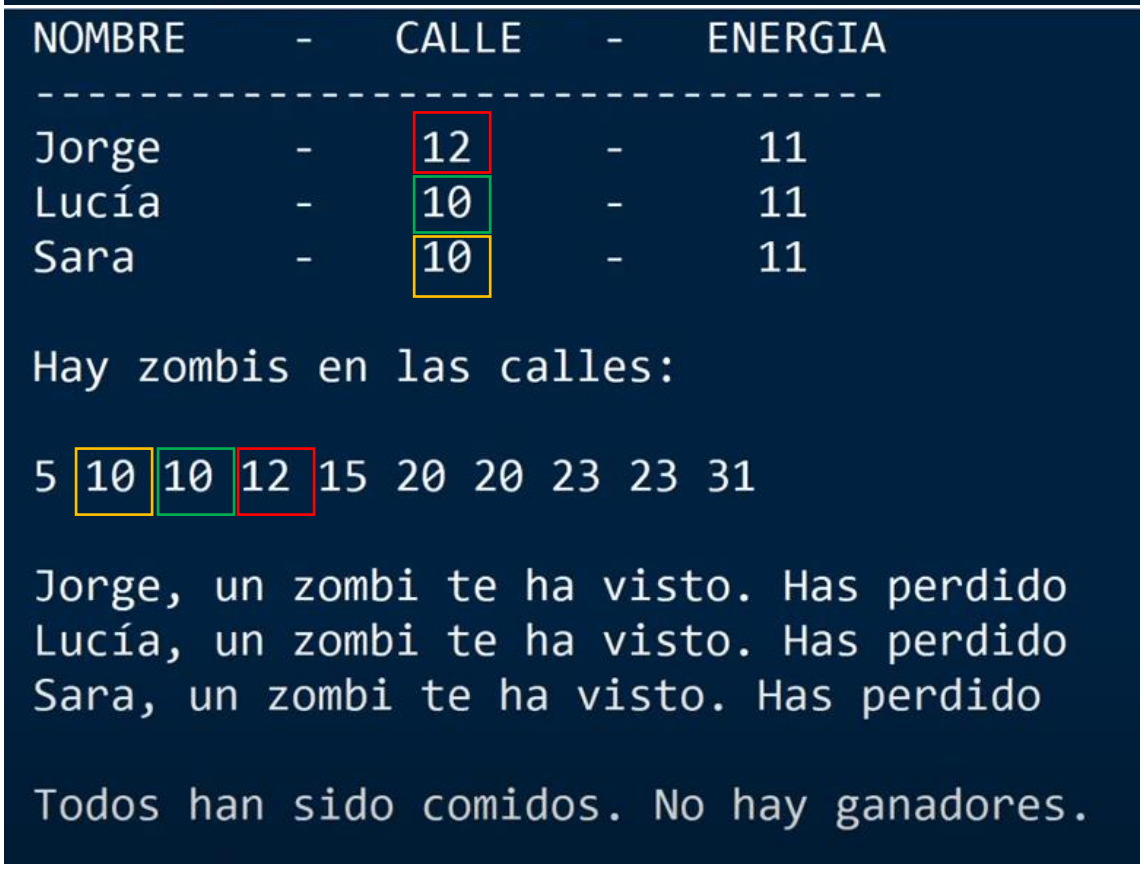


Un zombi ha visto a Andrés.



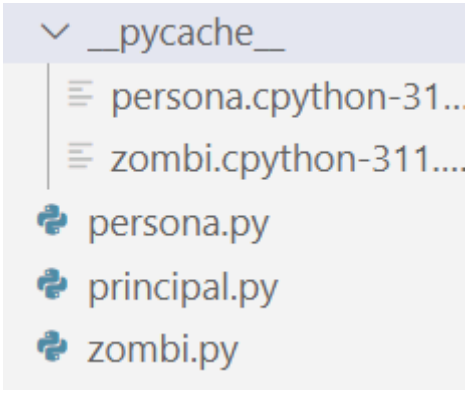
Ahora pregunta por los jugadores que siguen vivos.

```
NOMBRE     -    CALLE    -    ENERGIA
-----------------------------------
Jorge      -     12      -      11
Lucía      -     10      -      11
Sara       -     10      -      11

Hay zombis en las calles:

5 10 10 12 15 20 20 23 23 31

Jorge, un zombi te ha visto. Has perdido
Lucía, un zombi te ha visto. Has perdido
Sara, un zombi te ha visto. Has perdido

Todos han sido comidos. No hay ganadores.
```



Todos los jugadores han perdido.

Vamos a copiar la carpeta para trabajar con la versión 2.



La carpeta _pycche_ ha aparecido automáticamente porque Python lleva una compilación de los archivos que se importan para optimizar la ejecución del archivo principal.

```
1   class Persona:
2       def __init__(self, nombre):
3           self.nombre = nombre
4           self.calle = 1
5           self.energia = 15
```

En la clase persona le hemos agregado un nuevo atributo llamado energía igual a 15.

```
7       def situacion(self):
8           return self.nombre, self.calle, self.energia
```

Hemos cambiado el método situación.

```
10      def moverse(self, velocidad):
11          if velocidad == "1":
12              self.calle += 1
13          elif velocidad == "2":
14              self.calle += 2
15          else:
16              self.calle += 3
17          self.energia -= 1
```

En el método moverse a la energía le restamos una unidad.

La clase Zombi se deja igual.

Ahora nos vamos al archivo principal.py.

```
16  numero = ""
17  while numero not in ("1", "2", "3", "4"):
18      numero = input(" Número de jugadores (1/4): ")
```

Hemos introducido un opción para preguntar por el número de jugadores.

```
20  jugadores = []
21  for i in range(1, int(numero)+1):
22      nombre = input("  Nombre del " + str(i) + "º jugador: ").capitalize()
23      jugador = Persona(nombre)
24      jugadores.append(jugador)
```

Muestra por consola el nombre la calle y la energía de cada jugador.

```
31  while len(jugadores) > 0:
32      os.system("cls")
33
34      print(" NOMBRE    -    CALLE    -    ENERGIA")
35      print(" --------------------------------")
36      for jugador in jugadores:
37          nom, cal, ene = jugador.situacion()
38          print("   {:8}  -  {:2}  -  {:2}".format(nom, cal, ene))
39      print()
```

Ponemos :8 para asignar un número mínimo de caracteres, así las columnas se alinean.

```
51      ganadores = []
52      for jugador in jugadores:
53          if jugador.calle >= 40:
54              ganadores.append(jugador)
55      if len(ganadores)>0:
56          for jugador in ganadores:
57              print("   {}, has escapado de los zombis.".format(jugador.nombre))
58          print("   Has/habéis ganado la partida.")
59          print()
60          break
```

En la lista ganadores se guardan todos los jugadores que han llegado a la calle 40.

Si hay algún jugador que lo muestre por consola.

Se termina el juego.

Cuando el jugador pierde la energía:

```
62      for jugador in list(jugadores):
63          if jugador.energia <= 0:
64              print("  {}, has perdido toda la energía. Estás comido.".format(jugador.nombre))
65              jugadores.remove(jugador)
```

Si un jugador se ha encontrado con un zombi, “Has perdido”.

```
67      for jugador in list(jugadores):
68          for zombi in horda:
69              if zombi.calle == jugador.calle:
70                  if jugador in jugadores:
71                      if jugador in jugadores:
72                          print("  {}, un zombi te ha visto. Has perdido".format(jugador.nombre))
73                          jugadores.remove(jugador)
```

Pedimos a los jugadores que aun sobreviven la nueva velocidad.

```
75      print()
76      for jugador in jugadores:
77          velocidad = ""
78          while velocidad not in("1", "2", "3"):
79              velocidad = input("   Cuánto quieres corres (1/2/3): ".format(jugador.nombre))
80          jugador.moverse(velocidad)
```

Adjunto todo el código del archivo principal.py.

```
from persona import Persona
from zombi import Zombi
import os

os.system("cls")

print()
print("  La ciudad se ha llenado de zombis.")
print("  Estás en la calle 1 y has de llegar")
print("  a la calle 40 para poder salvarte.")
print()
print("  Los zombis avanzan 1, 2, ó 3 calles.")
print("  Tú puedes avanzar 1, 2, ó 3 calles.")
print()

numero = ""
while numero not in ("1", "2", "3", "4"):
    numero = input(" Número de jugadores (1/4): ")

jugadores = []
for i in range(1, int(numero)+1):
    nombre = input("  Nombre del " + str(i) + "º jugador: ").capitalize()
    jugador = Persona(nombre)
    jugadores.append(jugador)

horda = []
for i in range(10):
    z = Zombi()
    horda.append(z)

while len(jugadores) > 0:
    os.system("cls")

    print(" NOMBRE   -   CALLE   -   ENERGIA")
    print(" --------------------------------")
    for jugador in jugadores:
        nom, cal, ene = jugador.situacion()
        print(" {:8}  -    {:2}     -     {:2}".format(nom, cal, ene))
    print()

    calles =[]
    for zombi in horda:
```

```
            calles.append(zombi.calle)
        calles.sort()
        print("Hay zombis en las calles: ")
        for elemento in calles:
            print(elemento, end=" ")
        print()
        print()

        ganadores = []
        for jugador in jugadores:
            if jugador.calle >= 40:
                ganadores.append(jugador)
        if len(ganadores)>0:
            for jugador in ganadores:
                print("   {}, has escapado de los
zombis.".format(jugador.nombre))
            print("   Has/habéis ganado la partida.")
            print()
            break

        for jugador in list(jugadores):
            if jugador.energia <= 0:
                print("  {}, has perdido toda la energía. Estás
comido.".format(jugador.nombre))
                jugadores.remove(jugador)

        for jugador in list(jugadores):
            for zombi in horda:
                if zombi.calle == jugador.calle:
                    if jugador in jugadores:
                        if jugador in jugadores:
                            print("  {}, un zombi te ha visto. Has
perdido".format(jugador.nombre))
                            jugadores.remove(jugador)

        print()
        for jugador in jugadores:
            velocidad = ""
            while velocidad not in("1", "2", "3"):
                velocidad = input("  {} Cuánto quieres corres (1/2/3):
".format(jugador.nombre))
            jugador.moverse(velocidad)

        for z in list(horda):
            z.moverse()
            if z.no_visible():
                horda.remove(z)

else:
```

```
print(" Todos han sido comidos. No hay ganadores.")
print()
```

```
La ciudad se ha llenado de zombis.
Estás en la calle 1 y has de llegar
a la calle 40 para poder salvarte.

Los zombis avanzan 1, 2, ó 3 calles.
Tú puedes avanzar 1, 2, ó 3 calles.

Número de jugadores (1/4): 4
```

```
La ciudad se ha llenado de zombis.
Estás en la calle 1 y has de llegar
a la calle 40 para poder salvarte.

Los zombis avanzan 1, 2, ó 3 calles.
Tú puedes avanzar 1, 2, ó 3 calles.

Número de jugadores (1/4): 4
 Nombre del 1º jugador: juan
 Nombre del 2º jugador: luis
 Nombre del 3º jugador: pedro
 Nombre del 4º jugador: antonio
```

```
NOMBRE   -   CALLE   -   ENERGIA
--------------------------------
Juan     -     1     -     15
Luis     -     1     -     15
Pedro    -     1     -     15
Antonio  -     1     -     15

Hay zombis en las calles:
10 12 14 15 16 17 17 18 18 20
```

```
NOMBRE   -   CALLE   -   ENERGIA
--------------------------------
Juan     -     3     -     14
Luis     -     2     -     14
Pedro    -     4     -     14
Antonio  -     3     -     14

Hay zombis en las calles:
11 13 14 14 14 16 17 18 19 23
```

```
Juan Cuánto quieres corres (1/2/3): 2
Luis Cuánto quieres corres (1/2/3): 1
Pedro Cuánto quieres corres (1/2/3): 3
Antonio Cuánto quieres corres (1/2/3): 2
```

```
Juan Cuánto quieres corres (1/2/3): 3
Luis Cuánto quieres corres (1/2/3): 2
Pedro Cuánto quieres corres (1/2/3): 1
Antonio Cuánto quieres corres (1/2/3): 3
```

```
NOMBRE   -   CALLE   -   ENERGIA
--------------------------------
Juan     -     6     -     13
Luis     -     4     -     13
Pedro    -     5     -     13
Antonio  -     6     -     13

Hay zombis en las calles:
11 12 12 12 16 16 17 19 20 26
```

```
NOMBRE   -   CALLE   -   ENERGIA
--------------------------------
Juan     -     8     -     12
Luis     -     5     -     12
Pedro    -     8     -     12
Antonio  -     8     -     12

Hay zombis en las calles:
8 10 11 13 15 18 18 20 21 29
```

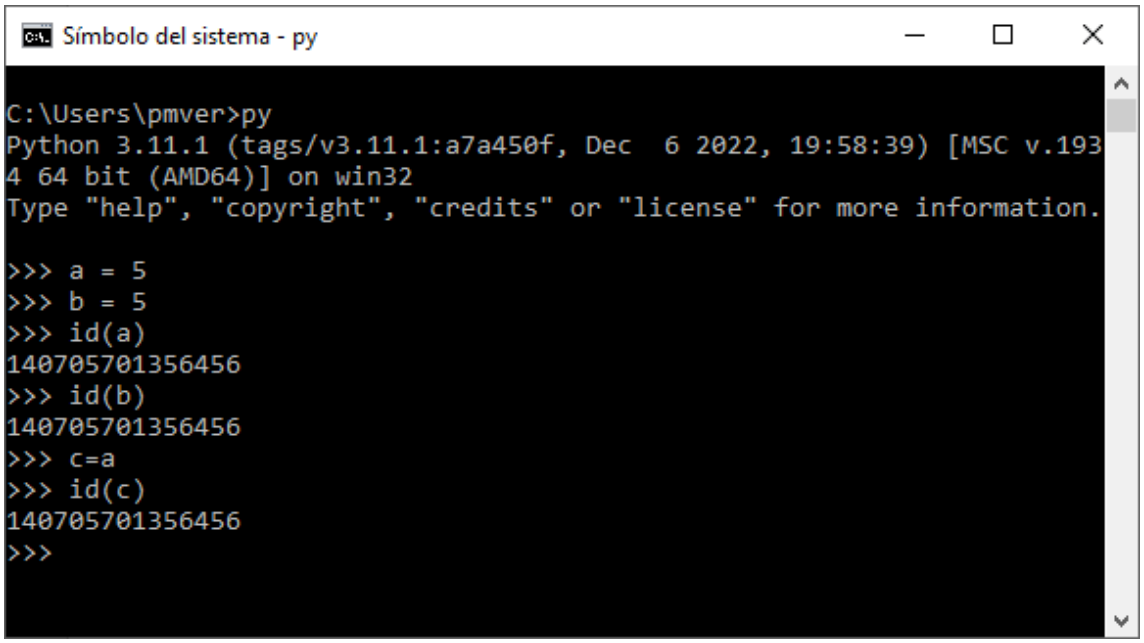
```
Juan Cuánto quieres corres (1/2/3): 2
Luis Cuánto quieres corres (1/2/3): 1
Pedro Cuánto quieres corres (1/2/3): 3
Antonio Cuánto quieres corres (1/2/3): 2
```

```
Juan, un zombi te ha visto. Has perdido
Pedro, un zombi te ha visto. Has perdido
Antonio, un zombi te ha visto. Has perdido

Luis Cuánto quieres corres (1/2/3):
```

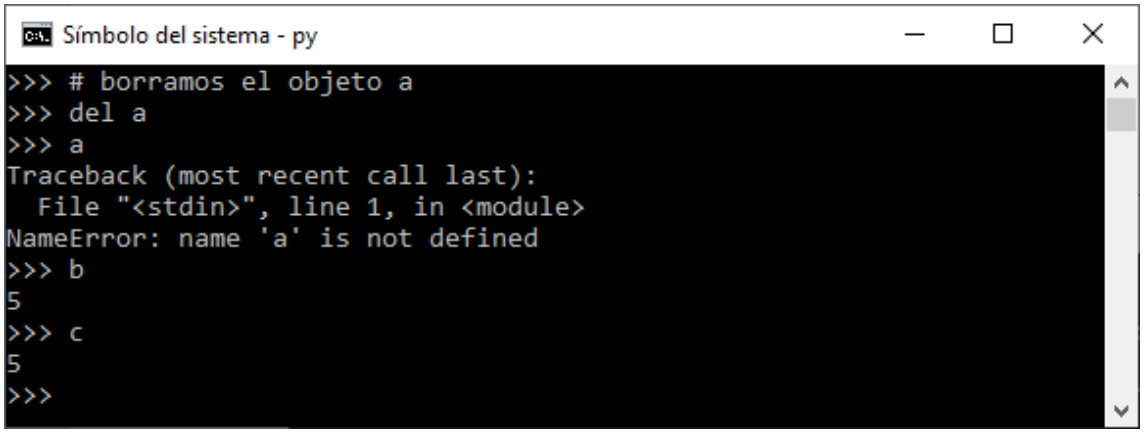
```
 NOMBRE    -   CALLE    -   ENERGIA
 -------------------------------
 Juan      -     8      -     12
 Luis      -     5      -     12
 Pedro     -     8      -     12
 Antonio   -     8      -     12

Hay zombis en las calles:
8 10 11 13 15 18 18 20 21 29

  Juan, un zombi te ha visto. Has perdido
  Pedro, un zombi te ha visto. Has perdido
  Antonio, un zombi te ha visto. Has perdido

  Luis Cuánto quieres corres (1/2/3): 3
```

```
 NOMBRE    -   CALLE    -   ENERGIA
 -------------------------------
 Luis      -     8      -     11

Hay zombis en las calles:
5 8 10 14 14 19 20 21 22 32

  Luis, un zombi te ha visto. Has perdido

 Todos han sido comidos. No hay ganadores.
```

6.- Copiar objetos



No hemos creado objetos diferente sino que hemos creado alias diferentes a un mismo objeto.



Hemos eliminado una de las referencias pero no hemos eliminado el objeto 5.

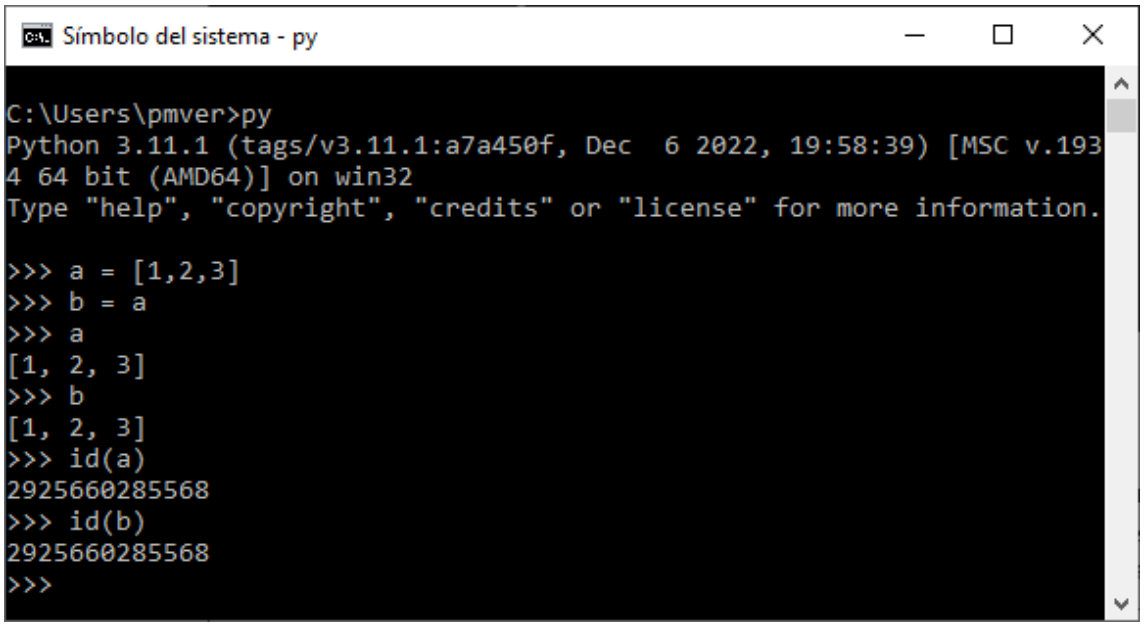
Tenemos que eliminar todas las referencias para que se elimine el objeto.

Python tiene una herramienta llamada el recolector de basura que tiene un contador de referencias, cuanto este contador llega a 0 el recolector de basura que se está ejecutando constantemente si encuentra un objeto que no tiene referencias lo elimina de la memoria.

El contador del 5 habrá llegado a 0 y el recolector de basura lo eliminará de la memoria.

La sentencia del no elimina objetos sino eliminar referencias, rompe el vínculo que hay entre una variable y el objeto al que está referido, de tal forma si no tiene más referencias este objeto queda libre de referencias y el recolector de basura lo elimina de la memoria.

Vamos a ver que ocurre con una lista.



Hasta aquí todo es igual.

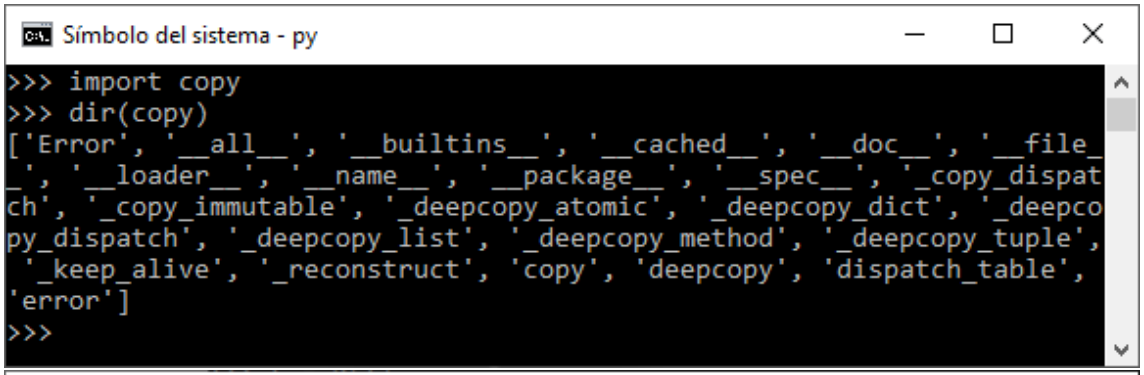
Si le cambiamos el índice 0 de a igual a 5, podemos observar que también ha cambiado en la lista b.

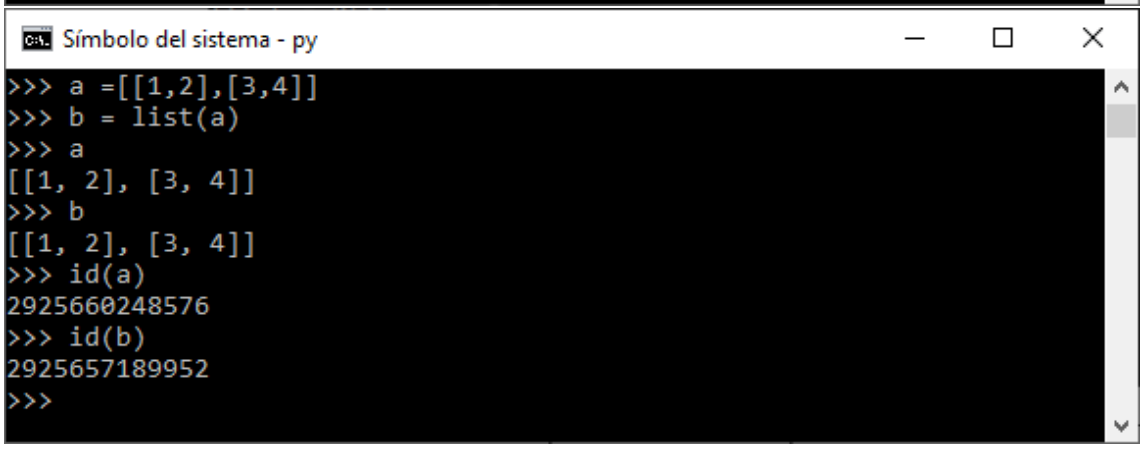
Las listas a y b hacen referencia a la misma lista.

En este caso hacemos una copia de a en la lista c, pero en distinta referencia, esto significa que si cambiamos el valor a[1] = 8, cuando mostremos la lista a el valor habrá cambiado pero si mostramos la lista c los valores siguen siendo los mismos. Son dos referencias distintas.

Otra forma de hacerlo es importando el módulo copy.

import copy





Creamos una lista que a su vez contiene dos listas y se la asignamos a la variable a.

Al hacer b = lista(a) estamos haciendo una copia de a en b en la misma referencia.

Modificando los datos de una variable afecta a la segunda variable.

Vamos a utilizar deepcopy() ya que al haber importado import copy tenemos acceso a ella:

```
>>> c[0][1] = 7
>>> a
[[5, 2], [3, 4]]
>>> b
[[5, 2], [3, 4]]
>>> c
[[5, 7], [3, 4]]
>>>
```

Si comparamos las listas a y b estas se mantienen iguales pero la lista c observamos un cambio.

Hemos realizado una copia profunda que son de dos objetos distintos.

Ahora vamos a ver que ocurren con los objetos de las clases que hemos creado nosotros mismos.

```
1   import random
2
3   class Zombi:
4       def __init__(self):
5           self.calle = random.randint(10,20)
6           self.direccion = random.choice(["izquierda", "derecha"])
7
8   z1 = Zombi()
9   z2 = z1
10
11  print(z1.calle)
12  print(z2.calle)
13
14  z1.calle = 100
15  print("-----------")
16  print(z1.calle)
17  print(z2.calle)
```

Vamos a ejecutar:

```
PS F:\Juego de zombis 2> & C:/Users/pmver/
ba.py"
19
19
-----------
100
100
```

Observamos que al cambiar uno también se cambia el segundo.

z1 y z2 son referencias a un mismo objeto.

```python
import random
import copy
class Zombi:
    def __init__(self):
        self.calle = random.randint(10,20)
        self.direccion = random.choice(["izquierda", "derecha"])

z1 = Zombi()
z2 = copy.copy(z1)

print(z1.calle)
print(z2.calle)

z1.calle = 100
print("-----------")
print(z1.calle)
print(z2.calle)
```

Vamos a ejecutar:

```
PS F:\Juego de zombis 2> & C:/Users/pmver/
ba.py"
13
13
-----------
100
13
```

Al principio hemos copiado un objeto con los mismos atributos, pero son objetos distintos.

```python
import copy
import random
class Zombi:
    def __init__(self):
        self.calle = random.randint(10,20)
        self.direccion = random.choice(["izquierda", "derecha"])

horda = []
for i in range(5):
    z = Zombi()
    horda.append(z)

copia = list(horda)

horda[3].calle = 200
horda[4].calle = 300
```

```
17
18  for z in horda:
19      print(z.calle)
20
21  print("-------------")
22
23  for z in copia:
24      print(z.calle)
```

Al haber realizado un copia de horda y a continuación cambiamos los valores en horda, cuando imprimimos de nuevo las dos listas tendrían que tener valores distintos:

Vamos a ejecutar:

```
PS F:\Juego de zombis 2> & C:/Users/pmver,
ba.py"
15
12
18
200
300
-------------
15
12
18
200
300
```

Podemos ver que han cambiado en los dos objetos.

Lo que pasa es que cada objeto apunta a la misma referencia. Hemos realizado una copia superficial.

```
13    copia = copy.deepcopy(horda)
```

Modificamos la línea 13 y ejecutamos.

```
PS F:\Juego de zombis 2> & C:/Users/pmver,
ba.py"
14
11
11
200
```

```
300
-------------
14
11
11
15
20
```

Ahora si son dos objetos distintos que empiezan con las propiedades con los mismos valores, pero luego serán independientes.

7.- Métodos especiales _ _init_ _ y _ _str_ _

```
1   class Persona:
2       def __int__(self, nombre, apellido, edad):
3           self.nombre = nombre
4           self.apellido = apellido
5           self.edad = edad
```

Ahora hemos visto un método especial que se caracteriza por tener dos guiones bajos delante de init y dos guiones bajos al final.

Estos métodos han sido implementados por los diseñadores del lenguaje Python, llevan a cabo tareas de una forma que permiten facilitar la utilización y el uso de los objetos en la codificación de los programas de Python.

Hay muchos métodos especiales por lo tanto cuando veas el nombre de un método con dos guiones bajos al inicio y al final, esto quiere decir que es un método diseñado para llevar a cabo una tarea especial.

El método __init__(self) nos permite inicializar los objetos de la clase Persona por ejemplo:

```
7   jorge = Persona("Jorge", "García", 26)
```

Ya tendríamos creado el objeto Persona llamado jorge, debido a que este método simplifica la tarea de crear un objeto al pasarle directamente los argumentos cuando llamamos a la clase Persona.

Sin este método se tendría que realizar de la siguiente forma:

```
1   class Persona:
2
3       def inicializar (self, nombre, apellido, edad):
4           self.nombre = nombre
5           self.apellido = apellido
6           self.edad = edad
7
8   jorge = Persona()
9   jorge.inicializar("Jorge", "García", 26)
```

Añadimos la siguiente línea:

```
11   print(jorge.nombre)
12   print(jorge.edad)
```

Este será el resultado cuando ejecutemos:

```
Jorge
26
```

Mediante el método __init__(def): sabemos que pasándole los argumentos a la clase Persona, nos crea el objeto y a la vez nos lo inicializa.

```
1   class Persona:
2
3       def __init__(self, nombre, apellido, edad):
4           self.nombre = nombre
5           self.apellido = apellido
6           self.edad = edad
7
8   jorge = Persona("Jorge", "García", 26)
9
10  print(jorge.nombre)
11  print(jorge.edad)
```

De este modo obtendremos el mismo resultado.

Hay otro método especial muy importante y utilizado que facilita mucho la codificación con objetos.

```
1   class Persona:
2
3       def __init__(self, nombre, apellido, edad):
4           self.nombre = nombre
5           self.apellido = apellido
6           self.edad = edad
7
8   jorge = Persona("Jorge", "García", 26)
9
10  print(jorge)
```

Si mostramos el resultado:

```
<__main__.Persona object at 0x000001CD035FB7D0>
```

Nos muestra que es un objeto, pero nos gustaría que al mostrar el objeto Jorge nos diese una información sobre el contenido de este objeto.

```python
class Persona:

    def __init__(self, nombre, apellido, edad):
        self.nombre = nombre
        self.apellido = apellido
        self.edad = edad

    def dime_datos(self):
        return self.nombre + " " + self.apellido

jorge = Persona("Jorge", "García", 26)

print(jorge.dime_datos())
```

Este será el resultado:

```
Jorge García
```

Existe un método especial que nos va a facilitar esta información.

```python
class Persona:

    def __init__(self, nombre, apellido, edad):
        self.nombre = nombre
        self.apellido = apellido
        self.edad = edad

    def __str__(self):
        return self.nombre + " " + self.apellido

jorge = Persona("Jorge", "García", 26)

print(jorge)
```

Este será el resultado:

```
Jorge García
```

Ahora ya no tenemos que llamar a ningún método.

Este método solo puede retornar caracteres, no números.

```
8       def __str__(self):
9           return self.nombre + " " + self.apellido + \
10              " de " +str(self.edad) + " años."
```

Hemos agregado más información al método, este será el resultado:

```
Jorge García de 26 años.
```

Existe muchos métodos especiales en Python, algunos muy útiles que iremos viendo en próximos capítulos.

8.- Objetos pasan mensajes

Una de las características esenciales, paradigma de la programación orientada a objetos van a ser que los objetos se pueden mandar mensajes de unos a otros, es decir los objetos mediante los métodos pueden pasar información o llevar a cabo acciones sobre otro objetos.

El paso de mensajes de un objeto a otro va a se un concepto esencial y una práctica clave en la programación orientada a objeto, esta práctica va a facilitar mucho la codificación de los programas y nos va a proporcionar una forma más directa y eficiente de programar.

Hasta ahora hemos hecho como esta:

```
1   class Persona:
2
3       def __init__(self, nombre, edad):
4           self.nombre = nombre
5           self.edad = edad
6
7       def __srt__(self):
8           return self.nombre
9
10      def presentarse(self):
11          print("Me llamdo {}. ¿Cómo te llamas?".format(self.nombre))
12
13  jorge = Persona("Jorge", 26)
14  maria = Persona("María", 24)
15
16  jorge.presentarse()
```

Vamos a ejecutar:

```
Me llamdo Jorge. ¿Cómo te llamas?
```

El objeto Jorge ha enviado una serie de caracteres, que no hace ni modifica nada más, pero mediante la programación orientada a objeto podemos mandar mensajes de un objeto a otro.

Por ejemplo:

```
1   class Persona:
2
3       def __init__(self, nombre, edad):
4           self.nombre = nombre
5           self.edad = edad
6
7       def __srt__(self):
8           return self.nombre
9
10      def presentarse(self):
11          print("Me llamdo {}. ¿Cómo te llamas?".format(self.nombre))
12
13      def responder(self, otro):
14          print("Hola {}, me llamo {}.".format(otro.nombre, self.nombre))
```

```
15
16
17    jorge = Persona("Jorge", 26)
18    maria = Persona("María", 24)
19
20    jorge.presentarse()
21    maria.responder(jorge)
```

Vamos a ejecutar:

```
Me llamdo Jorge. ¿Cómo te llamas?
Hola Jorge, me llamo María.
```

El objeto maria a pasado un mensaje al objeto Jorge.

Por lo tanto en los métodos no podemos pasar atributos como argumentos, sino que podemos pasar objetos completos.

Agregamos dos métodos más:

```
16        def preguntar_edad(self, otro):
17            print("Hola {}, ¿Cuantos años tienes?".format(otro.nombre))
18
19
20        def responder_edad(self, otro):
21            print("Hola {}, tengo {} años.".format(otro.nombre, self.edad))
```

Un método de preguntar_edad y otro método responder_edad

```
30    jorge.preguntar_edad(maria)
31    maria.responder_edad(jorge)
```

Jorge llama a un método y maria responde al otro método.

Vamos a ejecutar:

```
Hola María, ¿Cuantos años tienes?
Hola Jorge, tengo 24 años.
```

Ahora quien hace la pregunta es maría y quien responde es Jorge.

```
28    maria.preguntar_edad(jorge)
29    jorge.responder_edad(maria)
```

```
Hola Jorge, ¿Cuantos años tienes?
Hola María, tengo 26 años.
```

Propuesta practica que se comentará el en siguiente capítulo:

```
class Superheroe:
    def __init__(self,nombre, salud, ataque, escudo):
        self.nombre = nombre
        self.salud = salud
        self.ataque = ataque
        self.escudo = escudo

    def atacar():
        '''
        Si el enemigo no tiene puesto el escudo su salud diminuye
        la cantidad de ataque, si lo tienes puesto dla salud del que
        ataca disminuye la cantidad del ataque.
        Ponerk prints para mostar los efectos del ataque tanto en el
        superhéroe que ataca como en el otro.
        '''
        pass

thor =Superheroe("Thor", 20, 3, True)
hulk = Superheroe("Hulh", 20, 5, False)

thor.atacar(hulk)  # Salud de hulk: 20-3. Salud de thor: igual
hulk.atacar(thor)  # Salud de thor: Igual. Salud de hulk: 17-5
```

Este es mi resultado:

```
class Superheroe:
    def __init__(self,nombre, salud, ataque, escudo):
        self.nombre = nombre
        self.salud = salud
        self.ataque = ataque
        self.escudo = escudo

    def atacar(self, otro):
        if self.escudo:
            salud = otro.salud - self.ataque
            print("Salud de {}: {}. Salud de {}:igual ".format(otro.nombre, salud, self.nombre))

        else:
            salud = otro.salud - self.ataque - otro.ataque

            print("Salud de {}: Igual. Salud de {}: {}".format(otro.nombre, self.nombre, salud))


thor =Superheroe("Thor", 20, 3, True)
hulk = Superheroe("Hulk", 20, 5, False)

thor.atacar(hulk)  # Salud de hulk: 20-3. Salud de thor: igual
hulk.atacar(thor)  # Salud de thor: Igual. Salud de hulk: 17-5
```

Ejecutamos:

```
Salud de Hulk: 17. Salud de Thor:igual
Salud de Thor: Igual. Salud de Hulk: 12
```

9.- Métodos dentro de métodos

Propuesta del profesor:

```
1   class Superheroe:
2       def __init__(self,nombre, salud, ataque, escudo):
3           self.nombre = nombre
4           self.salud = salud
5           self.ataque = ataque
6           self.escudo = escudo
7
8       def __str__(self):
9           return self.nombre
10
11      def atacar(self, otro):
12          if otro.escudo:
13              self.salud -= self.ataque
14          else:
15              otro.salud -= self.ataque
16          print("{} ataca a {}".format(self, otro))
17          print("{} tiene de salud {}".format(self, self.salud))
18          print("{} tiene de salud {}".format(otro, otro.salud))
19
20
21  thor =Superheroe("Thor", 20, 3, True)
22  hulk = Superheroe("Hulk", 20, 5, False)
23
24  thor.atacar(hulk)  # Salud de hulk: 20-3. Salud de thor: igual
25  hulk.atacar(thor)  # Salud de thor: Igual. Salud de hulk: 17-5
```

Ejecutamos:

```
Thor ataca a Hulk
Thor tiene de salud 20
Hulk tiene de salud 17
Hulk ataca a Thor
Hulk tiene de salud 12
Thor tiene de salud 20
```

Vamos a seguir viendo las posibilidades que nos ofrece el pase de mensajes de un objeto a otro en la programación orientada a objetos, ya hemos visto como se puede modificar los atributos.

Vamos a ver ahora como un método puede llamar a otro método para realizar una acción complementaria, por ejemplo recuperamos la clase Persona del capítulo anterior.

```
1   class Persona:
2
3       def __init__(self,nombre, edad, cumple):
4           self.nombre = nombre
5           self.edad = edad
6           self.cumple = cumple
7
8       def __str__(self):
9           return self.nombre
10
11      def saludar(self, otro):
12          print("Hola {}. ¿Qué tal estás?".format(otro.nombre))
13
14      def responder(self, otro):
15          print("Hola {}. Muy bien".format(otro.nombre))
16
17  jorge = Persona("Jorge", 23, 2)
18  maria = Persona("María", 24, 3)
```

La hemos modificado ligeramente, el método saludar y el método responder.

Vamos a agregar el método felicitar.

```
17      def felicitar(self, otro):
18          print("{}, felicidades por su cumpleaños.".format(otro.nombre))
```

Como tenemos el métoto __str__() podemos modificar:

```
11      def saludar(self, otro):
12          print("Hola {}. ¿Qué tal estás?".format(otro))
13
14      def responder(self, otro):
15          print("Hola {}. Muy bien".format(otro))
16
17      def felicitar(self, otro):
18          print("{}, felicidades por su cumpleaños.".format(otro))
```

Modificamos el método saludar:

```
11      def saludar(self, otro, dia_actual):
12          print("Hola {}. ¿Qué tal estás?".format(otro))
13          if dia_actual == otro.cumple:
14              self.felicitar(otro)
```

```
26  jorge.saludar(maria, 2) # Pero no es su cumpleaños
```

Vamos a ejecutar:

```
Hola María. ¿Qué tal estás?
```

```
26    jorge.saludar(maria, 3) # Si es su cumpleaños
```

Vamos a ejecutar:

```
Hola María. ¿Qué tal estás?
María, felicidades por su cumpleaños.
```

```
11      def saludar(self, otro, dia_actual):
12          print("Hola {}. ¿Qué tal estás?".format(otro))
13          if dia_actual == otro.cumple:
14              self.felicitar(otro)
15          if dia_actual == self.cumple:
16              otro.felicitar(self)
```

Vamos a modificar el método saluda, en este caso queremos controlar que el que saluda es su cumpleaños.

```
28    jorge.saludar(maria, 2) # Cumpleaños de Jorge
```

Vamos a ejecutar:

```
Hola María. ¿Qué tal estás?
Jorge, felicidades por su cumpleaños.
```

Vamos a realizar unas modificaciones, te adjunto todo el código:

```
class Persona:

    def __init__(self,nombre, edad, cumple):
        self.nombre = nombre
        self.edad = edad
        self.cumple = cumple

    def __str__(self):
        return self.nombre

    def saludar(self, otro, dia_actual):
        print("Hola {}. ¿Qué tal estás?".format(otro))
        if dia_actual == otro.cumple:
            self.felicitar(otro)

    def responder(self, otro, dia_actual):
        print("Hola {}. Muy bien".format(otro))
        if dia_actual == otro.cumple:
            self.felicitar(self)
```

```
    def felicitar(self, otro):
        print("{}, felicidades de {} por tu cumpleaños.".format(otro,
self))


jorge = Persona("Jorge", 23, 2)
maria = Persona("María", 24, 3)

jorge.saludar(maria, 2)
maria.responder(jorge, 2)
```

Este será el resultado:

```
Hola María. ¿Qué tal estás?
Hola Jorge. Muy bien
María, felicidades de María por tu cumpleaños.
```

10.- Relaciones entre clases

Herencia	Es un	Animal → Perro
Composición	Tiene un	Casa → Habitación
Agregación	Usa un	Curso → Alumno

Los programas que se llevan a cabo para el paradigma de la programación orientada a objeto, lo más habitual es que hayan varias clases, diferentes clases con sus diferentes objetos que se relacionan entre sí. Esta relaciones entre las clases y sus objetos pueden ser de diferentes tipos, una de la más habituales van a ser la relación de Herencia, por ejemplo la clase Perro va a tener una relación con la clase Animal que va a ser de Herencia cuando la clase Perro hereda atributos, funcionalidad desde la clase Animal, esta relación es de tipo (Es un) porque una clase proviene de la otra, es una modificación de la otra.

Pero no solo hay este tipo de relación, hay otros dos tipos de relación muy importantes y muy habituales que nos conviene reconocer y para poder utilizarlos adecuadamente que van a ser la Composición y la Agregación. Estos dos tipos se parecen pero tienen una diferencia fundamental, por ejemplo si tenemos una clase Casa que nos permite crear objetos de la clase Casa y tenemos una clase Habitación que nos permite crear objetos de esta clase, los objetos de la clase Casa va a poder tener objetos de la clase Habitación dentro de su creación. Cuando creemos un objeto de la clase Casa vamos a necesitar objetos de la clase Habitación, para que formen parte de ese objeto de la clase Casa que estamos creando, por ello podemos decir que esta relación entre estas clases es de tipo (Tiene un).

La agregación en parecida a la composición pero se diferencia en algo fundamenta, por ejemplo la clase Curso contiene Alumno, usa objeto de la clase Alumno pero por ejemplo en el caso de la composición si un objeto de la clase Casa deja de existir los objetos de la clase Habitación no pueden existir independientemente fuera de la clase casa, porque si destruimos un objeto de la clase Casa destruimos también los objetos que teníamos de la clase Habitación, no pueden existir independientemente la clase casa, en cambio si tenemos un Curso que contiene varios objetos de la clase Alumno si este curso deja de existir los objeto de la clase Alumno no tienen que dejar de existir, pueden utilizarse para otro curso por ejemplo y pueden seguir existiendo independientemente del objeto de la clase Curso. La diferencia fundamental va a ser el ciclo de vida de los objetos, los objetos de la clase Alumnos van a existir independientemente de la clase Curso en contra que los objetos de la clase Habitación y el objeto de la clase Casa deja de existir ellos también van a dejar de existir.

Vamos a pasar a la agregación, después a la composición y por último a la herencia.

Para este capítulo hemos creado una nueva carpeta llamada superhéroes, y dentro de ella creamos tres archivos arma.py, principa.py y superhéroe.py.

Vamos a realizar la clase Superheroe en el archivo superhéroes.py.

```
1   class Superheroe:
2       def __init__(self, nombre, salud, ataque, escudo):
3           self.nombre = nombre
```

```
        self.salud = salud
        self.ataque = ataque
        self.escudo = escudo

    def __str__(self):
        return self.nombre
```

Vamos a crear una clase llamada Arma en el archivo arma.py

```
class Arma:
    def __init__(self, nombre, resistencia, destruccion, categoria=1):
        self.nombre = nombre
        self.resistencia = resistencia
        self.destruccion = destruccion
        self.categoria = categoria

    def __str__(self):
        return self.nombre
```

Ya hemos definido la clase Arma, ahora nos vamos al fichero principal.py.

```
from superheroe import Superheroe
from arma import Arma

martillo = Arma("Martillo",6, 4)
hacha = Arma("Hacha", 4, 5)

thor =Superheroe("Thor", 20, 3, True, martillo)
hulk = Superheroe("Hulk", 20, 5, False, hacha)
```

Estamos agregando un objeto en la creación de otro objeto.

Ahora vamos a definir un método atacar en la clase Superheroe.

```
class Superheroe:
    def __init__(self, nombre, salud, ataque, escudo, arma):
        self.nombre = nombre
        self.salud = salud
        self.ataque = ataque
        self.escudo = escudo
        self.arma = arma

    def __str__(self):
        return self.nombre


```

```
13      def atacar(self, otro):
14          otro.salud -= self.ataque + self.arma.destruccion
15          self.arma.resistencia -= 1
```

Además hemos realizando modificaciones

Ahora nos vamos a al archivo principal.

```
1   from superheroe import Superheroe
2   from arma import Arma
3
4   martillo = Arma("Martillo",6, 4)
5   hacha = Arma("Hacha", 4, 5)
6
7   thor =Superheroe("Thor", 20, 3, True, martillo)
8   hulk = Superheroe("Hulk", 20, 5, False, hacha)
9
10  print("Salud de Hulk:", hulk.salud)
11  thor.atacar(hulk)
12  print("Salud de Hulk:", hulk.salud)
13  print(martillo.resistencia)
```

Vamos a ejecutar:

```
Salud de Hulk: 20
Salud de Hulk: 13
5
```

Al principio la salud de Hulk es de 20, después del ataque que queda en 13 de salud y la resistencia del martillo que queda en 5.

Vamos a suponer ahora que Thor en lugar del martillo tiene el hacha y Hulk el martillo.

```
7   thor = Superheroe("Thor", 20, 3, True, hacha)
8   hulk = Superheroe("Hulk", 20, 5, False, martillo)
```

Modificamos parte del codigo archivo principal.py.

```
10  print("Salud de Hulk:", hulk.salud)
11  thor.atacar(hulk)
12  print("Salud de Hulk:", hulk.salud)
13  print("Resistencia del hacha: ", hacha.resistencia)
```

Vamos a ejecutar:

```
Salud de Hulk: 20
Salud de Hulk: 12
Resistencia del hacha:  3
```

En el próximo capítulo voy a proponerte el siguiente reto:

Thot.encontrar_mejora(“Hacha”) de esta forma el hacha que lleve Thor tendrá que aumentar su nivel de resistencia o su capacidad de destrucción o los dos.

Por ejemplo que aumente en una unidad la categoría de ese arma, por lo tanto habrá que definir un método encontrar_mejora(self) para la clase Thor que mejore el arma que lleve en ese momento el superhéroe si es que coincide con el tipo de arma la cual es la mejora.

11.- Relación de agregación entre clases

Vamos a resolver el reto que dejamos pendiente en el capítulo anterior en el que vimos el tipo de relaciones de clases y objetos Agregación.

Agregamos un objeto de la clase arma a los objetos de la clase superhéroe.

```
1  from superheroe import Superheroe
2  from arma import Arma
3
4  martillo = Arma("Martillo",6, 4)
5  hacha = Arma("Hacha", 4, 5)
6
7  thor = Superheroe("Thor", 20, 3, True, hacha)
8  hulk = Superheroe("Hulk", 20, 5, False, martillo)
9
10 print("Salud de Hulk:", hulk.salud)
11 thor.atacar(hulk)
12 print("Salud de Hulk:", hulk.salud)
13 print("Resistencia del hacha: ", hacha.resistencia)
```

Ahora teníamos que definir un método encontrar_mejora(“Hacha”)

Lo primero que vamos a hacer es ir a la clase Arma y crear el siguiente método.

```
11     def elevar_categoria(self):
12         self.categoria += 1
13         self.resistencia += 2
14         self.destruccion += 1
```

Ahora vamos a la clase Superheroe, para crear el siguiente método:

```
17     def encontrar_mejora(self, tipo_arma):
18         if self.arma.nombre == tipo_arma:
19             self.arma.elevar_categoria()
```

Ahora vamos al archivo principal.py.

```
1  from superheroe import Superheroe
2  from arma import Arma
3
4  martillo = Arma("Martillo",6, 4)
5  hacha = Arma("Hacha", 4, 5)
6
7  thor = Superheroe("Thor", 20, 3, True, hacha)
```

```
 8    hulk = Superheroe("Hulk", 20, 5, False, martillo)
 9
10    print("Salud de Hulk:", hulk.salud)  # 20
11    thor.atacar(hulk)
12    print("Salud de Hulk:", hulk.salud)  # 12
13    print("Resistencia del hacha: ", hacha.resistencia)  # 3
14
15    thor.encontrar_mejora("Hacha")
16    print("Resistencia del hacha: ", hacha.resistencia)  # 5
17    thor.atacar(hulk)
18    print("Salud de Hulk:", hulk.salud)  # 3
19    print("Resistencia del hacha: ", hacha.resistencia)  # 4
```

Vamos a ejecutar:

```
Salud de Hulk: 20
Salud de Hulk: 12
Resistencia del hacha:  3
Resistencia del hacha:  5
Salud de Hulk: 3
Resistencia del hacha:  4
```

Para el próximo capítulo os propongo es siguiente reto:

Tenemos definida una clase alumnos.

```
class Alumno:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

    def __str__(self):
        return self.nombre

class Curso:
    pass:

astronomia = Curso("Astronomia")

javier = Alumno("Javier", 30)
susana = Alumno("Susana", 35)
raquel = Alumno("Raquel", 40)

astronomia.matricular_alumno(javier)
astronomia.matricular_alumno(susana)
astronomia.matricular_alumno(raquel)
```

```
astronomia.anula_matricula(susana)

print("Aluimnos del curso {}: ".format(astronomia))
for alumno in astronomia.alumnos:
    print("-", alumno)

print("Edad media curso {} :".format(astronomia), end="" )
print(astronomia.edad_media_alumnos())
```

El resultado sería:

```
Alumnos del curso Astronomía:
- Javier
- Raquel
Edad media curso Astronomía: 35.0
```

El reto es definir la clase Curso que permita agregar objetos de la clase Alumno a los objetos de la clase Curso.

12.- Agregando objetos de al clase Alumnos a la clase Curso

Solución:

```
1   class Alumno:
2       def __init__(self, nombre, edad):
3           self.nombre = nombre
4           self.edad = edad
5
6       def __str__(self):
7           return self.nombre
8
9   class Curso:
10      def __init__(self, nombre):
11          self.nombre = nombre
12          self.alumnos = []
13
14      def __str__(self):
15          return self.nombre
16
17      def matricular_alumno(self, alumno):
18          self.alumnos.append(alumno)
19
20      def anula_matricula(self, alumno):
21          self.alumnos.remove(alumno)
22
23      def edad_media_alumno(self):
24          media = 0
25          for alumno in self.alumnos:
26              media += alumno.edad
27          media = media/len(self.alumnos)
28          return media
29
30      def mostrar_alumnos(self):
31          print("Alumnos del curso {}:".format(self))
32          for alumno in self.alumnos:
33              print("-", alumno)
```

```
34
35    astronomia = Curso("Astronomia")
36
37    javier = Alumno("Javier", 30)
38    susana = Alumno("Susana", 35)
39    raquel = Alumno("Raquel", 40)
40
41    astronomia.matricular_alumno(javier)
42    astronomia.matricular_alumno(susana)
43    astronomia.matricular_alumno(raquel)
44
45    astronomia.anula_matricula(susana)
46
47    astronomia.mostrar_alumnos()
48
49    print("Edad media curso {} :".format(astronomia), end="" )
50    print(astronomia.edad_media_alumno())
```

Este será el resultado:

```
Alumnos del curso Astronomia:
- Javier
- Raquel
Edad media curso Astronomia :35.0
```

Vamos a agregar otro alumno y lo matriculamos:

```
37    javier = Alumno("Javier", 30)
38    susana = Alumno("Susana", 35)
39    raquel = Alumno("Raquel", 40)
40    alvaro = Alumno("Alvaro", 45)
41
42    astronomia.matricular_alumno(javier)
43    astronomia.matricular_alumno(susana)
44    astronomia.matricular_alumno(raquel)
45    astronomia.matricular_alumno(alvaro)
```

Quitamos la anulación de matriculación de susana.

```
47    # astronomia.anula_matricula(susana)
```

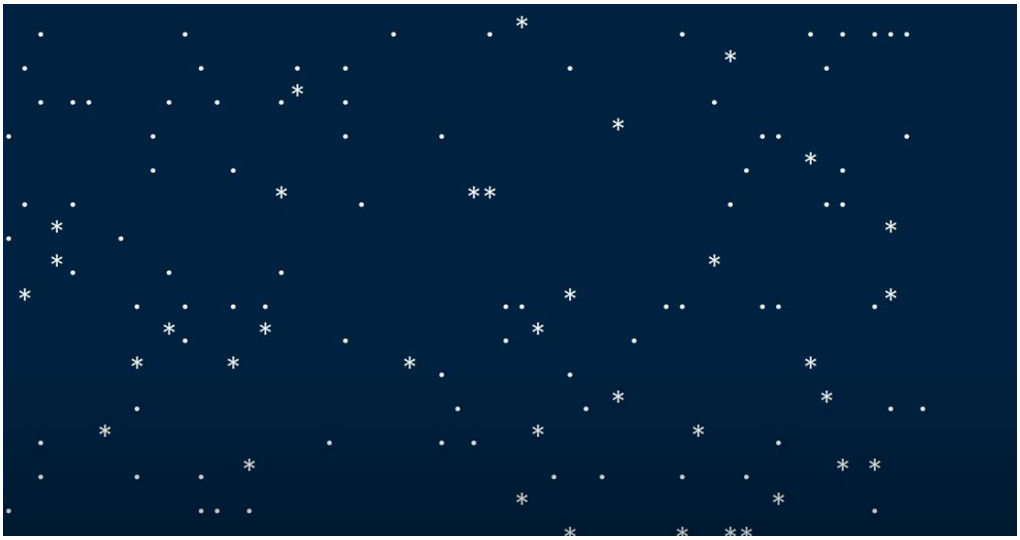
Vamos a ejecutar para ver si nos muestra los 4 alumnos y la media.

```
Alumnos del curso Astronomia:
- Javier
- Susana
- Raquel
- Alvaro
Edad media curso Astronomia :37.5
```

13.- Relación de composición entre clases.

En este capítulo vamos a ver otro tipo de relación entre clases ya hemos visto la agregación, vamos a ver la composición.

Aquí vemos un ejemplo de composición:



Un cielo compuesto de estrellas

Herencia	Es un	Animal → Perro
Composición	**Tiene un**	**Casa → Habitación**
Agregación	Usa un	Curso → Alumno

En este caso como solo vamos a realizar un dibujo, lo que vamos a hacer es una composición de cielo con estrellas, de tal forma que si por ejemplo le damos a ejecutar de nuevo, este cielo desaparece, se nos crea otro cielo distinto con otras estrellas distintas.

Si le volvemos a dar tenemos otro cielo con nuevas estrellas.

Vamos a realizar el código para realizar este programa.

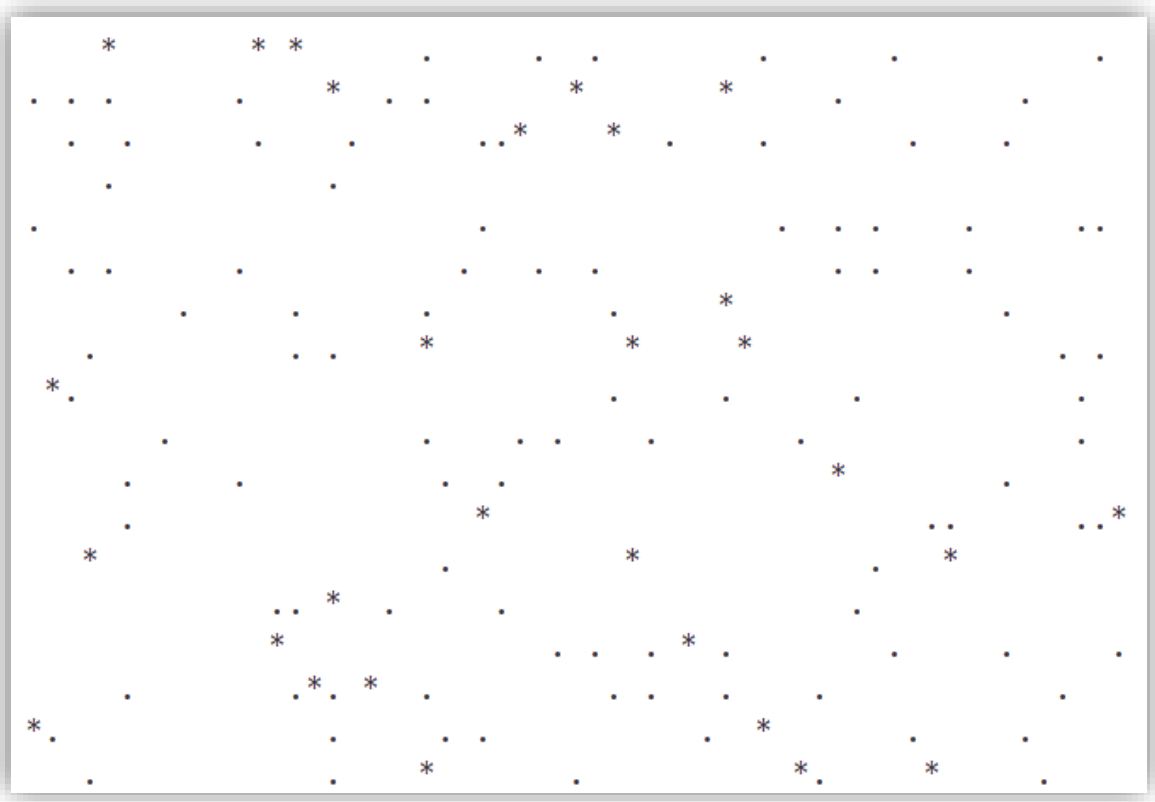
Para este proyecto vamos a crear una nueva carpeta llamada cielo y un archivo llamado cielo.py

```
1   import random
2
3   class Estrella:
4       def __init__(self, x, y):
5           self.x = x
6           self.y = y
7           # Una elección al azar entre asterisco, punto...
8           self.forma = random.choice(["*", ".", ".", "."])
9
10      def __str__(self):
11          return self.forma
12
13  class Cielo:
14      def __init__(self, filas, columnas):
15          self.filas = filas
16          self.columnas = columnas
17          self.cielo = []
18          for i in range(filas):
19              self.cielo.append([])
20              for j in range(columnas):
21                  self.cielo[i].append(" ")
```

```
21                  self.cielo[i].append(" ")
22
23      def poner_estrellas(self, numero_estrellas):
24          for i in range(numero_estrellas):
25              x = random.randint(0, self.columnas - 1)
26              y = random.randint(0, self.filas - 1)
27              estrella = Estrella(y, x)
28              self.cielo[y][x] = estrella
29
30      def mostrar(self):
31          for i in range(self.filas):
32              for j in range(self.columnas):
33                  print(self.cielo[i][j], end="")
34              print()
35
```

```
36   import os
37   os.system("cls")
38
39   cielo_1 = Cielo(18,60)
40   cielo_1.poner_estrellas(150)
41   cielo_1.mostrar()
42
```

Vamos a ejecutar:



Vamos a agregar:

```
43   cielo_2 = Cielo(18,80)
44   cielo_2.poner_estrellas(100)
45   cielo_2.mostrar()
```

Para el próximo capítulo te propongo el siguiente reto:

Tenemos una baraja con las picas, los corazones, los diamantes y los tréboles, de la A hasta el Rey. en el caso de una baraja de cartas que está compuesta de cartras.

Una carta por si sola no nos sirve para nada para un juego de cartas si no está dentro de una baraja, lo cual nos interesa es una clase baraja compuesta de objetos de tipo carta y podemos usar para ello una relación de composición entre la clase carta y la clase baraja.

```
1   pica ="\u2660"
2   corazon ="\u2665"
3   diamante ="\u2666"
4   trebol ="\u2663"
5
6   class Carta:
7       pass
8
9
```

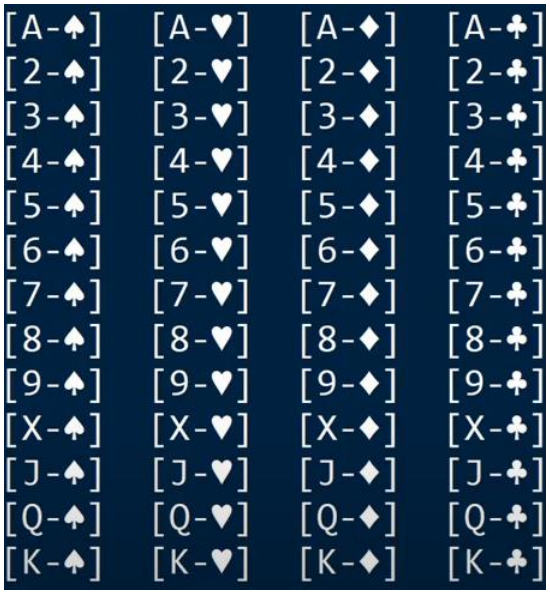
```
10  class Baraja:
11      pass
12
13
14  baraja = Baraja()
15  baraja.mostrar_baraja()
```

14.- Componiendo una baraja

Vamos a resolver el reto que dejamos pendiente en el capítulo anterior y así practicar con la relación de composición entre clases. Esta relación de tipo Tiene un

Herencia	Es un	Animal → Perro
Composición	**Tiene un**	**Casa → Habitación**
Agregación	Usa un	Curso → Alumno

Va a permitir crear fácilmente una baraja ya que contiene cartas, para un juego de cartas una carta individual no sirve para mucho en si misma, para un juego de cartas necesitamos una baraja, que las cartas existan dentro de la baraja, por lo tanto vamos a definir la clase Baraja que esté compuesta de cartas y lo vamos hacer para una baraja francesa.



Vamos ver como llevamos a cabo la definición de estas clases:

```
1   pica ="\u2660"
2   corazon ="\u2665"
3   diamante ="\u2666"
4   trebol ="\u2663"
5
6   class Carta:
7       def __init__(self, tanto, palo):
8           self.tanto = tanto
9           self.palo = palo
10
11      def __str__(self):
12          return "[{}-{}]".format(self.tanto, self.palo)
```

```
13
14
15  class Baraja:
16      def __init__(self):
17          palos = ["\u2660", "\u2665", "\u2666", "\u2663"]
18          tantos = ["A", "2", "3", "4", "5", "6", "7",
19                    "8", "9", "X", "J", "Q", "K"]
20          self.mazo = []
21          for t in tantos:
22              for p in palos:
23                  carta = Carta(t, p)
24                  self.mazo.append(carta)
25
26      def mostrar_baraja(self):
27          print()
28          for num, carta in enumerate(self.mazo):
29              if (num-3) % 4 != 0:
30                  print(carta, end="  ")
31              else:
32                  print(carta)
33
34
35
36  baraja = Baraja()
37  baraja.mostrar_baraja()
```

Este será el resultado:

```
[A-♠]  [A-♥]  [A-♦]  [A-♣]
[2-♠]  [2-♥]  [2-♦]  [2-♣]
[3-♠]  [3-♥]  [3-♦]  [3-♣]
[4-♠]  [4-♥]  [4-♦]  [4-♣]
[5-♠]  [5-♥]  [5-♦]  [5-♣]
[6-♠]  [6-♥]  [6-♦]  [6-♣]
[7-♠]  [7-♥]  [7-♦]  [7-♣]
[8-♠]  [8-♥]  [8-♦]  [8-♣]
[9-♠]  [9-♥]  [9-♦]  [9-♣]
[X-♠]  [X-♥]  [X-♦]  [X-♣]
[J-♠]  [J-♥]  [J-♦]  [J-♣]
[Q-♠]  [Q-♥]  [Q-♦]  [Q-♣]
[K-♠]  [K-♥] _[K-♦]  [K-♣]
```