

MÓDULO 1. PROGRAMACIÓN ORIENTADA A OBJETOS I

Loreto Pelegrín Castillo



MINISTERIO
DE EDUCACIÓN
Y FORMACIÓN PROFESIONAL



Índice:

1. Introducción
2. POO en Python
3. Algunos ejemplos

1

Introducción

1. Introducción

- La programación orientada a objetos o POO es un paradigma de programación en el que los elementos del mundo real se abstraen y se modelan en nuestros programas mediante lo que se conoce como clases y objetos.

2

P00 en Python

2. POO

- Una **clase** nos provee de un modo para empaquetar juntos datos y funcionalidades. Es una especie de molde o plano que nos va a definir las características que van a tener unas copias creadas a partir de ella, que es lo que se conoce como **objeto**.
- El mecanismo por el cual se crea un objeto a partir de una clase se conoce como crear una **instancia**.
- <https://docs.python.org/es/3/tutorial/classes.html>

2. POO

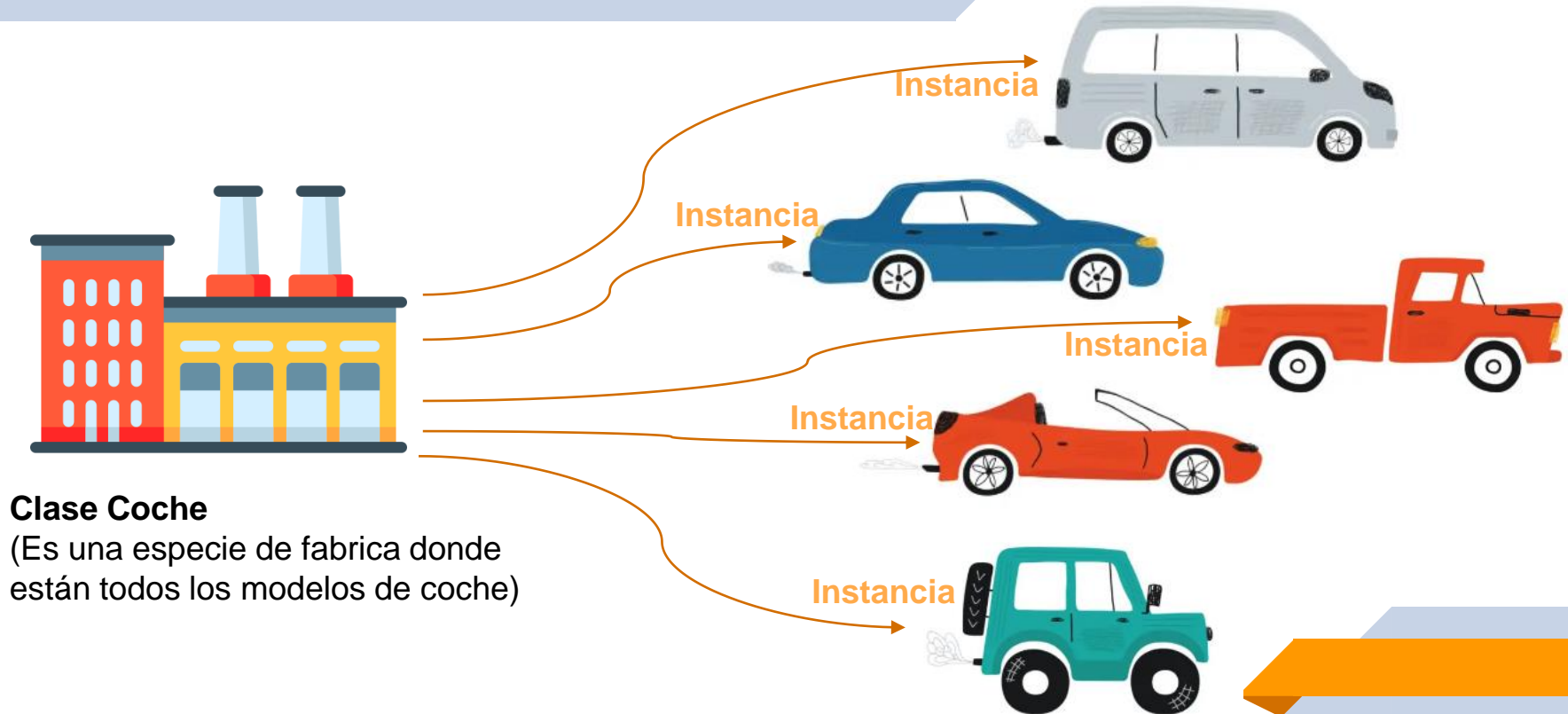
- Cada tipo de dato en Python (int, float, str, list, dict, etc.) es en realidad una clase. Cuando creamos una variable, en realidad estamos creando una instancia de esa clase.
- La función `type()` en Python te permite conocer el tipo de un objeto en tiempo de ejecución.

```
x = 10
print(type(x))  #<class 'int'>

y = "Hola, mundo!"
print(type(y))  #<class 'str'>

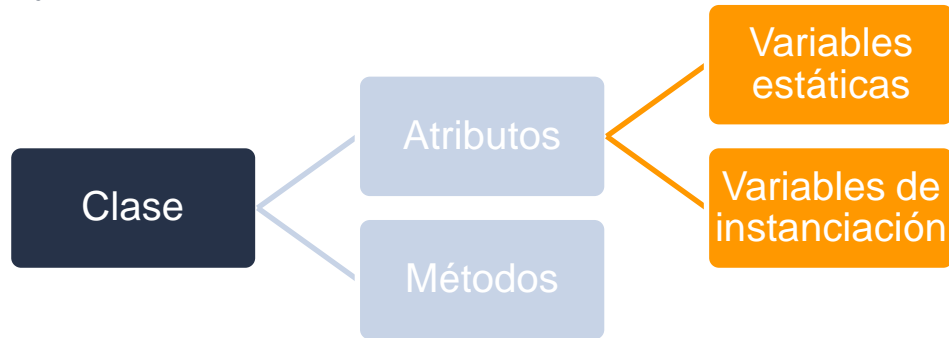
lista = [1, 2, 3]
print(type(lista))  #<class 'list'>
```

2. POO



2.1. Crear una clase

- Para crear una clase se utiliza la palabra reservada **class** seguida del nombre de la clase.
- En el siguiente esquema se muestra como están distribuidas las clases en Python.



2.1. Crear una clase

- Las características de las clases pueden dividirse en 2 grupos:
 - ▷ **Atributos:** Son las propiedades que pueden tener los objetos. Por ejemplo, en el caso de un coche, este tiene como atributos, el color, la marca, el modelo.
 - ▷ **Métodos:** Son las acciones que puede hacer el objeto. Un coche por ejemplo, puede acelerar, frenar,...
 - ▷ Hay que aclarar que estos métodos son funciones definidas dentro de la clase.

2.1. Crear una clase

- Los atributos o datos que empaquetamos dentro de las clases son variables. Y pueden ser de 2 tipos:
 - ▶ **Variables estáticas:** Están definidas dentro de la clase pero no dentro del método, se puede acceder directamente a ellas mediante el nombre de la clase.
 - ▶ **Variables de instancia:** Son variables dentro de la clase, pero solo se puede acceder a ellas una vez creado el objeto. (Instanciación del objeto)

2.1. Crear una clase

Clase

```
class Coche:
```

```
    #Atributos
```

```
    encendido = True #Variable estatica
```

```
    #Método __init__ (constructor de la clase)
```

```
    def __init__(self, marca, modelo, color):
```

```
        #Variables de instancia
```

```
        self.marca = marca
```

```
        self.modelo = modelo
```

```
        self.color = color
```

```
        self.velocidad = 0
```

Atributos

Métodos

```
    #Métodos
```

```
    def acelerar(self, cantidad):
```

```
        self.velocidad += cantidad
```

```
        print(f"El coche {self.marca} {self.modelo} está acelerando. Velocidad actual: {self.velocidad} km/h")
```

```
    def frenar(self, cantidad):
```

```
        self.velocidad -= cantidad
```

```
        print(f"El coche {self.marca} {self.modelo} está frenando. Velocidad actual: {self.velocidad} km/h")
```

2.1. Crear una clase

■ En la imagen anterior se mostraba una clase ya creada. Pero nosotros vamos a empezar por el principio:

1. Para crear la clase, se utiliza la palabra reservada **class** y el nombre de la clase que nosotros queramos.
2. Para el nombre de la clase se recomienda iniciar con una letra mayúscula y utilizar *CamelCase* (cada palabra que compone el nombre comienza con mayúscula, sin espacios). Por ejemplo: MiClase, Persona, CuentaBancaria o Coche.

2.1. Crear una clase

- En esta imagen se muestra la clase Coche.
- También podemos observar que tiene declarado un atributo (variable de la clase) y un método (acciones que puede realizar el objeto).

```
class Coche:  
  
    #Atributos  
    encendido = False  
  
    #Método  
    def arrancar():  
        print("El coche esta arrancando")
```

2.1. Crear una clase

- A continuación, vamos a añadirle más elementos a la clase.
- En la imagen, se observa la **variable estática**, fuera del **método `__init__`** o constructor de la clase, y las **variables de instancia**.

```
class Coche:

    #Atributos
    encendido = False #Variable estatica

    #Método __init__ (constructor de la clase)
    def __init__(self, marca, modelo, color):
        #Variables de instancia
        self.marca = marca
        self.modelo = modelo
        self.color = color
        self.velocidad = 0
```

2.1.1. El método constructor

- En el ejemplo anterior hemos visto el método `__init__`, este es un método especial de las clases llamado **método constructor**.
- Este método es llamado de manera automática en primer lugar cuando creamos un objeto a partir de una clase. Nos sirve para poder inicializar el objeto con unos atributos ya determinados.
- **Parámetro self**: Nos sirve para pasar el propio objeto como argumento del método. Este parámetro nos sirve para a Python que nos estamos refiriendo al propio objeto en si mismo. **En el método constructor SIEMPRE se colocará como primer parámetro.**

2.1.1. El método constructor


```
class Pais:

    def __init__(self, nombre, poblacion, continente="Europa"):

        """Crearemos el objeto con los argumentos que pasamos
        a la hora de instanciar el objeto"""

        self.nombre = nombre
        self.poblacion = poblacion
        self.continente = continente

    """Creamos instancias de la clase
    Nos dará error porque espera al menos 2 argumentos más"""
    Pais1 = Pais("Alemania")
```



```
class Pais(
    nombre: Any,
    poblacion: Any,
    continente: str = "Europa"
)
```

2.1.1. El método constructor

```
class Pais:

    def __init__(self, nombre, poblacion, continente="Europa"):

        """Crearemos el objeto con los argumentos que pasamos
        a la hora de instanciar el objeto"""

        self.nombre = nombre
        self.poblacion = poblacion
        self.continente = continente
```



```
#Se usará el argumento Europa por defecto
Pais1 = Pais("Alemania",40)
print(Pais1.continente) #Europa

Pais2 = Pais("Ecuador",60,"America")
print(Pais2.continente) #America
```

2.1.1. El método constructor

- No es estrictamente necesario utilizar un constructor en Python. Puedes crear clases y objetos sin definir un método `__init__` (el constructor). Sin embargo, es altamente recomendable utilizarlo en la mayoría de los casos.

```
class Perro:  
    pass  
  
mi_perro = Perro()
```

- En este ejemplo, se crea una clase `Perro` sin un constructor. Al crear un objeto `mi_perro`, se obtiene un objeto vacío sin ningún atributo definido.
- La instrucción **`pass`** simplemente se omite durante la ejecución del programa.

2.2. Instanciar la clase


- Para instanciar la clase, por ejemplo, Coche, hay que crear una variable, que en realidad es un objeto, en este caso, llamado Mercedes.
- Se llama a la clase: Coche y se le pasa al constructor tantos argumentos como tenga declarado este en su definición.

```
# Instancia del objeto  
Mercedes = Coche("Mercedes", "GLA 220", "Azul")
```


2.3. Acceso a las variables

```
class Coche:
    #Atributos
    encendido = True #Variable estatica


    #Método __init__ (constructor de la clase)
    def __init__(self, marca, modelo, color):
        #Variables de instancia
        self.marca = marca
        self.modelo = modelo
        self.color = color
        self.velocidad = 0
```




```
print(Coche.encendido) #True
```





```
# print(Coche.color)
"""Traceback (most recent call last):
  File "c:\Unidad_5\Ejemplos\coche.py", line 16, in <module>
    print(Coche.color)
    |      ^^^^^^^^^^^
AttributeError: type object 'Coche' has no attribute 'color'"""
```



```
# Instancia del objeto
Mercedes = Coche("Mercedes", "GLA 220", "Azul")
```



```
"""Una vez creado el objeto, podemos acceder a las variables de instancia"""
print(Mercedes.color) #Azul
print(Mercedes.modelo) #GLA 220
print(Mercedes.encendido) #True
```



2.3. Acceso a las variables

- Para acceder a las variables, hay que tener en cuenta que tipo de variable es.
- En el ejemplo anterior, tenemos la **variable estática** encendido. (1)
 - Para acceder a esa variable no hace falta instanciar la clase, se puede acceder al valor llamándola directamente con el nombre de la clase. Por ejemplo: Coche.encendido (2)
 - Pero también se puede acceder una vez instanciada la clase, con el nombre del objeto creado. Por ejemplo, Mercedes.encendido. (3)

2.3. Acceso a las variables

- Por el contrario, para acceder al valor de las variables de instancia, modelo, color o marca. Es necesario instanciar la clase. (4)
 - No podemos acceder al valor de la variable antes de instanciar el objeto. Dará error de que no encuentra el atributo. (5)
 - Una vez instanciada la clase, con el nombre del objeto creado podemos acceder a esa variable. Por ejemplo, Mercedes.color.(6)

2.4. Método `__str__`

- En Python, el método `__str__` es un método especial que se utiliza para definir cómo un objeto de una clase debe ser representado como una cadena de caracteres.
- En otras palabras, cuando llamas a `print()` sobre un objeto, Python internamente invoca este método para obtener la representación en cadena de ese objeto y luego imprime esa cadena en la consola.

2.4. Método `__str__`

- En nuestro ejemplo de Coche, si intentamos hacer un print, de un objeto instanciado sin haber creado el método `__str__`. Por ejemplo, Mercedes, nos da la información de que es un objeto Coche y esta en una dirección de memoria.

```
# Instancia del objeto
Mercedes = Coche("Mercedes", "GLA 220", "Azul")
print(Mercedes) #<__main__.Coche object at 0x000001F6970D3C20>
```

2.4. Método `__str__`

- Para este caso, vamos a crear el método `__str__`.

```
def __str__(self):  
    return (f"Coche: {self.marca}, modelo: {self.modelo}")
```

- Y al realizar la llamada al objeto instanciado, nos devuelve lo que se ha escrito en el método `__str__`.

```
# Instancia del objeto  
Mercedes = Coche("Mercedes", "GLA 220", "Azul")  
print(Mercedes) #Coche: Mercedes, modelo: GLA 220
```

Ejercicio de clase

Ejercicio 1:

- Crear una clase Perro.
- Declarar una variable estática llamada contador_perros y asignarle el valor 0.
- En el constructor, le pasamos como argumentos: nombre, raza y edad.
 - Dentro del constructor llamamos a la variable estática contador_perros y le decimos que cuando creamos un nuevo objeto, se le sume 1 a ese contador.
- Crear el método __str__, y que muestre nombre y raza.
- Instanciamos 3 objetos Perro diferentes.
- Imprimimos la llamada a contador_perros. El resultado debe de ser 3.
- Imprimir uno de los objetos Perro.

Ejercicio de clase

Ejercicio 2:

- Crear una clase Ordenador.
- Declarar una variable estática llamada encendido y asignarle el valor False.
- En el constructor, le pasamos como argumentos: procesador, memoria, memoriaRam y sistemaOperativo (valor booleano).
- Crear el método __str__ y que muestre todos los datos.
 - Tener en cuenta que sistemaOperativo es booleano, realizar un control mediante un if dentro del método. Si tiene sistema operativo que muestre un mensaje como que lo tiene además del resto de datos, y si por el contrario no lo tiene que diga que no tiene sistema operativo y el resto de los datos.

Ejercicio de clase

- Instanciamos 2 objetos Ordenador diferentes, uno con sistemaOperativo = True y el otro con sistemaOperativo = False.
- Imprimir los 2 objetos instanciados y ver los diferentes mensajes.

2.5. Diferentes tipos de métodos

- En Python podemos clasificar los métodos en tres tipos diferentes:
 - ▷ Métodos de instancia.
 - ▷ Métodos de clase.
 - ▷ Métodos estáticos.

2.5.1. Métodos de instancia

- Estos métodos siempre incorporan el parámetro **self** en primer lugar. Se llaman de este modo porque no podemos usarlos hasta que instanciamos un objeto de la clase, de ahí que el primer parámetro sea self.
- Estos métodos pueden obtener y cambiar los atributos de instancia, además de llamar a otros métodos de instancia y de clase.

2.5.1. Métodos de instancia

■ Para este ejemplo vamos a seguir utilizando la clase Coche.

```
#Métodos
def acelerar(self, cantidad):
    self.velocidad += cantidad
    print(f"El coche {self.marca} {self.modelo} está acelerando. Velocidad actual: {self.velocidad} km/h")

def frenar(self, cantidad):
    self.velocidad -= cantidad
    print(f"El coche {self.marca} {self.modelo} está frenando. Velocidad actual: {self.velocidad} km/h")
```

■ Declaramos 2 métodos, acelerar y frenar. En estos métodos modificamos el valor de velocidad, que la declaramos en el constructor con valor 0.

2.5.2. Métodos de clase

- Al igual que las variables estáticas, a este tipo de métodos se puede acceder directamente usando la clase sin necesidad de crear una instancia de esta.
- Para indicar que un método es de clase debemos usar el **decorador @classmethod** y pasar como parámetro **cls**, que es el que se suele usar por convenio. Estos métodos pueden acceder a otros métodos y atributos de la clase.

2.5.2. Métodos de clase

■ ¿Cuándo usar métodos de clase?

- ▶ Para crear objetos de una clase de diferentes maneras.
- ▶ Por ejemplo, para obtener información sobre la clase o modificar su comportamiento de forma global.
- ▶ Métodos que no necesitan acceder a los datos específicos de una instancia: Cuando una función no requiere conocer el estado de un objeto particular.

2.5.2. Métodos de clase

- ¿Qué es un decorador?
- Un decorador en Python es una función que modifica el comportamiento de otra función. Es como una capa adicional que se aplica a una función existente, agregándole funcionalidades sin alterar directamente su código original.
- Se coloca el decorador antes de la función a decorar, utilizando el símbolo @.

2.5.2. Métodos de clase

- Para este ejemplo vamos a utilizar el ejercicio de clase Perro.
- Creamos el método de clase:
 - Cantidad_perros(cls)

```
class Perro:
    contador_perros = 0 # Variable estática para contar perros

    def __init__(self, nombre, raza):
        self.nombre = nombre
        self.raza = raza
        Perro.contador_perros += 1

    @classmethod
    def cantidad_perros(cls):
        return cls.contador_perros

print(f"Hay {Perro.cantidad_perros()} perros") #Hay 0 perros

# Creando algunos perros
perro1 = Perro("Alma", "Golden Retriever")
perro2 = Perro("Lola", "Chihuahua")
perro3 = Perro("Ellie", "Border Collie")

print(f"Hay {Perro.cantidad_perros()} perros") #Hay 3 perros
```

2.5.5. Métodos estáticos

- Este tipo de métodos no necesitan el parámetro `self` y tampoco necesita crear un instancia de la clase para usarlos.
- Se utilizan para agrupar funciones relacionadas con una clase, pero que no requieren acceso a los atributos o métodos de instancia o de clase.
- Se utiliza el decorador `@staticmethod`, y no necesitan obligatoriamente que se les pase un parámetro.

2.5.5. Métodos estáticos

- En la clase Calculadora hemos creado 3 métodos.
- `saludar()` sin parámetros de entrada, y los otros 2 si reciben parámetros.

```
class Calculadora:
    @staticmethod
    def saludar():
        print("Bienvenidos a la calculadora")

    @staticmethod
    def sumar(a, b):
        return a + b

    @staticmethod
    def restar(a, b):
        return a - b

# Uso de los métodos estáticos
Calculadora.saludar() #Bienvenidos a la calculadora
resultado = Calculadora.sumar(3, 5)
print(resultado) #8
```

2.6. Documentación

```
class Perro:
    """Clase Perro, para crear perros y sus cosas de perros"""
    contador_perros = 0 # Variable estática para contar perros

    def __init__(self, nombre, raza):
        self.nombre = nombre
        self.raza = raza
        Perro.contador_perros += 1

    @classmethod
    def cantidad_perros(cls):
        return cls.contador_perros

    def ladrar(self):
        """Función para imprimir ¡Guau!"""
        print("¡Guau!")

print(f"Hay {Perro.cantidad_perros()} perros") #Hay 0 perros

# Creando algunos perros
perro1 = Perro("Alma", "Golden Retriever")
perro2 = Perro("Lola", "Chihuahua")
perro3 = Perro("Ellie", "Border Collie")

print(f"Hay {Perro.cantidad_perros()} perros") #Hay 3 perros

print(Perro.__doc__) #Clase Perro, para crear perros y sus cosas de perros
print(perro1.ladrar.__doc__) #Función para imprimir ¡Guau!
```

- Es conveniente documentar nuestras clases igual que hacemos con las funciones mediante el uso de docstrings.
- Posteriormente, para acceder a la documentación de las clases podemos usar el método `__doc__`

Ejercicio de clase

Ejercicio 1:

- En la clase Perro que creamos anteriormente, vamos a añadir 2 métodos:
 - Método ladrar: Que muestre el nombre del perro y un mensaje que ponga esta ladrando.
 - Método jugar: se le pasa por parámetro pelota que es de tipo booleano.
 - Si pelota == True. Nombre del perro esta jugando con su pelota
 - Si pelota == False. Nombre del perro esta jugando.

Ejercicio de clase

Ejercicio 2:

- En la clase Ordenador que creamos anteriormente, vamos a añadir 1 método:
 - Método instalarSO: Este método comprueba si el ordenador tiene SO. Si no lo tiene, le pregunta al usuario que SO quiere instalar. Y luego muestra el sistema operativo instalado. Si tiene SO, le dice al usuario que ya tiene SO.
 - Instanciar un objeto con SO = False. Imprimir el objeto creado.
 - Llamar al método instalarSO e imprimir de nuevo el objeto.

3

Algunos ejemplos

3.1. Ejemplo 1: Clases con parámetro lista

```
import statistics

class Estudiante:
    def __init__(self, nombre, calificaciones):
        self.nombre = nombre
        self.calificaciones = calificaciones

    def calcularMedia(self):
        return statistics.mean(self.calificaciones)
# Lista de calificaciones

# Creando un objeto Estudiante
estudiante1 = Estudiante("Carlos Pérez", [9, 8, 7, 10])
print(f"La media de {estudiante1.nombre} es {estudiante1.calcularMedia()}")
#La media de Carlos Pérez es 8.5
```

3.2. Ejemplo 2: Clases con parámetro diccionario

```
class Persona:
    def __init__(self, datos):
        self.nombre = datos['nombre']
        self.edad = datos['edad']
        self.ciudad = datos['ciudad']

# Creando un objeto Persona
persona1 = Persona({'nombre': 'Loreto', 'edad': 30, 'ciudad': 'Murcia'})
print(persona1.nombre) # Imprime: Loreto
```

3.3. Ejemplo 3: Listas de objetos

```
class Persona:
    def __init__(self, datos):
        self.nombre = datos['nombre']
        self.edad = datos['edad']
        self.ciudad = datos['ciudad']

    def __str__(self):
        return f"Nombre: {self.nombre}, Edad: {self.edad}, Ciudad: {self.ciudad}"

# Creando un objeto Persona
persona1 = Persona({'nombre': 'Loreto', 'edad': 30, 'ciudad': 'Murcia'})
persona2 = Persona({'nombre': 'Carlos', 'edad': 25, 'ciudad': 'Murcia'})
persona3 = Persona({'nombre': 'Jose', 'edad': 22, 'ciudad': 'Aguilas'})
persona4 = Persona({'nombre': 'Clara', 'edad': 18, 'ciudad': 'Murcia'})
```

3.3. Ejemplo 3: Listas de objetos

```
#VERSION 1
#Creamos una lista vacia
personas = []
#Añadimos los elementos a la lista
personas.append(persona1)
personas.append(persona2)
personas.append(persona3)
personas.append(persona4)

#VERSION 2
#Creamos una lista y le pasamos los objetos como elementos de la lista
personas1 = [persona1,persona2,persona3,persona4]
# Imprimir la información de cada persona en la lista
for persona in personas:
    print(persona)

# Nombre: Loreto, Edad: 30, Ciudad: Murcia
# Nombre: Carlos, Edad: 25, Ciudad: Murcia
# Nombre: Jose, Edad: 22, Ciudad: Aguilas
# Nombre: Clara, Edad: 18, Ciudad: Murcia
```



¡Gracias!