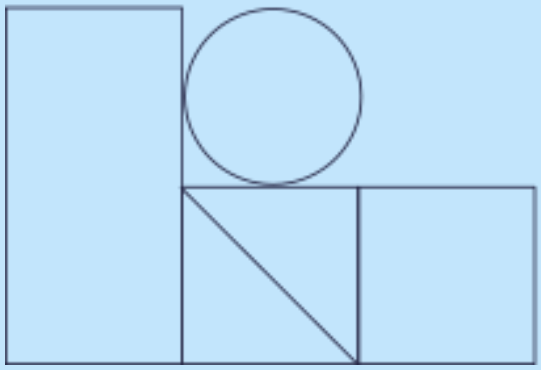
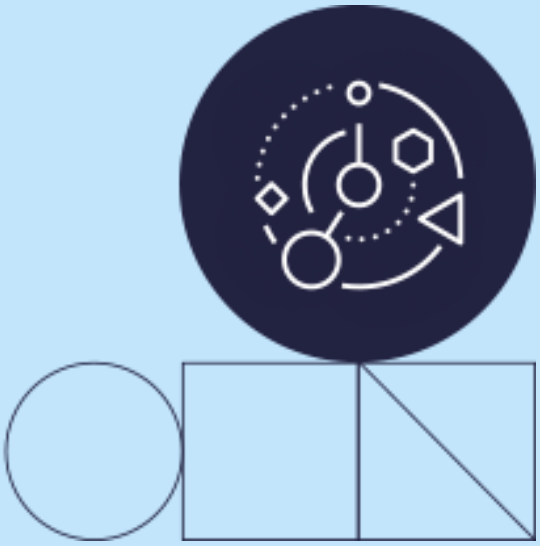
IBM SkillsBuild

IBM

POO con Python

Constructores y destructores





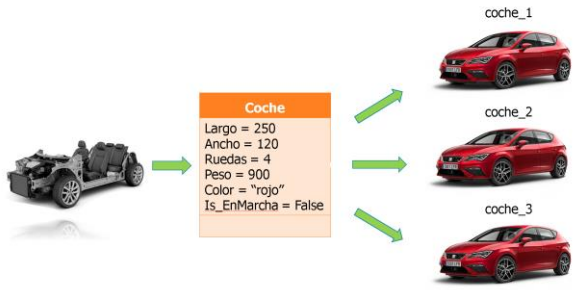
Índice

Constructores

Un constructor se define como un método especial con el que damos un estado inicial a nuestros objetos.

Siguiendo con el ejemplo de nuestra clase Coche, podemos crear un constructor con el que demos un estado inicial a los objetos de clase Coche que creemos, de tal forma que cada coche creado ya disponga de una serie de atributos y métodos “por defecto”. Luego podremos cambiarlos si queremos, pero “de fábrica” todos los coches saldrán con los atributos y métodos que pongamos en el constructor.

Por ejemplo:



La sintaxis para crear un constructor es:

```
def __init__(self):
```

Recordemos que hemos mencionado que un constructor es un método, por lo tanto, una función. Dentro de dicha función deberemos poner los atributos y métodos iniciales que queremos que nuestras instancias de clase tengan.

Pasemos el ejemplo gráfico del coche a código:

```
# Creación de la clase Coche
class Coche():
```

```
# Declaración del constructor de la clase Coche
    def __init__(self):
        self.largo = 250
        self.ancho = 120
        self.ruedas = 4
        self.peso = 900
        self.color = "rojo"
        self.is_enMarcha = False

# Declaración de métodos
    def arrancar(self):     #self hace referencia a la instancia de clase.
        self.is_enMarcha = True    #Es como si pusiésemos miCoche.is_enMarcha = True

    def estado(self):
        if (self.is_enMarcha == True):
            return "El coche está arrancado"
        else:
            return "El coche está parado"

# Declaración de una instancia de clase, objeto de clase o ejemplar de clase.
coche_1 = Coche()

# Acceso a un atributo de la clase Coche. Nomenclatura del punto.
coche_1.ruedas = 7
print("El largo del coche es de" , coche_1.largo, "cm.")
coche_1.arrancar()
print(coche_1.estado())

# Acceso a un método de la clase Coche. Nomenclatura del punto.
print("El coche está arrancado:" , coche_1.arrancar())
```

Hemos creado la clase **Coche** y un constructor con el que le damos un estado inicial a los objetos que creemos de dicha clase. En nuestro caso todos los coches que creemos van a tener:

- largo = 250
- ancho = 120
- ruedas = 4
- peso = 900
- color = "rojo"
- is_enMarcha = False

Como mencionamos anteriormente, este será el estado inicial. Podemos cambiarlo posteriormente y decir, por ejemplo que **coche_1.color = "verde"** y desde ese momento el color de **coche_1** pasará a valer **verde**, pero por defecto era **rojo** como el de todos los coches que creemos con nuestra clase **Coche**.

Toda clase de Python debe tener un constructor, es obligatorio. Pero, entonces, ¿cómo hemos creado nosotros anteriormente los ejemplos de las clases Coche de capítulos anteriores? Si no usamos constructores, ¿estaban mal programadas? La respuesta es no. En los ejemplos anteriores no usamos constructores, pero ello no produjo ningún fallo debido a que, si nosotros no creamos un constructor, Python nos crea uno por defecto. Lo crea vacío, pero lo crea.

De modo que, como norma general, siempre crearemos nosotros el constructor de nuestras clases, pero si no lo hacemos, Python lo creará por nosotros. Eso sí, lo crea vacío, sin atributos ni métodos por defecto.

En nuestro ejemplo de la clase **Coche**, hemos dado una serie de atributos fijos a todos los ejemplares de **coche** que creemos. Pero podemos crear nuestro constructor de forma que sea más flexible y acepte que decidamos lo que va a valer cada uno de esos atributos cuando instanciemos un objeto **coche**.

Es decir, al constructor le diremos los atributos que tiene que recibir un ejemplar de coche, sin darle valor de momento a dichos atributos y será cuando instanciemos dicho ejemplar de coche cuando le diremos lo que vale cada atributo. Es lo que se denomina un constructor con parámetros.

Veámoslo con un ejemplo:

```
class Coche:

# Declaración del constructor con
parámetros
    def __init__(self, largo, ancho,
ruedas, peso, color, is_enMarcha):
        self.ancho = ancho
        self.ruedas = ruedas
        self.peso = peso
        self.color = color
        self.is_enMarcha = is_enMarcha


# Declaración de dos instancias de clase pasándoles
los parámetros requeridos en el constructor.

coche_1 = Coche(400, 160, 4, 1200,
"amarillo", True)
coche_2 = Coche(420, 150, 4, 1500,
"verde", False)
```

Como vemos en el ejemplo, hemos creado una clase Coche y en su constructor hemos dicho que, cuando se cree una instancia de Coche se le deben pasar obligatoriamente el valor de los parámetros; **largo**, **ancho**, **ruedas**, **peso**, **color**, **is_enMarcha**.

Estamos obligados a pasar dichos parámetros (y en el mismo orden exacto) al instanciar un ejemplar de coche, de lo contrario, nos dará error.

Si cambiásemos la declaración de **coche_2** y no le pasásemos un parámetro obligatorio, por ejemplo **is_enMarcha** nos saldría un mensaje como el siguiente:

```
coche_2 = Coche(420, 150, 4, 1500,
"verde")
```

```
---> 21 coche_2 = Coche(420, 150, 4, 1500, "verde")

TypeError: Coche.__init__() missing 1 required posi
```

Como vemos, el programa ha fallado y se nos muestra explícitamente el motivo del fallo. En nuestro caso nos dice que nos falta un argumento (**is_enMarcha**) que está en el constructor, pero nosotros no lo hemos puesto en el ejemplar de **coche**.

Así pues, el método constructor nos permitirá asignar atributos cada vez que creemos un objeto a partir de esa clase haciéndolo obligatorio. Y además nos permitirá llamar métodos de la clase sin instanciar, entre otras cosas.

Destructores

Así como existe un método constructor, existe un método destructor cuyo cometido consiste en eliminar instancias de una clase. Es decir, elimina un objeto.

El método destructor es el método **__del__()**

```
class Book():
  """
  Clase para trabajar con libros
  """

  def __init__(self, title, author = "",
electronic = False):
    self.title = title
    self.author = author
    self.is_electronic = electronic

  def __del__(self):
    print("Acabas de llamar al método
destructor. El objeto acaba de ser
eliminado")
```

Para eliminar un objeto, utilizaos la palabra reservada **del**

```
book = Book("Lazarillo de Tormes")
book.title

del book
```

Si intentásemos acceder al objeto **book**, obtendríamos error pues ha dejado de ser una instancia de la clase **Book** porque lo hemos eliminado.