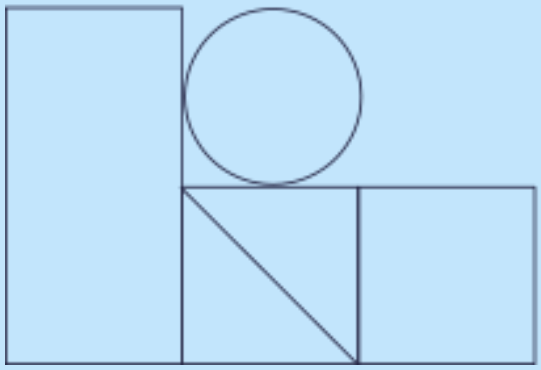
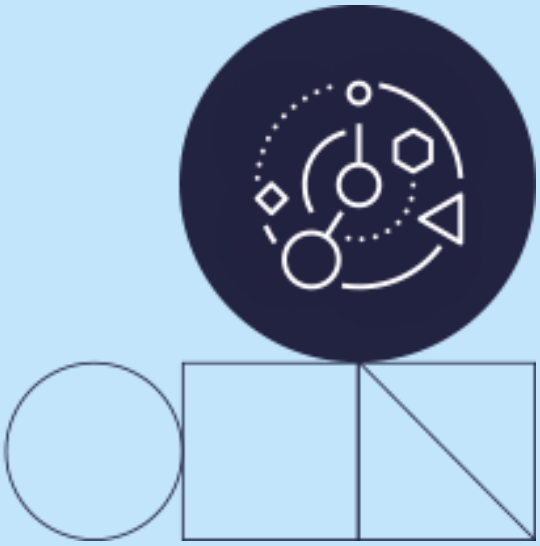
IBM SkillsBuild

IBM

POO con Python

Objetos y clases





Índice

Introducción

Como vimos en el tema anterior, en POO trabajamos siempre con objetos y todos nuestros objetos tendrán atributos y métodos.

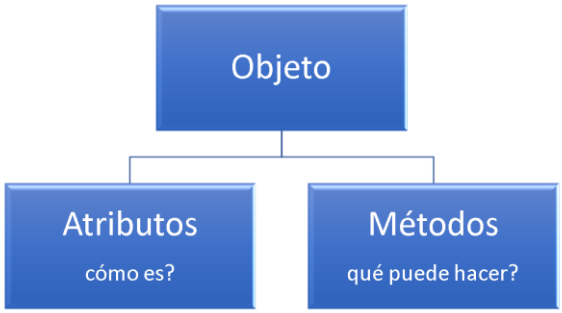
El primer y más importante concepto de la POO es la distinción entre clase y objeto.

Una **clase** es una plantilla. Define de manera genérica cómo van a ser los objetos de un determinado tipo. Por ejemplo, una clase para representar a animales puede llamarse ‘animal’ y tener una serie de atributos, como ‘nombre’ o ‘edad’ (que normalmente son propiedades), y una serie con los comportamientos que estos pueden tener, como caminar o comer, y que a su vez se implementan como métodos de la clase (funciones).

Objetos

Qué es un objeto

Un ejemplo sencillo de un objeto, como decíamos antes, podría ser un animal en concreto, por ejemplo un perro. Un perro tiene una edad, por lo que creamos un nuevo atributo de 'edad' y, además, puede envejecer, por lo que definimos un nuevo método. Datos y lógica. Esto es lo que se define en muchos programas como la definición de una clase, que es la definición global y genérica de muchos objetos.



Los atributos determinarán las cualidades de nuestro objeto y los métodos las funcionalidades. Por ejemplo, si quisiésemos crear objetos de tipo coche, nuestros objetos tendrían unos atributos; marca, modelo, color, peso, etc. y unos métodos; arrancar, frenar, girar, etc.

Objeto	Propiedades	Métodos
	car.marca = Seat car.modelo = Leon car.peso = 950kg car.color = Rojo	car.arrancar () car.girar () car.frenar () car.acelerar ()

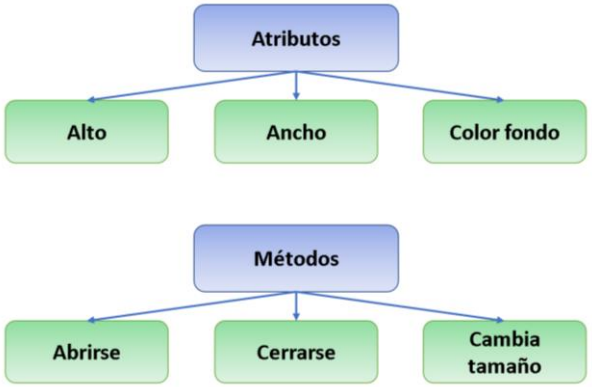
Cuando pasemos esto a código veremos que los atributos serán siempre variables (números, strings, etc.) y los métodos serán funciones.

El proceso de "pensar" en nuestro código como un conjunto de objetos es algo que, al principio, al alumno le puede costar bastante, pero en Python ya hemos estado usando objetos constantemente sin saberlo. Por ejemplo, la misma ventana de ejecución de un programa es un objeto, y como objeto que es, tiene atributos; un ancho, un alto, un color, una apariencia, etc. Y tiene funciones; se puede abrir, cerrar, ampliarse, puede mostrar información, etc.

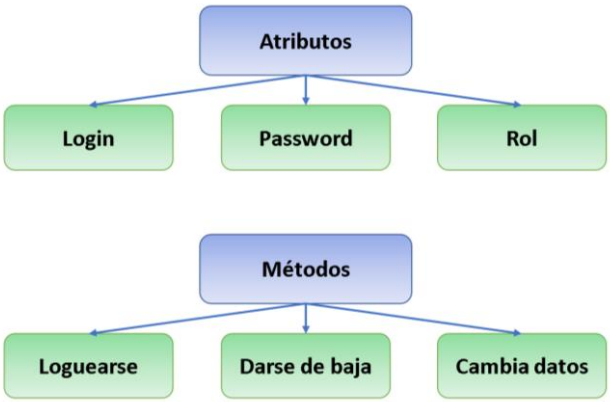
Un usuario de nuestro programa también sería un objeto. Sus atributos podrían ser tener un login, un password, unos privilegios, etc. Y sus métodos poder darse de alta, de baja, cambiar sus datos, etc.

Así pues, si lo pensamos, podremos deducir las propiedades y métodos de todos los elementos de un programa; botones, menús, etc.

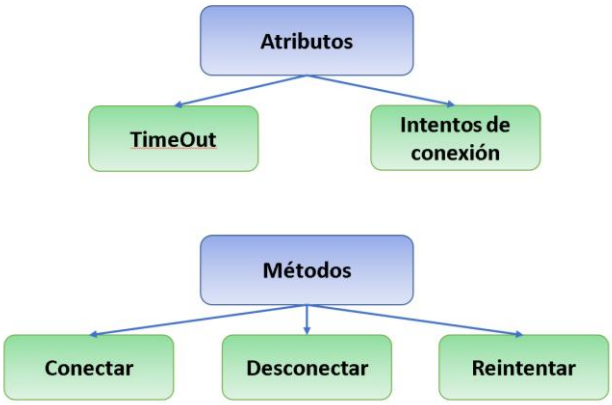
Un objeto de tipo ventana tendría los siguientes atributos y métodos (entre otros muchos):



Un objeto de tipo usuario podría tener los siguiente atributos y métodos:



Podríamos encontrarnos incluso con objetos cuyos atributos y métodos pueden parecer más difíciles de determinar en principio, como por ejemplo, una conexión a una base de datos:



Como vemos, el paradigma de la POO nos hará ver nuestro programa como un conjunto de objetos que se relacionan entre sí. En el siguiente capítulo veremos un concepto estrechamente relacionado con el de objeto, el concepto de clase, y cómo pasar a código nuestras clases y objetos.

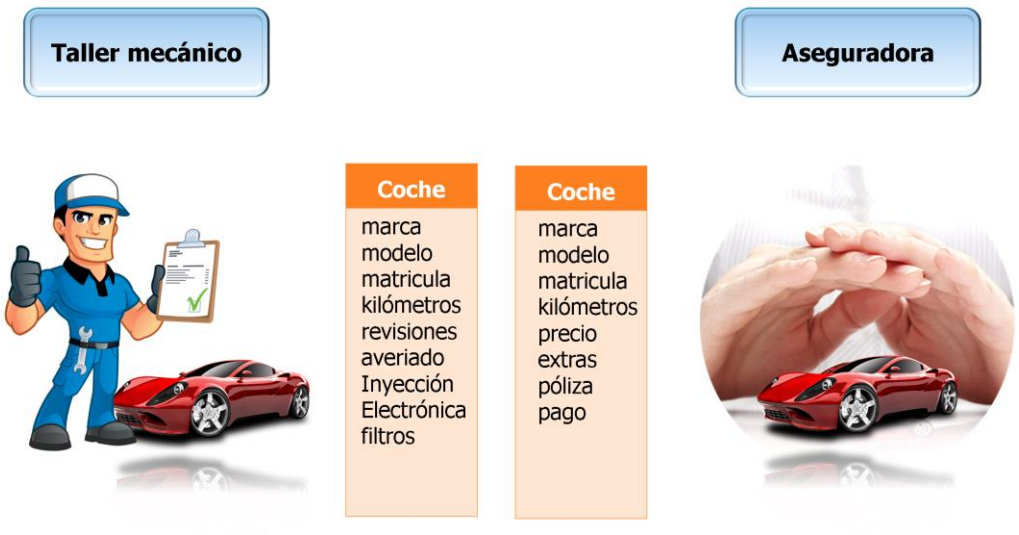
Abstracción

Estrechamente relacionado con los conceptos de clase y objeto se encuentra el concepto de abstracción.

Hemos visto varios ejemplos de objetos como el objeto ventana, usuario, conexión, etc. Dichos objetos poseen atributos y métodos que usaremos en nuestros programas. De hecho, cualquier objeto que creemos en nuestro programa tendrá atributos y métodos, y hablando estrictamente, se podrían sacar un número casi ilimitado de atributos y métodos de prácticamente cualquier objeto. Un coche, por ejemplo podría tener como atributos; alto, largo, ancho, peso, color, marca, modelo, matrícula, tipo, año, país de origen, si es nuevo o de segunda mano, si es de particular o de flota de empresa, etc. Por pura lógica no vamos a instanciar todas las posibilidades sino simplemente aquellas que nos interesan.

El concepto de abstracción consiste en saber elegir sólo los atributos y métodos que nos interesan para nuestro programa y desechar el resto. La definición estricta de abstracción sería “determinar las características específicas de un objeto, aquellas que lo distinguen de los demás tipos de objetos y que logran definir límites conceptuales respecto a quien está haciendo dicha abstracción del objeto”.

Supongamos que tenemos que implementar un programa para un taller de coches y otro para una compañía aseguradora. En ambos casos deberemos instanciar objetos de clase Coche, pero en el caso del taller nos interesarán unos datos sobre el coche y en el caso de la aseguradora, otros distintos. Por ejemplo, a la aseguradora le puede interesar si la póliza del seguro del coche va a ser pagada mensualmente o anualmente. Al dueño del taller poco le importa este dato.



Por lo tanto, una de las acciones que debemos realizar antes de ponernos a programar es tener claro los datos que vamos a necesitar introducir en nuestros objetos. Cuáles van a ser los atributos y métodos con los que necesitaremos trabajar.

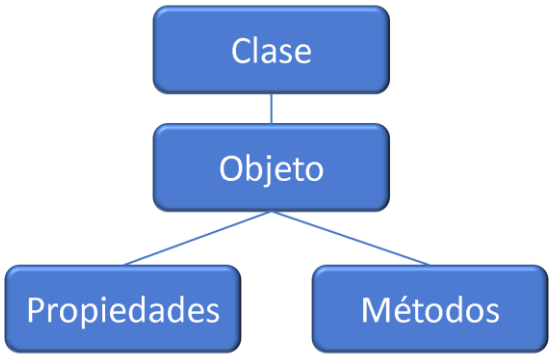
Clase

Hasta ahora hemos visto el concepto de objeto y hemos visto varios ejemplos de objetos que podemos incorporar a nuestros programas. Pero, ¿qué pasa si tenemos que crear decenas o cientos de objetos de tipo **coche**?. Es necesario programar uno a uno todos esos coches aunque tengan propiedades y métodos similares?. Aquí es donde entra en juego el concepto de clase.

Una clase es un modelo donde se redactan las características comunes a un grupo de objetos. Una clase representa al conjunto de objetos que comparten una estructura y un comportamiento comunes.

Dicho de otro modo, una clase no es más que un **molde para hacer objetos**, un conjunto de especificaciones que determinan cómo se van a comportar cierto tipo de elementos. Estos elementos, creados a partir de las especificaciones de la clase, es lo que llamamos objetos.

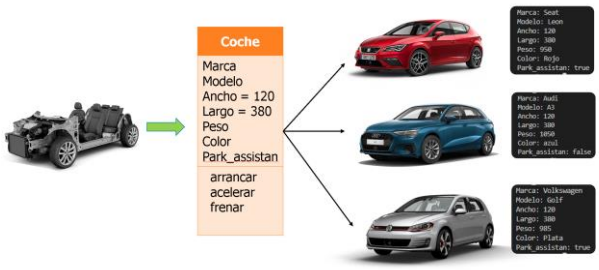
Así pues, a nuestro modelo de POO se le añade un nuevo elemento, las clases:



Siguiendo con el ejemplo que vimos anteriormente de nuestro **coche**, puede que necesitemos crear más coches con exactamente los mismos atributos, o similares, o incluso totalmente diferentes, pero que aun así se consideran coches. Entonces podemos decir que nuestro objeto **coche** se puede clasificar, es decir, nuestro objeto pertenece a una clase, concretamente a la clase **Coche**. Decimos entonces que nuestro **coche** es una instancia, objeto o ejemplar de la clase **Coche**.

Podemos crear una clase **Coche**, un molde para hacer coches, a ese molde le damos las características comunes a todos los objetos de la clase coche y con ese molde ya podemos crear objetos de la clase coche.

Podemos decidir si nuestros objetos de la clase coche (nuestras instancias de Coche o ejemplares de Coche) serán todos iguales o tendrán algunas propiedades comunes y otras específicas de cada coche.



En este ejemplo, hemos creado una clase **Coche** que servirá de molde para crear instancias de la clase coche. Hemos creado la clase **Coche** con un ancho y un largo ya determinado, luego las instancias de **Coche** tendrán todas el mismo ancho y largo, pero el resto de las propiedades están sin definir en la clase, tendremos que pasarlas al crear cada objeto en particular.

Lo normal es que una aplicación esté compuesta de varias clases, cada una responsable de un trabajo,

que se relacionan entre sí y con las que se crearán los objetos que necesite nuestro programa.

Ahora llevémonos todo esto a código.Para crear una clase, la sintaxis en Python es:

```
class NombreClase:
```

Por convenio los nombres de las clases comienzan en mayúsculas. Si no lo ponemos en mayúsculas no vamos a tener ningún fallo, es simplemente código de buenas prácticas de programación.

Dentro de la clase, indentados, irían los atributos y métodos la nuestra clase. Recordemos que los atributos serán variables (de hecho, también se llaman variables de clase) y los métodos son funciones.

Siguiendo con el ejemplo de nuestra clase coche, la implementación quedaría de la siguiente forma:

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

# Declaración de atributos
    largo = 250
    ancho = 120
    ruedas = 4
    peso = 900
    color = "rojo"
    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"
```

Al crear un método de clase hay que poner como primer parámetro obligatoriamente ***self***, es como el ***this*** de otros lenguajes de POO. Es una referencia al objeto instanciado en la clase.

El uso de ***self*** es necesario ya que es una referencia (un puntero) al objeto instanciado. Esta referencia ya que indica que un método o un atributo apuntado por ***self*** pertenece a una sola instancia y no a todos los objetos instanciados de la misma clase.

Al referenciar explícitamente a **self.is_enMarcha**, estamos ayudando al intérprete a localizar la referencia que queremos que encuentre. De hecho, si no referenciamos utilizando **self.is_enMarcha** el intérprete va a buscar directamente **is_enMarcha** fuera de la clase, por lo que si **is_enMarcha** no existiera a nivel de módulo, la búsqueda nos devolvería un error al no encontrar esa variable.

Hemos creado la clase coche y dentro hemos declarado:

- Una serie de atributos, que no son más que variables:
  - Largo, ancho, ruedas y peso, de tipo number.
  - Color, de tipo string.
  - Is_enMarcha, de tipo boolean.

- Una serie de métodos. Que son funciones:
  - Arrancar()
  - Estado()

Ya tenemos nuestra clase **coche** creada. Todas las instancias de la clase **coche** que creemos tendrán los atributos y métodos que hemos puesto en la clase. Luego les podremos cambiar su valor, pero en principio es el que tendrán.

Ahora sólo nos queda crear tantos ejemplares de clase (instancias de clase) como queramos. Vamos a crear dos. La sintaxis sería la siguiente:

```
# Declaración de una instancia de clase,
objeto de clase o ejemplar de clase.

miCoche = Coche()
miCoche2 = Coche()
```

Hemos creado dos objetos llamados **miCoche** y **miCoche2** que son dos instancias de la clase **Coche**, por lo tanto, tienen las propiedades y métodos pertenecientes a la clase **coche**. Ahora ya podemos usar los atributos y métodos de la clase coche en los ejemplares de hemos creado, para ello se usa la nomenclatura del punto, es decir: **nombreObjeto.atributo** o **nombreObjeto.propiedad**:

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

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

# Modificamos el valor de una propiedad
miCoche2.ruedas = 10
print("El coche2 tiene:" ,
miCoche2.ruedas, "ruedas.")
```

Veremos en temas posteriores, cuando veamos la encapsulación, que no es conveniente acceder a las propiedades y métodos de la clase de esta forma, pero de momento nos interesa simplemente quedarnos con el concepto de clase, objeto, atributos, métodos y la nomenclatura del punto para acceder a ellos.

Para finalizar este tema, vamos a ver otro ejemplo de creación de clase e instanciación de objetos de dicha clase.

Crearemos una clase **Usuario** que tendrá los siguientes atributos:

- nombre : string
- edad : number
- login : string
- password : string
- email : string
- telefono : number

Y los siguientes métodos:

- Resumen(): Sacará un resumen de los datos del usuario
- cambiaEdad(): Nos dará la posibilidad de pedir al usuario que introduzca una nueva edad.
- muestraEdad(): Nos mostrará la edad del usuario

Por último, crearemos una instancia de la clase Usuario a la que llamaremos administrador y llamaremos a los métodos de la clase Usuario para este administrador usando la nomenclatura del punto.

```
# CREACIÓN DE LA CLASE
class Usuario():

    # Declaración de atributos
    nombre = "Angel"
    edad = 47
    login = "admin"
    password = "1234"
    email = "angel@loquesea.com"
    telefono = 666666666
```

```
    # Declaración de métodos
    def resumen(self):  # self hace
referencia a la instancia de clase.
        print(f'Los datos del usuario
son:\n'
              f'Nombre: {self.nombre}\n'
              f'Edad: {self.edad}\n'
              f'Login: {self.login}\n'
              f'Password:
{self.password}\n'
              f'Email: {self.email}\n'
              f'Teléfono:
{self.telefono}')

    def cambiaEdad(self):
        edadIntroducida =
int(input("Introduce edad entre 18-
100:"))

        if 18 < edadIntroducida < 100:
            self.edad = edadIntroducida
            print("Edad introducida
correcta")
            return ""
        else:
            print("La edad introducida no
es correcta.")
            self.cambiaEdad()
            return ""


    def muestraEdad(self):
        print('La edad del usuario es:',
self.edad, 'años.')
        return ""

   # Creación de una instancia de la clase
Usuario a la que llamaremos administrador

administrador = Usuario()

   # Una vez creado el objeto administrador,
hacemos uso del método “resumen()” perteneciente
a la clase Usuario
administrador.resumen()
```

```
   # Usamos los métodos cambiaEdad() y
muestraEdad() de la clase Usuario.
print(administrador.cambiaEdad())
print(administrador.muestraEdad())
```

Si ejecutamos el programa, primero no sacará los datos del usuario, luego nos pedirá que introduzcamos la nueva edad y finalmente nos mostrará la edad introducida.

```
✓ class Usuario(): ···

···  Los datos del usuario son:
     Nombre: Angel
     Edad: 47
     Login: admin
     Password: 1234
     Email: angel@loquesea.com
     Teléfono: 666666666
     Edad introducida correcta

     La edad del usuario es: 55 años.
```