



# Clases y Objetos en Python

Como sabes, Python es un lenguaje de programación orientado a objetos. Si recuerdas de la sesión inicial, eso quiere decir que trabaja con elementos denominados objetos, que vienen definidos por unos templates o plantillas que llamamos clases. Es una manera de expresar en lenguaje máquina cosas de la vida real.

Los "objetos" de una clase son instancias de esa clase que determina los atributos y métodos de estos. En este notebook vamos a ver con un poco más de detalle el concepto de clase y cómo crear nuestras propias clases.

## Clases

### Contenidos

- [Clase vs Objeto](#)

Las clases son la manera que tenemos de describir los objetos. Hasta ahora hemos visto clases básicas que vienen incluidas en Python como *int*, *str* o clases algo más complejas como los *dict*. Pero, **¿y si queremos crear nuestros propios objetos?** En los lenguajes orientados a objetos tenemos la posibilidad de definir nuevos objetos que se asemejen más a nuestros casos de uso y hagan la programación más sencilla de desarrollar y entender.

**Un número entero es un objeto de la clase *int* que posee unas características diferentes a un texto**, que es de la clase *str*. Por ejemplo, **¿cómo sabemos que un coche es un coche?** ¿qué características tiene? Los coches tienen una marca, una cantidad de caballos, hay unos automáticos, otros no... De esta manera traducimos a lenguaje de máquina, a programación, un concepto que tenemos nosotros muy claro e interiorizado.

Hasta ahora, hemos visto varias clases, por ejemplo la clase `str`. Cuando veíamos el tipo de dato, Python imprimía por pantalla `str`. Y al ser `str`, tenía unas propiedades que no tenían otros objetos, como los métodos `upper()` o `lower()`.

La sintaxis para crear una clase es:

```
class NombreClase:
    # Cosas de la clase
```

Normalmente para el nombre de la clase se usa *CamelCase*, que quiere decir que se define en minúsculas, sin espacios ni guiones, y jugando con las mayúsculas para diferenciar palabras.

Mira cómo es la [clase `built\_in` de `String`](#)

```
In [2]: class Coche:
        # Atributos y métodos, Las cosas de la clase
        pass
```

La sentencia `pass` se usa para forzar el fin de la clase `Coche`. La hemos declarado, pero no lleva nada. Python demanda una definición de la clase y podemos ignorar esa demanda mediante la sentencia `pass`.

```
In [3]: print(type(Coche))
        print(type(str), type(dict))
```

```
<class 'type'>
<class 'type'> <class 'type'>
```

Bien, `coche` es de tipo `type`, claro porque **no es un objeto con tal**, sino que es una clase. Cuando creamos coches, estos serán de clase `Coche`, es decir, de tipo `Coche`, por lo que tiene sentido que `Coche` sea de tipo `type`.

## Clase vs Objeto

[al índice](#)

**La clase se usa para definir algo.** Al igual que con las funciones. Creamos el esqueleto de lo que será un objeto de esa clase. Por tanto, **una vez tengamos la clase definida, instanciaremos un objeto de esa clase.** Es como crear el concepto de coche, con todas sus características y funcionalidades. Después, a lo largo del programa, podremos crear objetos de tipo coche, que se ajusten a lo definido en la clase coche. Cada coche tendrá una marca, una potencia, etc...

```
In [4]: primer_coche = Coche()
        print(type(primer_coche))
        print(primer_coche)
```

```
<class '__main__.Coche'>  
<__main__.Coche object at 0x7f74f4717df0>
```


Ahora sí tenemos un objeto de tipo Coche, que se llama `primer_coche`. Cuando imprimimos su tipo, vemos que es de tipo Coche, y cuando lo imprimas el objeto por pantalla, simplemente nos dice su tipo y un identificador.

Podremos crear todos los coches que queramos

```
In [6]: citroen = Coche()  
        peugeot = Coche()  
  
        print(citroen==peugeot)
```

False

De momento todos nuestros coches son iguales, no hemos definido bien la clase, por lo que va a ser difícil diferenciar un coche de otro. Vamos a ver cómo lograr esa diferenciación.

imagen