

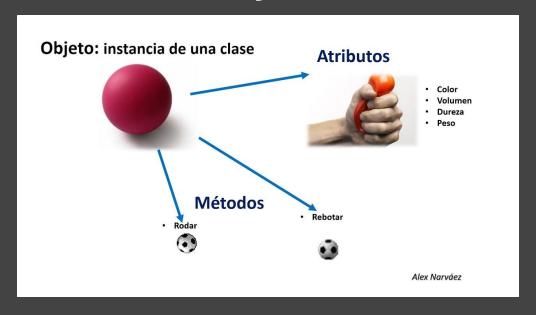
Clase 4

Repaso de lo que vimos en la #clase3

DECORADORES
RECURSIVIDAD
MÉTODOS
FUNCIONES PRE-DEFINIDAS
COMPRENSIÓN DE LISTAS
ÁMBITOS Y ALCANCES



UNIDAD VII CLASES, OBJETOS Y POO





TODO EN PYTHON ES UN OBJETO



Clase

Una clase es un *modelo o prototipo que define los atributos y métodos* comunes a todos los objetos de cierto grupo, un blueprint.

- Atributo: características que tendrán los objetos pertenecientes a la clase.
- Métodos: comportamientos que tendrán los objetos pertenecientes a la clase.



<u>Objeto</u>

Un objeto <u>es una entidad que agrupa un estado y una funcionalidad</u> <u>relacionadas</u>. <u>El estado del objeto se define a través de los atributos, mientras que la funcionalidad se modela a través de los métodos</u>.

Cuando se crea un objeto, si dice que se instancia la clase a la que pertenece, y luego, para acceder a sus atributos y métodos se utiliza la nomenclatura del punto.



```
self.letra = letra
    def cantar(self):
        for linea in self.letra:
           print(linea)
feliz cumpleanos = Cancion([":Qué los cumplas feliz!",
feliz cumpleanos.cantar()
musica ligera = Cancion(["De aquel amor de música ligera",
musica_ligera.cantar()
```



- El constructor __init__ : se ejecuta justo después de crear un nuevo objeto a partir de la clase, proceso que se conoce con el nombre de instanciación.
- Parámetro self: el primer parámetro de init y del resto de métodos de la clase es siempre self, y sirve para referirse al objeto actual. Este mecanismo es necesario para poder acceder a los atributos y métodos del objeto diferenciando, por ejemplo, una variable local mi_var de un atributo del objeto self.mi_var.
- Para crear un objeto se escribe el nombre de la clase seguido de cualquier parámetro que sea necesario entre paréntesis, excepto self. Estos parámetros son los que se pasarán al método __init__.



<u>Herencia</u>

En un lenguaje orientado a objetos, cuando hacemos que una clase (subclase) herede de otra clase (superclase) estamos haciendo que la subclase contenga todos los atributos y métodos que tenía la superclase. Esta acción también se suele llamar a menudo "extender una clase".



¿Qué ocurriría si quisiéramos especificar un nuevo parámetro a la hora de crear una subclase? Bastaría con escribir un nuevo método __init__ para la ella que se ejecutaría en lugar del __init__ de la súperclase. Esto es lo que se conoce como sobreescribir métodos. Ahora bien, puede ocurrir en algunos casos que necesitemos sobreescribir un método de la clase padre, pero que en ese método queramos ejecutar el método de la clase padre porque nuestro nuevo método no necesite más que ejecutar un par de nuevas instrucciones extra. En ese caso usaríamos la sintaxis super().__init__(*args) para llamar al método de igual nombre de la clase padre.



```
self.__nombre = nombre
       self.__anodelanz = ano_de_lanzamiento
       self. autor = autor
       self. disco = disco
       self.premio = []
          self.premio.append(i)
       self.premio[-1] = " y " + self.premio[-1]
       self.premio.pop()
       total_premios = self.premio + ult_premio
           print(f"{self._nombre} ha ganado los siguientes premios: {total premios}")
           print(f"{self.__nombre} no ha ganado premios")
           print(f"{self.__nombre} estuvo entre los tres temas más escuchados en {self.__anodelanz}")
           print(f"{self.__nombre} estuvo entre los dos temas más escuchados en {self.__anodelanz}")
           print(f"{self.__nombre} fue el tema más escuchado en {self.__anodelanz}")
class Letra(Cancion)
       self.__anodelanz = ano_de_lanzamiento
       self. autor = autor
       self. letra = letra
           print(f"\n\nLetra de {self.__nombre}\n")
           print(self.__letra)
letra cancion = """...""
letra = Letra("Another brick in the wall", "1979", "Pink Floyd", "The Wall", letra_cancion)
```



Si quisieran crear una clase y que herede métodos y atributos de más de una clase, la sintaxis es sencilla:

class Nombre (Súperclase1, Súperclase2,...):

Siendo el orden de importancia de izquierda a derecha, es decir que, si hay dos súperclases con atributos o métodos iguales, se toma el de la que está indicada primera en los parámetros. Esto se llama herencia múltiple.



Encapsulación

La encapsulación se refiere a impedir el acceso a determinados métodos y atributos de los objetos estableciendo así qué puede utilizarse desde fuera de la clase.

En Python no existen los modificadores de acceso, y lo que se suele hacer es que el acceso a una variable o función viene determinado por su nombre: si el nombre comienza con dos guiones bajos (y no termina también con dos guiones bajos) se trata de una variable o función privada, en caso contrario es pública



<u>Polimorfismo</u>

El polimorfismo se refiere a la habilidad de objetos de distintas clases de responder al mismo mensaje. Esto se puede conseguir a través de la herencia: un objeto de una clase derivada es al mismo tiempo un objeto de la clase padre, de forma que allí donde se requiere un objeto de la clase padre también se puede utilizar uno de la clase hija.



<u> POO</u>

La programación orientada a objetos es un paradigma de programación que busca representar entidades u objetos agrupando datos y métodos que puedan describir sus características y comportamientos.

En este paradigma, los conceptos del mundo real relevantes para nuestro problema a resolver se modelan a través de clases y objetos, y el programa consistirá en una serie de interacciones entre dichos objetos.



RESOLVER GUÍA 4