# PROGRAMACION ORIENTADA A OBJETOS POO

Acabamos de almacenar ciudades, es decir, un simple string dentro de un fichero. Qué sucede si deseamos almacenar más de un dato para cada ciudad, por ejemplo, el clima de la ciudad o la posición o la comunidad a la que pertenece…

Esto nos genera un problema porque necesitamos un objeto que NO tenemos, es decir, tenemos un String

Ciudad = “Alicante”

Pero en cuanto es algo más complicado el objeto, no tenemos nada:

Ciudad = “Alicante”, “20 grados”, “Comunidad Valenciana”

A partir de aquí es dónde entramos en la creación de objetos dentro del lenguaje Python.

Los objetos estarán compuestos por propiedades o métodos, por ejemplo:

Ciudad = nueva Ciudad()

Ciudad.nombre = “Alicante”

Ciudad.poblacion = “312000”

Ciudad.clima = “20 grados”

En este caso, es muy simple, los objetos se crean en ficheros externos y, mediante **def** podemos crear métodos y propiedades del objeto. En principio no es obligatorio crearlos en ficheros externos, pero si recomendable.

Podríamos crear un fichero llamado **claseciudad.py**

Al crear un fichero que representa una clase, debemos indicar el nombre de la clase dentro del archivo

class NombreClase

Propiedades y métodos

Una clase puede tener un **constructor**. Un constructor es el lugar que sirve para inicializar las variables de una clase.

Es el primer código que se leerá en cualquier clase.

En Python, un constructor se define con **\_\_init\_\_**

Las propiedades de un objeto son variables dentro de la clase y los métodos serán **def**

Cualquier método de la clase utiliza la palabra **self** para hacer referencia a los métodos o propiedades de la propia clase.

Ejemplo de una clase Lampara:

class Lampara:

# AQUÍ IRAN LAS PROPIEDADES

Iluminacion: 0

encendida: False

color: “Rojo”

# PODEMOS TENER UN CONSTRUCTOR PARA

# INICIAR EL VALOR DE LAS VARIABLES

def \_\_init\_\_(self):

self.iluminacion = 0

self.encendida = False

self.color = “Azul”

# METODOS DE LA CLASE

def endenderLampara(**self**):

self.encendida = True

self.iluminacion = 200

def apagarLampara(self):

self.encendida = False

self.iluminacion = 0

Para utilizar el objeto es muy simple

Objeto = Lampara()

Objeto.color = “Blanco”

Objeto.encenderLampara()

Vamos a realizar un ejemplo muy sencillo utilizando clases **Persona**

Tendremos, por un lado, la clase Persona y, por otro lado, una clase que manejará los objetos persona.

Ahora vamos a separar los nombres de archivo. Las clases las llamaremos **ClassXXPersona.py**

Creamos un fichero nuevo llamado **class29persona.py**

**CLASS33PERSONA.PY**

***class* Persona:**

**nombre = ""**

**apellidos = ""**

**email = ""**

**anyonacimiento = 0**

**pais = "Lo que sea..."**

**# El constructor es un método para iniciar**

**# los valores de las propiedades de la clase**

**# Por ejemplo, pongamos que queremos que cualquier**

**# persona, al crearla sea de una Pais: FRANCIA**

***def* \_\_init\_\_(*self*):**

***self*.*pais* = "Francia"**

**# METODO PARA RECUPERAR LOS DATOS COMPLETOS DE UNA PERSONA**

***def* getNombreCompleto(*self*):**

**return *self*.*nombre* + " " + *self*.*apellidos***

**# METODO PARA QUE NOS DEVUELVA LA EDAD**

***def* getEdad(*self*):**

**return 2022 - *self*.*anyonacimiento***

A continuación, nos creamos un programa para manejar a las personas. Llamamos a nuestro programa **python33mainpersonas.py**

Texto

Descripción generada automáticamente

**PYTHON33MAINPERSONAS.PY**

**# IMPORTAMOS LA CLASE PERSONA PARA TRABAJAR CON ELLA**

**from class33persona import Persona**

**print("Ejemplo clases persona")**

**# INSTANCIAMOS UN OBJETO DE LA CLASE PERSONA**

**person = Persona()**

**# Podemos cambiar la propiedad una vez creado el objeto**

**person.*pais* = "España"**

**print("Pais: " + person.*pais*)**

**# INCLUIMOS DATOS PARA LA PERSONA**

**person.*nombre* = "Alumno"**

**person.*apellidos* = "Python"**

**person.*email* = "alumnopython@gmail.com"**

**person.*anyonacimiento* = 2001**

**# LLAMAMOS A LOS METODOS**

**print(person.getNombreCompleto())**

**print("Edad: " + *str*(person.getEdad()))**

**# CREAMOS OTRO OBJETO Y LO VAMOS TAMBIEN**

**person2 = Persona()**

**print("Pais persona 2 " + person2.*pais*)**

**print("Fin de programa")**

Toda clase, al ser representada por pantalla, es devuelta mediante un STRING con un **id** que genera Python para el objeto en nuestro programa.

Por ejemplo, vamos a dibujar el objeto Persona al completo con un **print**





Dentro de las clases, también tenemos la posibilidad de personalizar cómo queremos que se represente la clase cuando un programador acceda al objeto directamente. Esto se realiza mediante un método en la clase llamado **\_\_def\_\_**

Imagen que contiene Texto

Descripción generada automáticamente

Por último, podemos recibir parámetros en los métodos de una clase, teniendo en cuenta que siempre recibirá **self** como primer parámetro.

Por ejemplo:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Vamos a realizar un programa para manejar objetos de la clase Coche.

Un coche podrá arrancar, acelerar, frenar y detener como métodos.

Como propiedades tendrá Marca, modelo, velocidad y estado.

Tendremos otra clase que será el conductor que utilizará el coche mediante un menú.

Creamos dos archivos: **class34coche.py** y **python34conductor.py**

Texto

Descripción generada automáticamente

**CLASS34COCHE.PY**

***class* Coche:**

**marca = ""**

**modelo = ""**

**velocidad = 0**

**estado = False**

***def* acelerar(*self*):**

**if (*self*.*estado* == False):**

**print("El coche no está arrancado.  Debe arrancar antes")**

**else:**

***self*.*velocidad* += 20**

**print("Velocidad actual " + *str*(*self*.*velocidad*))**

***def* frenar(*self*):**

**if (*self*.*velocidad* == 0):**

**print("El coche está detenido")**

**else:**

***self*.*velocidad* -= 10**

**print("Velocidad actual " + *str*(*self*.*velocidad*))**

***def* arrancar(*self*):**

***self*.*estado* = True**

**print("Coche arrancado")**

***def* detener(*self*):**

***self*.*estado* = False**

***self*.*velocidad* = 0**

**print("Coche detenido")**

**PYTHON34CONDUCTOR.PY**

**from class34coche import Coche**

**print("Conduciendo mi coche")**

**car = Coche()**

**car.*marca* = "Volkswagen"**

**car.*modelo* = "Escarabajo"**

**opcion = -1**

**while (opcion != 0):**

**print("------------------")**

**print("0.- Salir")**

**print("1.- Arrancar")**

**print("2.- Acelerar")**

**print("3.- Frenar")**

**print("4.- Detener coche")**

**print("Seleccione una opción")**

**opcion = *int*(input())**

**if (opcion == 1):**

**car.arrancar()**

**elif (opcion == 2):**

**car.acelerar()**

**elif (opcion == 3):**

**car.frenar()**

**elif (opcion == 4):**

**car.detener()**

**elif (opcion == 0):**

**print("Saliendo del programa")**

**else:**

**print("Opción incorrecta")**

**print("Fin de programa")**

# HERENCIA POO

La herencia en POO implica que un objeto se instancia a partir de otro objeto, es decir, un objeto recupera las propiedades de otro objeto.

No comenzamos de cero, sino que un objeto ya tendrá las características y métodos de otro objeto.

Por ejemplo, para entender esto…

Queremos crear una clase **Deportivo**.

Qué características tiene un deportivo??? Marca, Modelo, Velocidad, Estado, Arrancar(), Frenar()…

Lo único que diferencia a un Coche de un Deportivo es que el deportivo tiene **Turbo**

Si quisiéramos crear una clase Deportivo, deberíamos copiar todo el código de Coche, pegarlo en Deportivo y añadir Turbo, no?

La herencia nos permite indicar que un Deportivo ES un coche, por lo que tendrá todas las

características de un Coche y nosotros le añadimos las nuestras, por ejemplo, el Turbo.

Sintaxis para realizar la herencia:

class claseQueHereda (claseDeLaQueHeredamos):

Métodos y propiedades propios de la clase que Hereda

Vamos a comprobar el funcionamiento utilizando el ejemplo anterior del conductor.

Creamos una nueva clase llamada **class34deportivo.py**

**CLASS34DEPORTIVO.PY**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Por supuesto, podemos manejar las propiedades del coche dónde deseemos en el Deportivo.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Otro concepto dentro de la herencia es “adaptar” métodos de la clase **base** (Coche) a la clase **Deportivo**. Este concepto se conoce como **override**, lo que significa sobrescribir un método para adaptarlo a nuestra nueva clase.

Por ejemplo, un Deportivo acelera más que un coche, debemos modificar el código y hacerlo de nuevo en el Deportivo para cambiar su aceleración.

En Python, simplemente se escribe el método de nuevo tal y como esté escrito en la clase base.

Texto

Descripción generada automáticamente

**PYTHON34CONDUCTOR.PY**

**from class34coche import Coche**

**from class34deportivo import Deportivo**

**print("Conduciendo mi deportivo")**

**car = Deportivo()**

**car.*marca* = "Ferrari"**

**car.*modelo* = "Testarossa"**

**opcion = -1**

**while (opcion != 0):**

**print("------------------")**

**print("0.- Salir")**

**print("1.- Arrancar")**

**print("2.- Acelerar")**

**print("3.- Frenar")**

**print("4.- Detener coche")**

**print("5.- Turbo")**

**print("Seleccione una opción")**

**opcion = *int*(input())**

**if (opcion == 1):**

**car.arrancar()**

**elif (opcion == 2):**

**car.acelerar()**

**elif (opcion == 3):**

**car.frenar()**

**elif (opcion == 4):**

**car.detener()**

**elif (opcion == 5):**

**car.turbo()**

**elif (opcion == 0):**

**print("Saliendo del programa")**

**else:**

**print("Opción incorrecta")**

**print("Fin de programa")**

El concepto de **override** lo que hace es sustituir el código de la clase base/super

El código no es reutilizable, es sustituido completamente.

Vamos a realizar el siguiente supuesto:

* Un coche tendrá una velocidad Máxima de 140. Tendremos un método que nos dirá la velocidad máxima de un Coche (getVelocidadMaxima())
* Un Deportivo tendrá una velocidad máxima superior a un coche de 100, es decir, en un Deportivo, la velocidad será de 240.

Estos dos métodos nos **devolverán la velocidad máxima**

**COCHE**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Comprobamos que las dos clases realizan lo mismo

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Sustituimos (override) el método getVelocidadMaxima en Deportivo

**DEPORTIVO**

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Podremos comprobar que estamos **sustituyendo el código de la clase SUPER (Coche)**

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

Si necesitamos que se ejecute primero el método de la clase Coche y después el método de la clase Deportivo, debemos llamar al método utilizando una palabra clave en la herencia llamada **super**

**DEPORTIVO**

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.