¿Para qué usamos Clases en Python?

Python es un lenguaje orientado a objetos, lo que significa que se basa en objetos que pueden almacenar datos y tener un comportamiento concreto. Las clases nos permiten crear esos objetos, aparte de poder mantener el código de una manera más clara, modular y encapsulado, implica una mejora en código reutilizable.

Por ejemplo, supongamos que tenemos una clase Vehiculo que crea instancias (objetos), de vehículo. Esa clase tendrá un constructor con el que crear esas instancias y que asignará valores a sus atributos (las variables que almacenan los datos). Si nuestro constructor pidiese el número de ruedas que tendría, podría crear dos vehículos con el mismo código con sólo instanciar la clase de esta forma.

```
coche = Vehiculo(4)
moto = Vehiculo(2)
```

Por otro lado, esas instancias tendrán un comportamiento asignado con los métodos internos de la clase. Tal vez para este caso podríamos tener un método que arranque el coche.

```
def arrancar_motor(self):
  print(“El motor está en marcha”)

coche.arrancar_motor()  # Imprime “ El motor está en marcha”
```

¿Qué método se ejecuta automáticamente cuando se crea una instancia de una clase?

El método que se ejecuta es el constructor y que identificamos con la palabra reservada __init__.

El constructor de nuestra clase Vehiculo sería:

```
def __init__(self, ruedas):
   self.ruedas = ruedas  #Lo que asignaría al atributo ruedas en número de ruedas típico deseado.
```

Y podríamos llamarlo para crear la instancia de la forma indicada antes:

```
coche = Vehiculo(4)
```

¿Cuáles son los tres verbos de API?

Con una Interfaz de Programación de Aplicaciones podemos interactuar con una aplicación o servidor, para acceder o manejar sus datos. Para ello, las solicitudes principales son GET, POST y DELETE.

Cuando usamos GET lo que estamos haciendo es solicitar la información de nuestro servidor y que nos la muestre de una forma concreta. Al usar la opción POST transmitimos información para que la almacene. Como última opción, con DELETE tenemos la de borrar un recurso concreto en el servidor.

¿Es MongoDB una base de datos SQL o NoSQL?

MongoDB no es una base de datos relacional basada en tablas y columnas. Su sistema es de colecciones, que puede ser lo más parecido a esas tablas, y la ventaja que ofrece es el poder tratar ingentes cantidades de documentos reales.

¿Qué es una API?

La Interfaz de Programación de Aplicaciones es un conjunto de reglas y protocolos que permiten que dos aplicaciones se puedan comunicar entre sí. Esto puede ahorrar tiempo de programación, porque si tenemos el ejemplo de un software de venta, éste puede utilizar las API del gestor del almacén para consultar el stock que hay en proveedor, y la API del banco para confirmar que el pago por tarjeta sea correcto, mientras que el software de venta sólo se encarga de la facturación.

¿Qué es Postman?

Postman es una plataforma que nos permite diseñar APIs con mayor facilidad permitiendo diseñar, testear, crear documentación y simular nuestra API.

¿Qué es el polimorfismo?

Es la forma de dar diferentes respuestas a una misma llamada, y esto está muy ligado a la herencia de clases.

Supongamos que tenemos la clase Empleado y que de ella heredan dos clases, Directivo y Conserje. Todos serán empleados pero cada uno tendrás sus funciones e ingresos.

```
class Empleado:
  def __init__(self, sueldo):
    self.sueldo = sueldo

  def cargo(self):
     raise NotImplementedError("Subclass must implement render method")


class Directivo(Empleado):
  def cargo(self):
     print(f"Soy el que da órdenes y gano {self.sueldo}€" )
```

```
class Conserje(Empleado):
  def cargo(self):
     print(f"Soy el que atiende a la gente y gano {self.sueldo}€")

ana = Directivo(100000)
pedro = Conserje(20000)

ana.cargo()   # Imprime: Soy el que da órdenes y gano 100000€

pedro.cargo()   # Imprime: Soy el que atiende a la gente y gano 20000€
```

Usamos la misma función, pero tiene diferentes comportamientos y manejo de datos.

¿Qué es un método dunder?

Son métodos “especiales” que se forman con dos guiones bajos al principio, una palabra reservada y dos guiones bajos otra vez. Son utilizados para funciones específicas y los ejemplos más claros son los de __init__ y __repr__, pero hay más como __str__, __len__.

__init__ se utiliza para el constructor:

```
def __init__(self, sueldo):
```

__repr__ se usa para representar el objeto en forma de cadena de texto:

```
def __repr__(self):
  return f”Empleado <Sueldo: {sueldo}>”
```

¿Qué es un decorador de python?

Para proteger las variables internas de una clase utilizamos el guion bajo para que Python no te permita el acceso y su modificación. Pero si nos interesa hacerlo, necesitamos a los decoradores.

Si queremos acceder a una propiedad, usaremos @property para crear una función con el nombre de esa propiedad en la clase y que nos devuelva su dato.

```
...
self._sueldo = 100000  #Propiedad protegida
...
@porperty    #Acceso a la propiedad
def sueldo(self):
  return self._sueldo
```

Y si queremos modificar la propiedad:

```
@sueldo.setter  #Modificación de la propiedad
def sueldo(self, sueldo):
 self._sueldo = sueldo
```