***¿Para qué usamos Clases en Python?**
Las clases en Python sirven de molde o plantilla que utilizaremos para objetos que compartiran los mismos atributos o métodos. Por ejemplo podemos tener un molde para crear un coche, el cual tendrá los mismos componentes que serían como los atributos de la clase como puede ser el motor, la carrocería, el color, las ruedas y por otro lado sus funciones que equivaldrían a los métodos como arrancar, acelerar, frenar,etc. Con ese molde podriamos fabricar diferentes modelos de coche.

La sintaxis para definir una clase es utilizando la palabra “class” seguido del nombre de la clase y dos puntos “:”. Siguiendo con el anterior ejemplo vamos a crear una clase para generar coches:

```
class Coche:
    def __init__(self, marca, color, motor):
        self.marca = marca
        self.color = color
        self.motor = motor
```

No os preocupéis si la sintaxis del método __init__ es nueva para vosotros, simplemente se trata de un método algo similar a una función. Más adelante explicaremos como funciona y para que se crea.

Algunas de las ventajas de las clases son las siguientes:

- Ahorro de tiempo y no duplicar códgio. Las clases se pueden utilizar en diferentes partes del programa sin necesidad de reescribir el código. Es un comportamiento similar al de la funciones, se crean una vez y se pueden reutilizar cunado se necesiten.
- Permiten ocultar la complejidad de un objeto simplificando su usabilidad y la interacción con el mismo. Esto se denomina encapsulación.
- Pueden descomponer un programa en componentes más pequeños y manejables. Facilita la resolución de problemas. Lo descrito se define como modularidad.
- Ayudan a implementar el mismo conjunto de métodos con diferentes comportamientos para distintos tipos de objetos, lo que permite una mayor flexibilidad y extensibilidad en el diseño de programas. Esto se defne como polimorfismo.

Entre las desventajas que presentan las clases podemos encontrar las siguientes:

- Las clases pueden agregar complejidad adicional a un programa y hacer que sea más difícil de entender y depurar.
- El aprendizaje de las clases y la programación orientada a objetos en general pueden requerir una curva de aprendizaje más pronunciada para los programadores principiantes.

- A veces, las clases se utilizan innecesariamente en situaciones en las que una función simple podría haber hecho el trabajo de manera más eficiente.

Os recomiendo consultar más información sobre las clases en los siguientes enlaces y escribir una clase en vuestro editor de texto favorito. Os ayudará a familiarizaros con la sintaxix e ir interiorizando los conceptos necesarios para crear las clases. Ánimo!!

https://docs.python.org/es/3/tutorial/classes.html
https://blog.hubspot.es/website/clases-python
https://tutorial.recursospython.com/clases/

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

El método __init__ es el que es llamado automáticante al crear una instancia de una clase. Sirve para asignar valores a los atributos que se hayan establecido para la clase. En el ejemplo del coche al crear una instancia de esa clase se ejecutará el método __init_ y se crearan los atributos de marca, motor y color.

A continuación creo una clase denominada “persona” y la sintaxis del método __init__ con los parámetros de nombre, edad y sexo:

```
class Persona:
        def __init__(self, nombre, edad, sexo):
                self.nombre = nombre
                self.edad = edad
                self. sexo = sexo
```

Podréis encontrar más información en los siguientes enlaces:
https://www.tutorialesprogramacionya.com/pythonya/detalleconcepto.php?punto=44&codigo=44&inicio=30
https://geekflare.com/es/init-in-python/
https://blog.hubspot.es/website/clases-python

***¿Cuáles son los tres verbos de API?**
El acrónimo API viene de “Application Programming Interface”. Se trata de una pieza de código que permite comunicarse a diferentes aplicaciones entre si y compartir información y funcionalidades. Difernciamos las API y las API REST. Las API se utilizan para acceder a datos en servidores remotos de manera más eficiente y las API REST son las más utilizadas para la integración de datos, lo que facilita la transferencia eficiente de datos.

El intercamibo de datos se realiza a través de una seria de protocolos. El más común es http y se utilizan los siguientes verbos en las API REST: GET (obtener información), POST (publicar información), PUT(actualizar datos) y DELETE (borrar datos).

Encontraréis información relevante sobre lo comentado en estos enlaces:
https://codigofacilito.com/articulos/rails-verbos-http
https://aws.amazon.com/es/what-is/api/
https://www.xataka.com/basics/api-que-sirve
https://www.bbvanexttechnologies.com/blogs/como-utilizar-los-metodos-put-y-patch-en-el-diseno-de-tus-apis-restful/

***¿Es MongoDB una base de datos SQL o NoSQL?**
MongoDB es una base de datos NoSQL. Para gestionar las bases de datos existe un lenguaje de consulta estructurada denominado SQL. Para realizar las operaciones de consulta u otras que necesitemos con las bases de datos utilizaremos ese lenguaje, sin embargo, MongoDB al ser una BBDD NoSQL funciona con un lenguaje similar al JavaScript y concretamente al formato de texto JSON y que forma parte de JavaScirpt.

Podéis encontrar más información en las siguientes plataformas:
https://aws.amazon.com/es/nosql/
https://www.mongodb.com/es
https://kinsta.com/es/base-de-conocimiento/que-es-mongodb/

***¿Qué es una API?**
El acrónimo API viene de “Application Programming Interface”. Se trata de una pieza de código que permite comunicarse a diferentes aplicaciones entre si y compartir información y funcionalidades. Diferneciamos las API y las API REST. Las API se utilizan para acceder a datos en servidores remotos de manera más eficiente y las API REST son las más utilizadas para la integración de datos, lo que facilita la transferencia eficiente de datos.

Tenéis definiciones más extensas y completas en los siguientes enlaces:
https://aws.amazon.com/es/what-is/api/
https://www.xataka.com/basics/api-que-sirve
https://www.ibm.com/es-es/topics/api

***¿Qué es Postman?**
Postman es una herramienta que nos permite realizar peticiones de una forma sencilla para probar APIs de tipo REST propias o de terceros. Al inicio se trataba de una extensión de Google Chrome pero actualmente se puede integrar como una aplicación nativa para los sistemas operativos Windows, Mac y Linux. Hay una versión gratis y otras de pago. Si queréis consultar los planes podéis hacerlo en: https://www.postman.com/

Encontraréis más infomración en las siguientes Webs:
https://assemblerinstitute.com/blog/que-es-postman/

https://formadoresit.es/que-es-postman-cuales-son-sus-principales-ventajas/
https://openwebinars.net/blog/que-es-postman/

***¿Qué es el polimorfismo?**
El polimorfismo es la capacidad de objetos creados de diferentes clases responder al mismo mensaja, es decir, dos objetos de diferentes clases pueden tener métodos con el mismo nombre, y ambos métodos pueden ser llamados con el mismo código, dando respuestas diferentes.

El polimorfismo está estrechamente ligado con el concepto de herencia. La herencia surge cuando creamos una clase “padre” y una o varias clases “hijo”.
Vamos a poner un ejemplo creando una clase padre donominanda “animal” y dos clases hijas “perro” y “gato” para entender los conceptos de polimorfismo y herencia:

```
class Animal:
        def hacerRuido(self):
                raise NotImplementedError

class Gato(Animal):
        def hacerRuido(self):
                print("Miiiauuuuu")

class Perro(Animal):
        def hacerRuido(self):
                print("Guuaaauuu")
```

En este ejemplo se puede ver la sintaxis para crear una clase padre con el método “hacerRuido” y la creación de las dos clases “hijo” que simplemente se consigue definiendo la clase y despues del nombre de ésta poniendo el nombre de la clase padre entre paréntesis. Como se puede ver las clases hijo heredan el método “hacerRuido” pero en cada una se especifica el funcionamiento del mismo.

Más información en:
https://apuntes.de/python/herencia-y-polimorfismo-en-python-amplia-la-flexibilidad-de-tus-clases/#gsc.tab=0
https://www.guru99.com/es/polymorphism-in-python.html
https://ellibrodepython.com/polimorfismo-en-programacion

***¿Qué es un método dunder?**

Los métodos dunder son operaciones de bajo nivel y están por debajo o ocultos de lo que vemos en el código. Estos métodos tienen un significado particular para el intérprete de Python. Sus nombres empiezan y terminan en __ (doble guión bajo). Por ejemplo **init**. Normalmente estos métodos no son invocados directamente por el programador. Por ejemplo cuando haces una simple suma 2 + 2 se está invocando al método __add__ internamente.

Por ejemplo vamos a crear una clase con el método dunder __missing_ que nos devuelva un mensaje en el caso de que queramos acceder al valor de una “key” que no exista en ese diccionario:

```
class Diccionario(dict):
    def __missing__(self, key):
        return "Esta key no existe en este diccionario"

mi_dicc = Diccionario()
mi_dicc["nombre"]= "Josu"
mi_dicc["apellido"]= "Beltran de Heredia"
print(mi_dicc["nombre"])
print(mi_dicc["edad"])
````
Resultado:
Josu
Esta key no existe en este diccionario

El proceso que sigue este código es: se crea una clase con el nombre de Diccionario a la cual se le pasa un diccionario como parámetro, se define el método __missing__ al que se le pasa una key como parámetro y este método si no encuentra la key en el diccionario que le hemos pasado nos devuelve el mensaje "Esta key no existe en este diccionario". Después creamos un diccionario mediante una instancia de la clase de esta forma:

```
mi_dicc = Diccionario()
```

A continuación, introducimos las keys nombre y apellido para las cuales les asignamos los valores correspondientes. Finalmente imprimimos el valor de la key nombre e intentamos imprimir el valor de la key edad. Como esta última key no existe en el diccionario, el método duder missing es llamado y nos devuelve el mensaje que hemos configurado.

Os aconsejo consultar los siguientes enlaces para ampliar información y practicar con los métodos dunder:
https://barcelonageeks.com/dunder-o-metodos-magicos-en-python/
https://www.netinetidesign.com/post/metodos-especiales-en-python/
https://geekflare.com/es/magic-methods-in-python/

***¿Qué es un decorador de python?**
Un decorador en Python es una función que recibe otra función como parámetro, le añade cosas y retorna una función diferente.

A través de los decoradores seremos capaces reducir las líneas de código duplicadas, haremos que nuestro código sea legible, fácil de testear, fácil de mantener.

En el siguiente ejemplo vamos a crear un decorador que convierta una cadena a mayúsculas y lo vamos a aplicar a una función que nos devuelva un saludo:

```
def convertir_mayus(func):
    def modificar():
        return func().upper()

    return modificar

@convertir_mayus
def saludo():
    return 'hola, soy un desarrollador de Python!'

print(saludo())
```

Resultado:

HOLA, SOY UN DESARROLLADOR DE PYTHON!

Podéis encontrar más informaición sobre los decoradores en los siguientes enlaces:
https://codigoencasa.com/dominando-los-decoradores-de-python-una-guia-completa-para-mejorar-la-modularidad-y-funcionalidad-del-codigo/#:~:text=Los%20decoradores%20son%20una%20potente,modificar%20directamente%20su%20c%C3%B3digo%20fuente.
https://codigofacilito.com/articulos/decoradores-python
https://platzi.com/blog/decoradores-en-python-que-son-y-como-funcionan/
https://www.freecodecamp.org/espanol/news/python-decorador-property/