DOCUMENTACIÓN

¿Para qué usamos Clases en Python?
En Python, usamos clases para organizar nuestro código de manera más ordenada y estructurada. Con clases, podemos agrupar datos y funciones relacionadas, lo que nos permite escribir programas más estructurados y fáciles de entender.

¿Por qué se utilizan?

- Las clases nos permiten organizar nuestro código en unidades lógicas y reutilizables.
- Nos ayudan a simplificar los detalles complicados y concentrarnos en cómo se comporta un objeto en general.
- Una vez definida una clase, podemos crear múltiples objetos (instancias) basados en esa clase, lo que nos permite reutilizar código.
- Las clases nos permiten agrupar datos y funciones juntos, lo que nos permite controlar quién puede acceder a ellos desde fuera de la clase.

Ejemplo:

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

    def saludar(self):
        print(f"Hola, soy {self.nombre} y tengo {self.edad} años.")

persona1 = Persona("Izei", 17)

persona1.saludar()
```

Output:

```
Hola, soy Izei y tengo 17 años.
```

¿Qué método se ejecuta automáticamente cuando se crea una instancia de una clase?
El método que se ejecuta automáticamente cuando se crea una instancia de una clase en Python se llama **__init__()**. Este método se conoce como el método de inicialización de la clase.

Cuando creamos una instancia (objeto) de una clase, Python automáticamente llama al método **__init__()** de esa clase si está definido. Este método se utiliza para inicializar los atributos de la instancia y realizar cualquier configuración inicial.

¿Por qué se utiliza?

- Se utiliza para garantizar que los objetos de una clase estén correctamente inicializados con los valores adecuados cuando se crean.
- Permite configurar el estado inicial de un objeto y realizar cualquier tarea de inicialización necesaria.

Ejemplo:

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

persona1 = Persona("Izei", 17)
```

¿Cuáles son los tres verbos de API?

GET: Se utiliza para solicitar datos para un recurso específico o una colección de recursos del servidor. Por ejemplo, puede realizar una solicitud GET a la API de Películas para obtener detalles sobre una película en particular o una lista de todas las películas disponibles.

POST: Se utiliza para enviar datos al servidor para crear un nuevo recurso. Por ejemplo, puede publicar un mensaje nuevo o crear un usuario nuevo enviando una solicitud POST a una API de redes sociales.

PUT o PATCH: Se utiliza para actualizar los recursos existentes en el servidor. La diferencia entre PUT y PATCH es cómo se actualiza el recurso.

¿Qué es Postman?

Postman es una herramienta de desarrollo de API que permite crear, probar, documentar y compartir API rápidamente.

Características principales de Postman:

- Creación de Solicitudes HTTP

- Pruebas Automatizadas
- Entornos y Variables
- Documentación de API

¿Qué es el polimorfismo?

El polimorfismo es un concepto orientada a objetos (POO) y se refiere a la capacidad de objetos de diferentes clases para responder al mismo mensaje o método de diferentes maneras.

El polimorfismo permite que objetos de diferentes clases implementen métodos con el mismo nombre pero con comportamientos específicos para cada clase.

Hay dos tipos principales de polimorfismo:

Overloading: Se refiere a la capacidad de una función o método para realizar diferentes tareas según los tipos o cantidad de parámetros que recibe.

Overriding: Se refiere a la capacidad de una clase hija para proporcionar una implementación específica de un método que ya está definido en su clase padre.

El polimorfismo es una herramienta poderosa que permite escribir código más flexible, modular y reutilizable en la programación orientada a objetos.

¿Qué es un método dunder?

es un tipo de método especial que su nombre comienza y termina con dos guiones bajos (__). Estos métodos se usan para realizar operaciones específicas en objetos, como la inicialización, representación, operaciones aritméticas, comparaciones, entre otros.

Ejemplo:

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

    def __str__(self):
        return f"Persona: {self.nombre}, {self.edad} años"

persona1 = Persona("Izei", 17)

print(persona1)
```

¿Qué es un decorador de python?

Un decorador en Python es una función que toma otra función como argumento y devuelve una nueva función, normalmente extendiendo o cambiando el comportamiento de la función original de alguna manera.

Los decoradores se utilizan comúnmente para:

- Agregar funcionalidad
- Modificar comportamiento
- Reutilizar código