CHECKPOINT 6

¿Para qué usamos Classes en Python?

En Python, las clases se utilizan para crear objetos que tienen propiedades y métodos asociados. Aquí hay algunas razones comunes por las que se usan clases en Python:

- **Abstracción y encapsulación:** Las clases permiten definir una estructura de datos que encapsula datos y comportamientos relacionados. Esto ayuda a ocultar los detalles internos de cómo funciona un objeto y proporciona una interfaz clara para interactuar con él.

- **Reutilización de código:** Las clases facilitan la reutilización del código al permitir la creación de múltiples objetos que comparten la misma estructura y comportamiento. Esto reduce la duplicación de código y facilita la mantenibilidad del programa.

- **Herencia:** Python admite la herencia, lo que significa que una clase puede heredar atributos y métodos de otra clase base. Esto permite la creación de jerarquías de clases, donde las clases derivadas pueden extender o modificar el comportamiento de la clase base.

- **Polimorfismo:** El polimorfismo permite que objetos de diferentes clases se utilicen de manera intercambiable en lugares donde se espera un tipo común. Esto facilita la escritura de código que pueda manejar una variedad de objetos de manera genérica.

- **Organización del código:** Las clases ayudan a organizar el código de manera más estructurada y modular, lo que facilita la comprensión y el mantenimiento del código, especialmente en proyectos grandes y complejos.

En resumen, las clases en Python proporcionan una forma eficiente y poderosa de modelar objetos del mundo real y desarrollar software modular y escalable.

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

Cuando se crea una instancia de una clase en Python, el método que se ejecuta automáticamente se llama __init__(). Este método se conoce como el método de inicialización o constructor de la clase. Es utilizado para realizar cualquier inicialización necesaria de los atributos del objeto.

Por ejemplo, si tienes una clase llamada Persona, el método __init__() puede ser utilizado para inicializar los atributos de nombre, edad, etc., cuando se crea una nueva instancia de la clase Persona.

Aquí hay un ejemplo básico:

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

# Creando una instancia de la clase Persona
persona1 = Persona("Juan", 30)

print("Nombre:", persona1.nombre)
print("Edad:", persona1.edad)
```

En este ejemplo, cuando se crea la instancia persona1 de la clase Persona, se ejecuta automáticamente el método __init__(), que inicializa los atributos nombre y edad con los valores proporcionados.

¿Cuáles son los tres verbos de API?

Los tres verbos principales de API son:

- **GET:** Este verbo se utiliza para solicitar datos de un recurso específico o para recuperar información de un servidor. Por ejemplo, al hacer una solicitud GET a una API de clima con la URL /weather, se puede recuperar la información del clima actual.

- **POST:** El verbo POST se utiliza para enviar datos al servidor para crear o actualizar un recurso. Por ejemplo, al enviar un formulario en una página web, los datos se envían al servidor utilizando una solicitud POST.

- **DELETE:** Este verbo se utiliza para eliminar un recurso específico en el servidor. Al hacer una solicitud DELETE a una URL de recurso, se elimina el recurso asociado en el servidor.

Estos tres verbos, junto con otros como PUT (para actualizar un recurso), PATCH (para actualizar parcialmente un recurso) y OPTIONS (para obtener información sobre los métodos HTTP soportados por un servidor), son los más comúnmente utilizados en las operaciones de API RESTful.

¿Es MongoDB una base de datos SQL o NoSQL?

MongoDB es una base de datos NoSQL. No sigue el modelo relacional utilizado en las bases de datos SQL tradicionales, como MySQL o PostgreSQL. En cambio, MongoDB es una base de datos de documentos, lo que significa que almacena datos en documentos similares a JSON con un esquema flexible. Esta flexibilidad lo hace especialmente adecuado para aplicaciones con necesidades de escalabilidad y donde los datos pueden tener una estructura variable.

¿Qué es una API?

API son las siglas en inglés de "Application Programming Interface" (Interfaz de Programación de Aplicaciones). En términos generales, una API es un conjunto de reglas, protocolos y herramientas que permiten que diferentes aplicaciones se comuniquen entre sí.

En el contexto del desarrollo de software, una API se refiere típicamente a un conjunto de definiciones y protocolos que permite a los desarrolladores de software integrar fácilmente funcionalidades específicas en sus propias aplicaciones sin necesidad de entender los detalles internos de cómo funciona esa funcionalidad. En otras palabras, proporciona un conjunto de métodos y funciones bien definidos que los desarrolladores pueden utilizar para interactuar con una aplicación, un servicio web o un sistema operativo.

Las APIs pueden ser de diferentes tipos, incluyendo:

- **APIs web:** Son APIs que se acceden a través de la web utilizando protocolos estándar como HTTP. Estas APIs permiten la comunicación entre sistemas a través de Internet.

- **APIs de biblioteca:** Son APIs que se proporcionan como parte de una biblioteca de software o un framework y se utilizan para interactuar con ese software específico.

- **APIs de sistema operativo:** Son APIs proporcionadas por un sistema operativo que permiten a las aplicaciones acceder a los recursos del sistema, como el sistema de archivos, la red o los dispositivos de entrada/salida.

En resumen, una API actúa como un intermediario que permite que diferentes componentes de software se comuniquen y trabajen juntos de manera eficiente y coherente. Es una herramienta fundamental en el desarrollo de software moderno y facilita la integración y la interoperabilidad entre sistemas y aplicaciones.

¿Qué es una Postman?

Postman es una plataforma de colaboración para el desarrollo de APIs que ofrece una variedad de herramientas para probar, desarrollar y documentar APIs. Es ampliamente utilizado por desarrolladores de software y equipos de desarrollo para simplificar el proceso de trabajar con APIs.

Las principales características de Postman incluyen:

- **Envíos de solicitudes HTTP:** Postman permite enviar solicitudes HTTP, como GET, POST, PUT, DELETE, etc., a cualquier API y ver las respuestas correspondientes.

- **Colecciones:** Los usuarios pueden organizar solicitudes relacionadas en colecciones, lo que facilita la gestión y la ejecución de grupos de solicitudes.

- **Pruebas automatizadas:** Postman permite escribir y ejecutar pruebas automatizadas para verificar la integridad y el comportamiento de las APIs.

- **Entornos y variables:** Postman permite definir variables y entornos para simplificar la configuración y la reutilización de valores en las solicitudes.

- **Documentación:** Postman ofrece herramientas para generar documentación automática para APIs, lo que facilita la comprensión y el uso de las APIs por parte de otros desarrolladores.

En resumen, Postman es una herramienta poderosa y versátil que facilita el desarrollo, la prueba y la documentación de APIs, lo que ayuda a los equipos de desarrollo a colaborar de manera más eficiente y a construir APIs de alta calidad.

¿Qué es el polimorfismo?

El polimorfismo es un concepto en la programación orientada a objetos que permite que objetos de diferentes clases puedan ser tratados de manera uniforme si pertenecen a la misma jerarquía de clases. Esto significa que un objeto puede tomar muchas formas, es decir, puede referirse a diferentes tipos de objetos y, sin embargo, invocar métodos que se comportan de manera específica para cada tipo.

Hay dos tipos principales de polimorfismo:

- Polimorfismo de sobrecarga (overloading): Se refiere a la capacidad de una función o método para realizar diferentes tareas en función de los parámetros que recibe. Por ejemplo, en muchos lenguajes de programación, es posible definir varias funciones con el mismo nombre pero con diferentes listas de parámetros.

- Polimorfismo de subtipos (subtyping): Se refiere a la capacidad de un objeto de una clase derivada (subclase) para ser tratado como un objeto de su clase base (superclase). Esto significa que una variable de tipo superclase puede hacer referencia a un objeto de la subclase, y aún así se comportará de acuerdo con los métodos específicos de la subclase. Esto se logra mediante la herencia y la implementación de métodos comunes en las clases base y derivadas.

El polimorfismo es una característica clave de la programación orientada a objetos que promueve la reutilización del código, la flexibilidad y la modularidad en el diseño del software. Permite escribir código más genérico y flexible que puede adaptarse a diferentes tipos de objetos sin necesidad de conocer los detalles específicos de cada uno.

¿Qué es un método dunder?

Un método dunder (también conocido como "método especial" o "método mágico") en Python es un método con un nombre que comienza y termina con doble guion bajo (__). Estos métodos son predefinidos y tienen un propósito específico en Python. Son utilizados para realizar operaciones especiales o sobrecargar comportamientos predeterminados en las clases.

Algunos ejemplos comunes de métodos dunder incluyen:

- __init__: El método de inicialización, que se llama automáticamente cuando se crea una nueva instancia de la clase.

- __str__: El método que devuelve una representación de cadena legible para humanos del objeto

- __repr__: El método que devuelve una representación de cadena del objeto que es útil para la depuración y la inspección del objeto.

- __len__: El método que devuelve la longitud del objeto.

- __add__, __sub__, __mul__, etc.: Métodos para sobrecargar operadores aritméticos como la suma, la resta, la multiplicación, etc.

Estos métodos dunder proporcionan una forma de personalizar el comportamiento de las clases y hacer que las instancias de esas clases se comporten de manera más intuitiva o predecible en diferentes contextos. Son una parte integral de la programación orientada a objetos en Python y se utilizan ampliamente en muchos aspectos del lenguaje y su biblioteca estándar.

¿Qué es un decorador de python?

Un decorador en Python es una función que toma otra función como argumento y extiende o modifica su comportamiento sin modificar su código fuente. Los decoradores proporcionan una forma elegante y flexible de agregar funcionalidades a funciones o métodos existentes.

En Python, los decoradores se utilizan comúnmente para:

- Agregar funcionalidad adicional a una función: Por ejemplo, un decorador podría registrar el tiempo de ejecución de una función, manejar excepciones, realizar validaciones de entrada, o cualquier otra tarea antes o después de que se llame a la función.

- Modificar el comportamiento de una función: Por ejemplo, un decorador podría transformar la salida de una función, cachear resultados para mejorar el rendimiento, o aplicar autenticación o autorización a una función.

- Reutilizar lógica de decoración: Los decoradores permiten encapsular la lógica de decoración en una función separada, lo que facilita la reutilización y la modularidad del código.

Los decoradores se implementan utilizando funciones anidadas y el concepto de funciones de orden superior en Python. Un decorador típicamente toma una función como argumento, define una nueva función que envuelve la función original con el comportamiento adicional deseado, y luego devuelve la nueva función.

Aquí hay un ejemplo básico de un decorador en Python:

```python
def decorador(funcion):
    def wrapper():
        print("Antes de llamar a la función")
        funcion()
        print("Después de llamar a la función")
    return wrapper

@decorador
def funcion_a_decorar():
    print("Función a decorar")

funcion_a_decorar()
```

En este ejemplo, decorador es un decorador que toma una función como argumento y define una nueva función llamada wrapper que imprime un mensaje antes y después de llamar a la función original. Al usar @decorador antes de la definición de funcion_a_decorar, aplicamos el decorador a esa función. Cuando llamamos a funcion_a_decorar(), en realidad estamos llamando a wrapper() que a su vez ejecuta la función original con el comportamiento adicional del decorador.