¿Para qué usamos las clases en Python?

En Python, utilizamos las clases como plantillas o moldes que se utilizan para crear objetos que comparten los mismos métodos o atributos. La sintaxis para definir una clase es utilizando la palabra clave 'class', seguida del nombre de la clase y dos puntos. Por ejemplo:

```
class Casa:
    def numerodecasa(self):
        return 'El número de casa es 001'

mi_num_cas = Casa()
print(mi_num_cas.numerodecasa())
```

En este ejemplo definimos la clase Casa con un método numerodecasa que se encarga de devolver el número de casa. Luego, creamos un objeto de la clase Casa llamado mi_num_cas y llamamos al método numerodecasa() para obtener el número de casa.

En la parte del código que es:

```
mi_num_cas = Casa()
```

Estamos creando una instancia.

En programación orientada a objetos, una "instancia" es un objeto específico creado a partir de una clase. Una clase en sí misma es como un plano o una plantilla que define cómo deben ser los objetos que se van a crear (como se mencionó previamente), ahora, cuando creas un objeto utilizando esa plantilla, estás creando una instancia de esa clase.

Las clases nos permiten organizar y estructurar nuestro código de manera más eficiente, facilitan la reutilización del código y nos permiten modelar entidades del mundo real de manera más natural

Las clases en Python ofrecen diferentes ventajas, encapsuladas en la siguiente lista:

1. **Organización y estructura:** Las clases te permiten organizar y estructurar tu código de manera lógica y ordenada. Puedes agrupar variables y funciones relacionadas en una sola unidad, lo que facilita la comprensión y el mantenimiento del código. Mas fácil de entender por diferentes usuarios.
2. **Reutilización de código:** Una vez que has definido una clase, puedes crear múltiples objetos (instancias) basados en esa clase. Esto te permite reutilizar el código de la clase en diferentes partes de tu programa, evitando la necesidad de escribir el mismo código una y otra vez. Comprime el código.
3. **Abstracción:** Las clases te permiten abstraer los detalles internos de cómo funciona una parte específica de tu programa. Puedes definir una interfaz clara y definir cómo interactuar con objetos de esa clase sin necesidad de conocer todos los detalles de su implementación interna.
4. **Modularidad:** Al definir clases, puedes dividir tu programa en módulos más pequeños y manejables. Cada clase puede representar una unidad funcional independiente, lo que facilita la creación, prueba y mantenimiento del código. Ordenado y de fácil mantención.

5. **Herencia:** La herencia te permite crear nuevas clases basadas en clases existentes. Esto te permite extender y personalizar el comportamiento de una clase padre, evitando la duplicación de código y promoviendo la reutilización. Minimizar el código de nuevo.

En resumen, las clases proporcionan una forma eficiente y organizada de estructurar y escribir código, lo que facilita su comprensión, mantenimiento y reutilización.

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

En Python, el método __init__ es un método especial que se llama automáticamente cuando se crea una nueva instancia de una clase. Este método se conoce como el "constructor" de la clase, ya que se utiliza para inicializar los atributos de un objeto recién creado.

Cuando defines una clase en Python, puedes incluir un método __init__ para especificar cómo se deben inicializar los atributos de la clase cuando se crea una instancia de esa clase. El método __init__ toma como primer parámetro self, que es una referencia al objeto recién creado, y luego puede tomar cualquier número de parámetros adicionales para inicializar los atributos de la clase.

Por ejemplo, supongamos que queremos definir una clase Helado que represente un helado con un vaso, un sabor y un tamaño. Podemos definir la clase de la siguiente manera:

```
class Helado:
    def __init__(self, vaso, sabor, tamano):
        self.vaso = vaso
        self.sabor = sabor
        self.tamano = tamano
```

En este ejemplo, el método __init__ toma tres parámetros: vaso, sabor y tamano. Dentro del método __init__, asignamos estos valores a los atributos self.vaso, self.sabor y self.tamano, respectivamente. Cuando creamos una nueva instancia de la clase Helado, podemos proporcionar valores para estos atributos y el método __init__ se encargará de inicializarlos correctamente.

Un ejemplo de esto sería:

```
mi_helado = Helado(vaso="cono", sabor="chocolate", tamano="grande")
```

En este caso, estamos creando una nueva instancia de la clase Helado y proporcionando valores para los atributos vaso, sabor y tamano. El método __init__ se llama automáticamente con estos valores y se utilizan para inicializar los atributos de la instancia mi_helado.

Cuando creamos una instancia de una clase en Python, podemos acceder a los atributos de esa instancia utilizando la sintaxis de punto. Por ejemplo, si tenemos una instancia de la clase Helado llamada mi_helado, y queremos acceder al atributo sabor, podemos hacerlo de la siguiente manera:

```
print(mi_helado.sabor)
```

Este código imprimirá el valor del atributo sabor de la instancia mi_helado.

Además, podemos crear múltiples instancias de una clase y acceder a sus atributos de manera similar. Por ejemplo, podríamos crear dos instancias diferentes de la clase Helado y acceder a sus atributos de la siguiente manera:

```
helado1 = Helado(vaso="cono", sabor="chocolate", tamano="grande")
helado2 = Helado(vaso="vaso", sabor="fresa", tamano="mediano")

print(helado1.sabor)  # Imprime "chocolate"
print(helado2.sabor)  # Imprime "fresa"
```

Aquí, creamos dos instancias diferentes de la clase Helado, helado1 y helado2, cada una con diferentes valores para los atributos vaso, sabor y tamano. Luego, podemos acceder a los atributos de cada instancia utilizando la sintaxis de punto.

Enlaces de referencia:

https://blog.hubspot.es/website/clases-python

https://www.w3schools.com/python/python_classes.asp

https://www.programacionfacil.org/cursos/python_poo/python_poo_3_el_metodo_init_y_self.html

¿Cuáles son los tres verbos de API?

En el contexto de las API (Interfaz de Programación de Aplicaciones), los verbos se refieren a los métodos HTTP utilizados para interactuar con los recursos de la API. Estos verbos definen las operaciones que se pueden realizar en los datos de la API y determinan cómo se manipulan los recursos en el servidor.

Los verbos más comunes utilizados en las API son GET, POST, PUT y DELETE. Cada uno de estos verbos tiene un propósito específico y se utiliza para realizar diferentes acciones en los recursos de la AP

Ahora profundicemos en los verbos como tal:

1. **GET:** Este verbo se utiliza para recuperar datos de un recurso específico en el servidor. Cuando realizas una solicitud GET a una URL determinada, el servidor responde con los datos asociados a esa URL. Por ejemplo, al realizar una solicitud GET a una API de películas, puedes obtener la lista de películas disponibles.

2. **POST:** El método POST se utiliza para enviar datos al servidor para crear un nuevo recurso. Cuando envías una solicitud POST a una URL específica, estás enviando datos al servidor para que los procese y cree un nuevo recurso. Por ejemplo, al enviar un formulario de registro en un sitio web, los datos del formulario se envían al servidor utilizando el método POST.

3. **PUT:** El verbo PUT se utiliza para actualizar un recurso existente en el servidor. Al enviar una solicitud PUT a una URL específica, estás enviando datos al servidor para que actualice o modifique el recurso asociado con esa URL. Por ejemplo, si deseas actualizar la

información de un usuario en una base de datos, puedes enviar una solicitud PUT con los nuevos datos del usuario.

4. **DELETE:** Como su nombre lo indica, el método DELETE se utiliza para eliminar un recurso existente en el servidor. Al enviar una solicitud DELETE a una URL específica, estás indicando al servidor que elimine el recurso asociado con esa URL. Por ejemplo, al eliminar una publicación en una plataforma de redes sociales, se enviaría una solicitud DELETE para eliminar esa publicación de la base de datos del servidor.

Estos verbos proporcionan un conjunto de operaciones estándar que se utilizan para interactuar con las API y manipular los recursos en un servidor. Dependiendo de la acción que desees realizar (obtener datos, crear, actualizar o eliminar), puedes elegir el verbo HTTP adecuado para enviar la solicitud correspondiente al servidor.

Enlaces de referencia:

https://www.xataka.com/basics/api-que-sirve

https://codigofacilito.com/articulos/rails-verbos-http

¿Es MongoDB una base de datos SQL o NoSQL?

MongoDB es una base de datos NoSQL. A diferencia de las bases de datos relacionales SQL tradicionales, MongoDB utiliza un enfoque no relacional para almacenar y gestionar datos. Esto significa que MongoDB no sigue el modelo de almacenamiento de datos basado en tablas utilizado por las bases de datos SQL, sino que utiliza una estructura de almacenamiento flexible y dinámica basada en documentos.

Sus características principales son:

Formato de almacenamiento: MongoDB almacena los datos en un formato similar a JSON llamado BSON (Binary JSON). Este formato permite una representación eficiente de datos complejos y anidados, lo que lo hace ideal para aplicaciones con esquemas de datos flexibles y dinámicos.

Modelo de datos: En MongoDB, los datos se organizan en colecciones y documentos. Una colección es un conjunto de documentos relacionados, y cada documento es una estructura de datos similar a un JSON que puede contener cualquier cantidad de campos y valores. Esta flexibilidad permite a los desarrolladores modelar datos de manera más natural y adaptarse fácilmente a cambios en los requisitos de la aplicación.

Escalabilidad: MongoDB está diseñado para ser altamente escalable, lo que significa que puede manejar grandes volúmenes de datos y crecer con el tamaño y la complejidad de una aplicación. Utiliza un enfoque de arquitectura distribuida que permite distribuir los datos en múltiples servidores y escalar horizontalmente para satisfacer las demandas de rendimiento de aplicaciones de alto tráfico.

Flexibilidad y velocidad: Al no tener un esquema fijo, MongoDB permite a los desarrolladores almacenar y recuperar datos de forma flexible, sin necesidad de realizar cambios en la estructura de la base de datos. Esto puede mejorar la velocidad de desarrollo y la agilidad del equipo, ya que pueden adaptarse rápidamente a los cambios en los requisitos de la aplicación sin tener que preocuparse por migraciones de esquema complejas.

MongoDB se utiliza en una amplia variedad de aplicaciones y casos de uso, incluidos:

- **Aplicaciones web y móviles**: MongoDB es una opción popular para el desarrollo de aplicaciones web y móviles debido a su capacidad para manejar datos no estructurados y su escalabilidad horizontal.
- **Análisis de datos en tiempo real:** MongoDB se utiliza en aplicaciones que requieren procesamiento de datos en tiempo real, como análisis de registros, monitoreo de aplicaciones y detección de fraudes.
- **Internet de las cosas (IoT):** Debido a su capacidad para manejar grandes volúmenes de datos y su escalabilidad, MongoDB es una opción común para aplicaciones de IoT que recopilan y procesan datos de dispositivos conectados.

MongoDB se promociona a si mismo de la siguiente manera:

Y tiene una pagina muy completa que atrae a su uso particular.

Enlaces de referencia::

https://www.mongodb.com/es/solutions/use-cases

https://datascientest.com/es/mongodb-todo-sobre-la-base-de-datos-nosql-orientada-a-documentos#:~:text=ingenier%C3%ADa%20de%20datos.,MongoDB%20es%20una%20base%20de%20datos%20NoSQL%20orientada%20a%20documentos,almacenan%20como%20colecciones%20y%20documentos

¿Qué es una API?

Es una pieza de código que permite que diferentes aplicaciones se comuniquen entre sí, compartir información y funcionalidades. Es un intermediario entre dos sistemas, que permite que una aplicación se comunique con otra y pida datos o acciones específicas.

Por ejemplo, si se tiene una app para móviles acerca de cocteles y se hace una búsqueda de un coctel en específico, la API ayudara a que esta app se comunique con el sitio web de recetas de cocteles y pida las recetas que cumplen con los criterios de búsqueda. La API entonces se encarga de recibir la solicitud, buscar las recetas apropiadas y regresar los resultados a la aplicación.

En resumidas palabras, es lo que permite comunicar diferentes sistemas, oara obtener la información que se necesita, ese compartir de información hace que todo sea mas eficiente minimizando espacios tiempos y trabajo.

Ejemplos de API:

- **Google Maps:** gracias a los estándares aplicados por Google, la mayoría de los sitios web pueden usar las APIs de Google Maps para integrar mapas.

- **Vulkan:** esta API multiplataforma permite que los desarrolladores creen interfaces gráficas en tiempo real y de alta calidad en aplicaciones, brindando mayor rapidez y eficiencia en la comunicación entre apps y unidades de procesamiento gráfico.

- **Skyscanner:** esta plataforma de metabúsqueda facilita que viajeros puedan encontrar mejores tarifas para sus vuelos. Además, proporciona una API para aliados comerciales compatible con XML y JSON para el intercambio de datos.

Para más información pueden acceder a :

https://www.sydle.com/es/blog/api-6214f68876950e47761c40e7

https://www.mulesoft.com/es/resources/api/what-is-an-api

https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

¿Qué es Postman?

Postman es una herramienta muy útil para los desarrolladores de software. Piensa en ella como una especie de caja de herramientas para trabajar con APIs, que son como conjuntos de reglas que permiten que diferentes aplicaciones se comuniquen entre sí.

Con Postman, puedes enviar solicitudes a estas APIs y ver las respuestas de una manera muy clara y organizada. Es como si estuvieras hablando con otra aplicación y recibiendo información de vuelta, pero todo en un entorno digital.

Lo genial de Postman es que tiene una interfaz fácil de entender, por lo que no necesitas ser un experto en programación para usarlo. Además, puedes guardar tus solicitudes y respuestas para usarlas más tarde o compartirlas con otros desarrolladores.

También puedes automatizar algunas tareas con Postman, lo que significa que puedes hacer que realice ciertas acciones automáticamente, lo que ahorra mucho tiempo y esfuerzo. En resumen, Postman es una herramienta muy práctica y útil para trabajar con APIs en el desarrollo de software. Para mayor información, visitar:

https://www.postman.com/

POSTMAN

¿Qué es el polimorfismo?

El polimorfismo es un concepto de la programación orientada a objetos que nos permite usar objetos de diferentes clases de manera intercambiable, incluso si pertenecen a una jerarquía de herencia. Esto significa que podemos tratar objetos de distintas clases de manera uniforme, siempre y cuando compartan una interfaz común.

Imaginemos un escenario donde tenemos varias clases de animales, cada una con su propio método para hacer un sonido:

```python
class Animal:
    def sonido(self):
        raise NotImplementedError

...
class Gato(Animal):
    def sonido(self):
        print("Mew mew")

...
class Perro(Animal):
    def sonido(self):
        print("Woof woof")
```

En este ejemplo, tenemos una clase base llamada Animal que define un método sonido(), pero no proporciona una implementación concreta. Esto es porque cada animal puede tener su propio sonido, y no queremos definir un comportamiento predeterminado en la clase base.

Luego, tenemos dos clases derivadas: Gato y Perro. Ambas clases heredan de la clase Animal y proporcionan su propia implementación del método sonido(). El gato hace "Mew mew" y el perro hace "Woof woof".

Ahora, gracias al polimorfismo, podemos tratar objetos de las clases Gato y Perro de la misma manera, ya que ambos tienen un método sonido() definido. Por ejemplo:

```python
gato = Gato()
perro = Perro()

gato.sonido()  # Output: Mew mew
perro.sonido()  # Output: Woof woof
```

Aunque ambos animales tienen comportamientos diferentes, podemos llamar al mismo método sonido() en cada uno de ellos y obtener el resultado específico de cada clase. Esto es posible gracias al polimorfismo, que nos permite tratar diferentes tipos de objetos de manera uniforme mediante una interfaz común.

¿Qué es un método dunder?

En Python, los métodos dunder son como magia detrás de escena que permite a las clases comportarse de manera especial en ciertas situaciones. Por ejemplo, imagina que estás creando una clase llamada Pelicula para representar películas en una aplicación de gestión de películas. Quieres que tus objetos de película puedan interactuar de forma intuitiva con otras partes de tu código, como imprimir información sobre la película, obtener su duración, etc.

Aquí hay un ejemplo simple de cómo podrías definir la clase Película con algunos métodos dunder:

```python
class Pelicula:
    def __init__(self, titulo, duracion):
        self.titulo = titulo
        self.duracion = duracion

    def __str__(self):
        return f"{self.titulo} - Duración: {self.duracion} minutos"

    def __len__(self):
        return self.duracion

# Crear una instancia de la clase Pelicula
mi_pelicula = Pelicula("El Padrino", 175)

# Imprimir la película usando el método dunder __str__
print(mi_pelicula)  # Salida: El Padrino - Duración: 175 minutos

# Obtener la duración de la película usando el método dunder __len__
print(len(mi_pelicula))  # Salida: 175
```

En este ejemplo, hemos definido la clase Película con dos métodos dunder: __init__, que se llama automáticamente cuando creamos una nueva instancia de la clase para inicializar sus atributos, y __str__ y __len__, que nos permiten obtener una representación de cadena legible de la película y su duración respectivamente.

Cuando creamos una instancia de la clase Pelicula y la imprimimos (print(mi_pelicula)), el método __str__ se invoca automáticamente para proporcionar una representación de cadena legible de la película.

Del mismo modo, cuando usamos la función len() en una instancia de la clase Pelicula (len(mi_pelicula)), el método __len__ se invoca automáticamente para devolver la duración de la película.

Los métodos dunder son un tipo especial de métodos en Python. Su nombre dunder, viene del inglés double underscore methods, traducido al español métodos de doble guión bajo. Como lo describe su nombre la forma de declarar dichos métodos es con doble guion bajo _ _ al principio y al final de su nombre.

Existen muchos métodos dunder predefinidos que tienen un significado especial en Python y son invocados automáticamente en ciertas situaciones específicas, por ejemplo: cuando hacemos una suma de 4 + 4, se está invocando al método **__add__** internamente y sin que nosotros lo visualicemos ni o llamemos de forma específica.

Pero la peculiaridad de estas funciones es que aparte de ejecutarse automáticamente, los podemos llamar cuando creamos oportuno dentro de nuestro código.

En resumen, los métodos dunder son como pequeños trucos mágicos que nos permiten personalizar el comportamiento de nuestras clases y hacer que interactúen de forma intuitiva con otros aspectos de nuestro código.

Para mayor información visitar:

https://geekflare.com/es/magic-methods-in-python

https://pythonintermedio.readthedocs.io/es/latest/classes.html#:~:text=Las%20clases%20en%20Python%20son,final%20del%20nombre%20del%20mismo.

https://geekflare.com/es/magic-methods-in-python/

¿Qué es un decorador de python?

Los decoradores en Python son como capas adicionales de funcionalidad que puedes agregar a tus funciones sin tener que modificarlas directamente. Piensa en ellos como aderezos que puedes poner en tus comidas para darles un toque extra de sabor, sin alterar la receta principal.

Imagina que estás en una fiesta y tienes un pastel básico. Quieres decorarlo con diferentes coberturas: una capa de crema, un glaseado de chocolate o una lluvia de confeti. En lugar de cambiar el pastel cada vez que quieras una decoración diferente, simplemente agregas la cobertura que prefieras encima.

Ahora, volviendo a Python, puedes tener una función básica que hace algo simple, como saludar a alguien. Pero luego quieres agregarle funcionalidades adicionales, como registrar cada saludo en un archivo de registro o medir el tiempo que tarda en ejecutarse. Los decoradores te permiten hacer eso sin modificar la función original.

Imagina que tienes una función que imprime un mensaje de saludo:

```python
def saludar(nombre):
    return f"Hola, {nombre}!"
```

Ahora, quieres agregar una funcionalidad adicional a esta función para que también registre la fecha y hora en la que se realizó el saludo. En lugar de modificar la función original, puedes usar un decorador para lograrlo:

```python
import datetime

def registrar_tiempo(funcion):
    def envoltura(nombre):
        mensaje = funcion(nombre)
        tiempo_actual = datetime.datetime.now()
        registro = f"Saludo registrado a las {tiempo_actual}: {mensaje}"
        return registro
    return envoltura

@registrar_tiempo
def saludar(nombre):
    return f"Hola, {nombre}!"

print(saludar("Juan"))
```

Este código define una función llamada saludar(nombre) que devuelve un mensaje de saludo con el nombre proporcionado. Luego, se define un decorador llamado registrar_tiempo que envuelve la función saludar y agrega la fecha y hora actual al mensaje de saludo. Finalmente, el decorador se aplica a la función saludar usando la sintaxis @registrar_tiempo, lo que significa que todas las llamadas a saludar serán gestionadas por el decorador.