UTILIDAD DE LAS CLASES EN PYTHON

Una clase en Python es una estructura de programación que permite definir un conjunto de métodos y atributos que describen un objeto o entidad. Las clases son un concepto fundamental en la programación orientada a objetos, que se utilizan para modelar entidades del mundo real o abstracto en un programa de computadora.

Una clase define una plantilla o molde para crear objetos, los cuales son instancias de esa clase. Los objetos creados a partir de una clase tienen las mismas propiedades y comportamientos definidos por la clase, pero pueden tener valores diferentes para los atributos que se definen en la clase.

Ventajas del uso de las clases en Python:

- Reutilización de código: las clases pueden reutilizarse en diferentes partes del programa o en distintos programas, lo que ahorra tiempo y reduce la duplicación de código.
- Encapsulación: permiten ocultar la complejidad de un objeto y exponer solo una interfaz simple y fácil de usar para interactuar con él.
- Modularidad: pueden descomponer un programa en componentes más pequeños y manejables, lo que facilita el mantenimiento y la solución de problemas.
- Polimorfismo: 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.

Desventajas del uso de las clases en Python:

- Sobrecarga de complejidad: las clases pueden agregar complejidad adicional a un programa y hacer que sea más difícil de entender y depurar.
- Curva de aprendizaje: 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.
- Uso innecesario: a veces, las clases se utilizan innecesariamente en situaciones en las que una función simple podría haber hecho el trabajo de manera más eficiente.

En Python, una clase se define mediante la palabra clave «class», seguida del nombre de la clase y dos puntos (:) y luego el cuerpo de la clase. El cuerpo de la clase contiene definiciones de métodos y atributos, que pueden ser públicos o privados según su acceso.

```
1   class Invoice:
2
3       def greeting(self):
4           return 'Hi there'
5
```

¿QUÉ MÉTODO SE EJECUTA AUTOMÁTICAMENTE, CUANDO SE CREA UNA INSTANCIA DE UNA CLASE?

Cuando se crea una instancia de una clase, el método __init__ es llamado automáticamente por el intérprete de Python y se utiliza para realizar cualquier inicialización que sea necesaria para la instancia. El método __init__ se usa para asignar valores iniciales a los atributos de una instancia de la clase.

```python
class Invoice:
  def __init__(self, client, total):
    self.client = client
    self.total = total

  def formatter(self):
    return f'{self.client} owes: ${self.total}'


google = Invoice('Google', 100)
snapchat = Invoice('SnapChat', 200)
```

Los atributos son las variables que pertenecen a una instancia particular de la clase. Al llamar al método __init__, podemos establecer los valores de estos atributos y configurar la instancia de la clase para su uso posterior.

¿CUÁLES SON LOS TRES VERBOS DE API?

Los 3 más comunes son los siguientes:

- GET: es el más simple de todos, es el que usamos para obtener un recurso. Las peticiones GET no deben causar efectos secundarios en un servidor, no deben producir nuevos registros, ni modificar los ya existentes. A esta cualidad la llamamos idempotencia, cuando una acción ejecutada un número indefinido de veces, produce siempre el mismo resultado.Esto quiere decir, que no importa cuántas veces hagamos una petición GET, los resultados obtenidos serán los mismos.

```python
import requests

# The API endpoint
url = "https://jsonplaceholder.typicode.com/posts/1"

# A GET request to the API
response = requests.get(url)
```

- POST: para crear recursos nuevos, no para eliminarlos, ni para modificarlos. Cada llamada con POST debería producir un nuevo recurso. Lo que es interesante acerca de POST no es el verbo en sí, queda muy claro que es para crear, más bien es los recursos a los que se dirige.

```
# Define new data to create
new_data = {
    "userID": 1,
    "id": 1,
    "title": "Making a POST request",
    "body": "This is the data we created."
}

# The API endpoint to communicate with
url_post = "https://jsonplaceholder.typicode.com/posts"

# A POST request to tthe API
post_response = requests.post(url_post, json=new_data)
```

- DELETE: eliminación de un recurso. En muchas ocasiones es un “soft delete”, es decir, no se elimina definitivamente un recurso, sino que únicamente es marcado como eliminado o desactivado.

```
import requests

URL = "https://jsonplaceholder.typicode.com/posts/1"
response = requests.delete(URL)
```

Cabe decir que también existe el verbo PUT, el cual se utiliza para reemplazar la información de un recurso. Para actualizar la descripción de un “grid”, se utilizaría este verbo. Esta llamada es idempotente.

¿ES MONGODB UNA BASE DE DATOS SQL O NOSQL?

MongoDB es una base de datos no relacional de código abierto que almacena datos en documentos similares a JSON. A diferencia de una base de datos relacional tradicional que almacena datos en filas y columnas, MongoDB almacena datos en colecciones. Cada colección tiene documentos, y dentro de esos documentos hay campos. No necesita definir el esquema mientras escribe datos en MongoDB, lo que lo hace ideal

para almacenar grandes cantidades de datos no estructurados. Además, le permite agregar nuevos campos sobre la marcha.

Una de las características que distingue a MongoDB de otras bases de datos es la escalabilidad horizontal, que divide la base de datos en partes. Para agregar más capacidad, puede agregar un servidor sobre la marcha sin obstaculizar el rendimiento de la base de datos o experimentar tiempo de inactividad.

En poco tiempo, MongoDB se ha convertido en una de las bases de datos NoSQL favoritas por los desarrolladores. Se trata de un sistema creado por Apache y escrito en lenguaje Erlang que funciona en la mayoría de sistemas POSIX, incluyendo GNU/LINUX y OSX, pero no así en sistemas Windows.

Otras bases de datos NoSQL serían neo4j y DynamoDB.

Diferencia entre MongoDB y SQL: MongoDB es una base de datos de código abierto, mientras que SQL Server tiene licencia para fines comerciales. Sin embargo, solo necesita una licencia para ejecutar varias instancias en SQL Server.

¿QUÉ ES UNA API?

API significa “interfaz de programación de aplicaciones” (Application Programming Interfaces). En el contexto de las API, la palabra aplicación se refiere a cualquier software con una función distinta. La interfaz puede considerarse como un contrato de servicio entre dos aplicaciones. Este contrato define cómo se comunican entre sí mediante solicitudes y respuestas. La documentación de su API contiene información sobre cómo los desarrolladores deben estructurar esas solicitudes y respuestas.

Las API son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos. Por ejemplo, el sistema de software del instituto de meteorología contiene datos meteorológicos diarios. La aplicación meteorológica de su teléfono “habla” con este sistema a través de las API y le muestra las actualizaciones meteorológicas diarias en su teléfono.

Las API se clasifican tanto en función de su arquitectura como de su ámbito de uso. En base a su arquitectura tendríamos las siguientes:

- SOAP: estas API utilizan el protocolo simple de acceso a objetos. El cliente y el servidor intercambian mensajes mediante XML. Se trata de una API menos flexible que era más popular en el pasado.
- RPC: estas API se denominan llamadas a procedimientos remotos. El cliente completa una función (o procedimiento) en el servidor, y el servidor devuelve el resultado al cliente.

- WebSocket: la API de WebSocket es otro desarrollo moderno de la API web que utiliza objetos JSON para transmitir datos. La API de WebSocket admite la comunicación bidireccional entre las aplicaciones cliente y el servidor. El servidor puede enviar mensajes de devolución de llamada a los clientes conectados, por lo que es más eficiente que la API de REST.
- REST: estas son las API más populares y flexibles que se encuentran en la web actualmente. El cliente envía las solicitudes al servidor como datos. El servidor utiliza esta entrada del cliente para iniciar funciones internas y devuelve los datos de salida al cliente.

Existen también cuatro tipos de API en base al ámbito de uso:

- Privadas: estas son internas de una empresa y solo se utilizan para conectar sistemas y datos dentro de la empresa.
- Públicas: Están abiertas al público y pueden cualquier persona puede utilizarlas. Puede haber o no alguna autorización y coste asociado a este tipo de API.
- De socios: Solo pueden acceder a ellas los desarrolladores externos autorizados para ayudar a las asociaciones entre empresas.
- Compuestas: Estas combinan dos o más API diferentes para abordar requisitos o comportamientos complejos del sistema.

¿QUÉ ES POSTMAN?

Postman en sus inicios nace como una extensión que podía ser utilizada en el navegador Chrome de Google y básicamente nos permite realizar peticiones de una manera simple para testear APIs de tipo REST propias o de terceros.

Gracias a los avances tecnológicos, Postman ha evolucionado y ha pasado de ser de una extensión a una aplicación que dispone de herramientas nativas para diversos sistemas operativos como lo son Windows, Mac y Linux.

Cuenta con una versión libre de pago y con tres planes (básico, profesional y empresarial), si deseas consultar el detalle entre cada plan y sus precios puedes verlo en su web oficial (https://www.postman.com/).

Postman sirve para múltiples tareas dentro de las cuales destacaremos en esta oportunidad las siguientes:

- Testear colecciones o catálogos de APIs tanto para Frontend como para Backend.
- Organizar en carpetas, funcionalidades y módulos los servicios web.

- Permite gestionar el ciclo de vida (conceptualización y definición, desarrollo, monitoreo y mantenimiento) de nuestra **API**.
- Generar documentación de nuestras APIs.
- Trabajar con entornos (calidad, desarrollo, producción) y de este modo es posible compartir a través de un entorno **cloud** la información con el resto del equipo involucrado en el desarrollo.

Si deseas iniciar con esta herramienta lo primero que debes hacer es instalarla y para ello deberás descargar el software según el sistema operativo que tengas (Linux, Mac o Windows) en la web oficial de Postman, así mismo puedes elegir descargar la aplicación o probar con la versión web.

¿QUÉ ES EL POLIMORFISMO?

El polimorfismo en POO (Programación Orientada a Objetos) es la capacidad que tienen ciertos lenguajes para hacer que, al enviar el mismo mensaje (o, en otras palabras, invocar al mismo método) desde distintos objetos, cada uno de esos objetos pueda responder a ese mensaje (o a esa invocación) de forma distinta.

La herencia nos permite construir nuevas clases basadas en clases existentes, facilitando la reutilización y extensión del código. Por otro lado, el polimorfismo otorga a los objetos la capacidad de comportarse de diversas maneras según el contexto, aportando flexibilidad y simplicidad al código.

¿QUÉ ES UN MÉTODO DUNDER?

Las clases en Python son famosas por sus métodos mágicos, comúnmente referidos con dunder que viene del inglés y significa “double underscore”. Es decir, son métodos definidos con doble barra baja, tanto al principio con al final del nombre del mismo.

Como ejemplo de método dunder tendríamos el previamente explicado __init__, y también estarían __str__ (este método dunder se utiliza para devolver una representación de cadena/string de una instancia de una clase) y __repr__ (este método dunder se utiliza para devolver una representación de cadena legible de un objeto), entre otros.

```python
class InfiniteLineup:
    def __init__(self, players):
        self.players = players

    def lineup(self):
        lineup_max = len(self.players)
        idx = 0

        while True:
            if idx < lineup_max:
                yield self.players[idx]
            else:
                idx = 0
                yield self.players[idx]

            idx += 1

    def __repr__(self):
        return f'InfiniteLineup({self.players})'

    def __str__(self):
        return f"InfiniteLineup with the players: {', '.join(self.players)}"
```

¿QUÉ ES UN DECORADOR DE PYTHON?

Los decoradores son una funcionalidad relativamente importante en Python. Se podría decir que son funciones que modifican la funcionalidad de otras funciones, y ayudan a hacer nuestro código más corto y Pytónico o Pythonic.

un decorador no es más que una función la cual toma como input una función y asu vez retorna otra función. Puede sonar algo confuso ¿no? lo que nos debe quedar claro es que al momento de implementar un decorador estaremos trabajando, con por lo menos, 3 funciones. El input, el output y la función principal. Para que nos quede más en claro nombremo a las funciones como: a, b y c.

Donde a recibe como parámetro b para dar como salida a c. Es decir, a(b) -> c.

```python
def funcion_a(funcion_b):
    def funcion_c():
        print('Antes de la ejecución de la función a decorar')
        funcion_b()
        print('Después de la ejecución de la función a decorar')

    return funcion_c
```