Documentación para la asignación M3C6

¿Para qué usamos Clases en Python?

Las clases en Python son como los planos de los objetos. Pueden contener datos, funciones y comportamientos. Como la clase es como un plano, por ella misma no hace nada. Un objeto debe ser creado. El objeto será la copia de la clase con valores. A esta acción se le llama instanciación.

Sintaxis:

```
class Nombre:
    def algo(self): #cualquier cosa que vaya dentro necesita tener un argumento.
                    El argumento por defecto que siempre debe estar es "self".
        return

variable = Invoice()#instanciación de la clase para que devuelva algo.

print(variable.algo())
```

Todas las instancias comparten los atributos y el comportamiento de la clase, pero el valor de esos atributos (es decir, el estado) es único para cada objeto. Una misma clase puede tener cualquier número de instancias.

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

El método que se ejecuta automáticamente al crear una instancia de una clase es el método __init__. Asigna valores iniciales a los atributos de una instancia de la clase. Se utiliza para realizar cualquier inicialización que sea necesaria para la instancia.

Es un método específico de las clases de Python y es una función constructora. Se puede llamar de forma manual en las clases, a continuación se muestra un ejemplo de ello en el que se devuelve la cadena con los valores de la edad y nombre introducidos al instanciar la clase.

Ejemplo con el método __init__ escrito:

```
Class Conductor:

    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

def saludo(self):
    print(f"Buenos días, mi nombre es {self. nombre}
            y tengo {self. Edad} años.")
```

#Crear objeto de la clase:
```
conductor1 = Conductor("Marcos", 43)
```

#Llamar al método
```
conductor1.saludo()
```
#Esto devolverá la cadena con los datos del conductor 1.

¿Cuáles son los tres verbos de API?

Primero de todo, aclarar qué son los verbos de API. Son métodos de http que indican las acciones que un cliente API quiere hacer con unos recursos dados. Cada uno está enfocado a una operación concreta, entre los que se incluyen los siguientes:

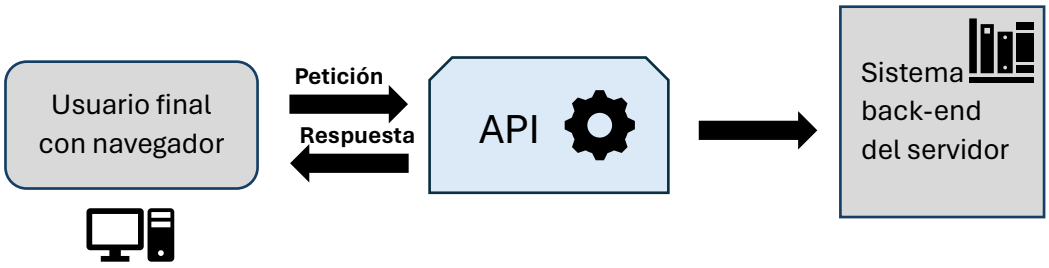
- GET: este método se usa para recuperar datos en un servidor. Mediante él se accede a los recursos de determinado punto. Por ejemplo, una petición GET a unos productos de supermercado online devolvería todos los productos de esa base de datos.
- POST: se utiliza para crear nuevos recursos. Siguiendo con el ejemplo anterior, si quisiese crear un producto de supermercado nuevo en la base de datos, mandaría una petición POST. En este caso, a diferencia de la de GET, se suele especificar algo en el cuerpo de solicitud mediante código.
- PUT: se emplea para sustituir un recurso ya existente por una versión actualizada. Por ejemplo, si se quisiese cambiar el nuevo producto generado con POST, o su precio, su versión, lo sustituye. Para ello, se incluyen los datos en la solicitud y esta se cambia en el punto final.
- DELETE: este método se emplea para eliminar un recurso. Cuando se manda esta petición se está pidiendo que se borra una URL específica. Algunas APIs tienen restricciones y autorizaciones concretas para que este método solo pueda usarse por clientes con permisos específicos.

¿Es MongoDB una base de datos SQL o NoSQL?

MongoDB es una base de datos de documentos NoSQL, por lo que es escalable, flexible y tiene un modelo de consultas e indexaciones avanzado que ofrece una gran escalabilidad y flexibilidad, y un modelo de consultas e indexación avanzado. almacena datos en documentos flexibles similares a JSON, por lo que los campos pueden variar entre documentos y la estructura de datos puede cambiarse con el tiempo, además es fácil de aprender por lo que los desarrolladores tienen todas las funcionalidades que necesitan sin tener que hacer acciones más complejas. Por último, otra característica de que sea NoSQL es que su escalada es horizontal, lo que significa que puede añadir servidores básicos más baratos siempre que se necesite.

¿Qué es una API?

API son las siglas de *Application Programming Interface* en inglés. En español se traduciría como interfaz de programación de aplicaciones, que es una colección de protocolos de comunicación y subrutinas utilizadas por varios programas para comunicarse entre ellos. Una API ayuda a dos programas o aplicaciones a comunicarse entre sí proporcionándoles las herramientas y funciones necesarias. Toma la solicitud del usuario y la envía al proveedor de servicios y, a continuación, envía el resultado generado por el proveedor de servicios al usuario deseado.

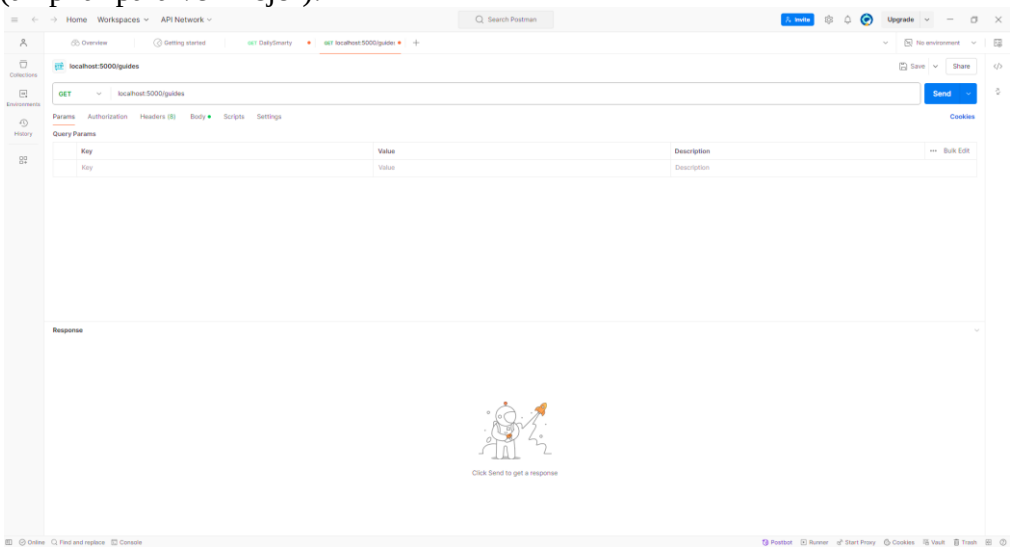


Un desarrollador utiliza ampliamente las API en su software para implementar diversas funciones mediante el uso de una llamada a la API sin escribir códigos complejos para la misma.

¿Qué es Postman?

Postman es una plataforma API para crear, modificar, probar y usar APIs. Casi cualquier funcionalidad que pueda necesitar un desarrollador está encapsulada en esta herramienta. Una de sus funcionalidades más importantes es que si construyes una API para otra persona, necesitas que se comunique con tu máquina local, y esto lo puedes hacer con Postman. Tiene la capacidad de hacer varios tipos de peticiones HTTP (GET, POST, PUT, PATCH), guardar entornos para su uso posterior, convertir la API a código para varios lenguajes (como JavaScript y Python).

Tiene una interfaz user friendly, es decir, fácil de manejar , como lo muestra la siguiente foto (ampliar para ver mejor):



¿Qué es el polimorfismo?

Como su nombre indica, el polimorfismo significa que un elemento puede tomar diferentes formas. En las clases de Python suele ir ligado al concepto de herencia. En ese caso, permite definir métodos en la clase hija con el mismo nombre que en la clase madre.

A continuación, se enseña un ejemplo de su uso en clases:

```
class Html:

    def __init__(self, content):
        self.content = content

    def render(self):
        raise NotImplementedError('Subclass must implement
                                   render method')


class Heading(Html):

    def render(self):
        return f'<h1>{self.content}</h1>'



tags = [
        Div('Some content'),
        Heading('Here a heading')
       ]

for tag in tags:
    print(tag.render()) #Esto devolverá
                                   <div>Some content</div>
                                   <h1>Here a heading</h1>
```

Como se observa, a pesar de ser render() el mismo método, en diferentes clases (Html y Heading) toma diferente forma y trabaja con diferente tipo de contenido.

¿Qué es un método dunder?

Son los métodos que empiezan y terminan con dobles guiones bajos '_ _'. Se definen mediante clases incorporadas en Python y se utilizan habitualmente para la sobrecarga de operadores. Estos métodos están directamente relacionados con cómo funciona Python con los términos privado y protegido en las clases y es específico del lenguaje Python.

Ejemplo de dos métodos dunder:

- __str__(): devuelve el input que se le haya incluido de forma leíble e informal para humanos. Es decir, la representación en forma de cadena de un objeto. Se

utiliza a menudo con fines de depuración. Cuando se necesita saber si el código que se está creando funciona o no, se incluye este método para avisar en caso de que no sea así.

- __repr__(): devuelve datos sin procesar, que no suele tener un buen formato. Más bien devuelve objetos. Se utiliza para registros o registros de errores.

¿Qué es un decorador de Python?

Los decoradores son una herramienta de Python que permite a los programadores modificar el comportamiento de una función o clase. Los decoradores nos permiten envolver otra función para extender el comportamiento de la función envuelta, sin modificarla permanentemente. Se definen como un método, pero se pueden usar como atributo. Se ponen delante de la función con un '@' delante.

Ejemplo: en este ejemplo se empleará el decorador @property. Al definir la propiedad con @property el acceso a ese atributo se realiza a través de una función, siendo por lo tanto un acceso controlado.

```
class Invoice:

    def __init__(self, client, total):
        self._client = client
        self._total = total

    def formatter(self):
        return f'{self._client} owes: ${self._total}'

    @property
    def client(self):
        return self._client

    @property
    def total(self):
        return self._total

google = Invoice('Google', 100)

print(google.client)

print(google.total)
```