Checkpoint 6

1.-¿Para qué usamos Clases en Python?

En Python, una clase es una plantilla o un plano para crear objetos. Una clase define las propiedades y los comportamientos que tendrán los objetos que se creen a partir de ella. Las propiedades se llaman **atributos** y los comportamientos se llaman **métodos**.
Los **atributos** son **variables asociadas a la clase** y los **métodos** son **funciones asociadas a la clase**.

Para declarar una clase en Python, se utiliza la palabra reservada «**class**» seguida del nombre de dicha clase. Un ejemplo básico de declaración de una clase en Python sería la siguiente:

```python
#Definimos clase
class Persona:
    def __init__(self,nombre,edad):
        self.nombre = nombre
        self.edad = edad
    def saludar(self):
        print(f"Hola, mi nombre es {self.nombre} y tengo {self.edad} años")

#Creamos objetos de la clase "Persona"
persona1 = Persona("Juan",25)
persona2 = Persona("María",30)
#LLamamos al Método "saludar" de ambos objetos
persona1.saludar()
persona2.saludar()
```

En este ejemplo, hemos creado una clase llamada «**Persona**» con dos atributos («**nombre**» y «**edad**«) y un método llamado «**saludar**» que imprime un saludo personalizado. Luego creamos dos objetos de la clase Persona, «**persona1**» y «**persona2**«, y llamamos al método «**saludar**» en cada objeto.

Output:
Hola, mi nombre es Juan y tengo 25 años
Hola, mi nombre es María y tengo 30 años

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

Cuando queremos desarrollar un programa con esta metodología orientada a objetos, hay que comenzar por definir la *estructura* de los objetos. cada una de estas estructuras es a lo que llamamos **clase**. Y una vez que hemos definido una clase (la clase "persona", por ejemplo) ya podemos comenzar a crear objetos de esa clase (el objeto "Alicia Arjona" o el objeto "David Martín"). La clase no es más que la descripción "teórica" de los objetos, incluyendo información sobre sus atributos y métodos. Cuando queremos crear un objeto de una clase lo que hacemos es crear una **instancia** de dicha clase.

Una **instancia de clase** es un objeto que se crea a partir de una clase. Es decir, una clase actúa como un "molde" o plantilla, y cada vez que se utiliza para crear un objeto, ese objeto se denomina **instancia**. Cada instancia puede tener sus propios valores para los atributos definidos en la clase, pero comparte la estructura y el comportamiento de la clase.

Ejemplo:

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

# Crear instancias de la clase Usuario
usuario1 = Usuario("Ana", 25)
usuario2 = Usuario("Carlos", 30)
```

Si tienes una clase Usuario, puedes crear varias instancias de esta clase, cada una representando un usuario diferente. En este caso usuario1 y usuario2 son **instancias** de la clase Usuario, pero tienen valores diferentes para sus atributos.

El **método** que se ejecuta automáticamente cuando se crea una instancia de una clase en Python es el método __init__(). Este es un **método especial** (llamado **constructor**) que se utiliza para inicializar los atributos de la clase cuando se crea una instancia.Ejemplo:

```python
class Usuario:
    def __init__(self, nombre, edad):
        # Se ejecuta automáticamente al crear una instancia
        self.nombre = nombre
        self.edad = edad

# Al crear una instancia, __init__ se ejecuta y asigna valores a los atributos
usuario1 = Usuario("Ana", 25)  # Aquí se llama automáticamente al método __init__
```

El método __init__ recibe los parámetros al crear una instancia y se encarga de inicializar los atributos del objeto.

3.-¿Cuáles son los tres verbos de API?

En programación, el término API (abreviatura de Application Programming Interface) se refiere a una parte de un programa informático diseñada para ser utilizada o manipulada por otro programa, a diferencia de las interfaces, que están diseñadas para ser utilizadas o manipuladas por humanos.

Los programas informáticos a menudo necesitan comunicarse entre sí o con el sistema operativo subyacente, y las API son una forma de hacerlo.

Los **tres verbos principales de una API** que se usan en el contexto de las API son los siguientes:

1. **GET**: Se utiliza para **recuperar** información desde el servidor. Es un verbo de lectura, por lo que no modifica ningún dato en el servidor. Por ejemplo, al solicitar una lista de usuarios o los detalles de un solo recurso.
    - Ejemplo: Obtener un usuario de la API.
      GET /users/1

2. **POST**: Se usa para **enviar datos al servidor**, generalmente con el fin de **crear** un nuevo recurso. Los datos enviados con una solicitud POST son procesados por el servidor y pueden llevar a la creación de un nuevo objeto.
    - Ejemplo: Crear un nuevo usuario en la API. POST /users
3. **PUT**: Se emplea para **actualizar** o **reemplazar** un recurso existente en el servidor. Envías datos actualizados en la solicitud para modificar un recurso específico.
    - Ejemplo: Actualizar los detalles de un usuario en la API. PUT /users/1

Aunque estos tres son los más comunes, las APIs RESTful también utilizan otros verbos como **DELETE** (para eliminar un recurso) y **PATCH** (para actualizaciones parciales).

4.-¿Es MongoDB una base de datos SQL o NoSQL?

MongoDB es un sistema de gestión de bases de datos (DBMS, por sus siglas en inglés) no relacionales y de código abierto, que utiliza documentos flexibles en lugar de tablas y filas para procesar y almacenar varias formas de datos. Es una solución de base de datos **NoSQ**L, es decir, MongoDB no requiere un sistema de gestión de bases de datos relacionales (RDBMS), por lo que proporciona un modelo de almacenamiento de datos elástico que permite a los usuarios almacenar y consultar fácilmente tipos de datos multivariados.

Características clave que lo hacen NoSQL:

1. **Modelo basado en documentos**: Los datos en MongoDB se almacenan en documentos JSON, lo que permite almacenar estructuras de datos más complejas que las tablas tradicionales de SQL.
2. **No usa esquemas rígidos**: A diferencia de las bases de datos SQL, en las que los esquemas son estrictos (requieren definir tablas, columnas y sus tipos), MongoDB permite que cada documento pueda tener un esquema diferente, lo que otorga mayor flexibilidad.
3. **Escalabilidad horizontal**: MongoDB está diseñado para distribuir los datos a través de varios servidores (sharding), lo que facilita su escalabilidad.
4. **Consultas flexibles**: Aunque no usa SQL, MongoDB permite hacer consultas avanzadas a través de su propio lenguaje de consulta.

MongoDB es una opción popular cuando se necesitan sistemas con gran flexibilidad de datos y escalabilidad masiva.

5.-¿Qué es una API?

API o *Application Programming Interface*, que en español quiere decir Interfaz de Programación de Aplicaciones, **es un conjunto de funciones y procedimientos que permite integrar sistemas**, permitiendo que sus funcionalidades puedan ser reutilizadas por otras aplicaciones o software.

Una API sirve para intercambiar datos entre diferentes tipos de software y así automatizar procedimientos y desarrollar nuevas funcionalidades.

5.1.- ¿Cómo funciona?

Una **API es una especie de puente que conecta diversos tipos de software o aplicaciones y puede crearse en varios lenguajes de programación**. Además de un buen desarrollo, una API debe tener una documentación clara y objetiva para poder facilitar su implementación.

Asimismo, suele utilizarse un formato predefinido de datos para compartir información entre los sistemas con el objetivo de **lograr la integración** entre ellos. Los más usados son XML (*Extensible Markup Language*), YAML (originalmente *Yet Another Markup Language*, pero oficialmente *YAML Ain't Markup Language*) y JSON (*JavaScript Object Notation*) para las aplicaciones web.

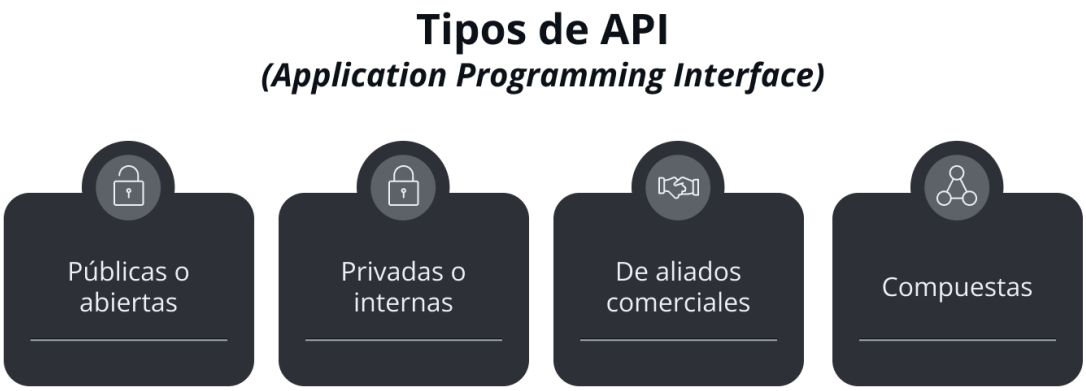
También existe un patrón en las APIs web llamado REST (*Representational State Transfer*), que es un conjunto de reglas y definiciones que permite desarrollar proyectos con interfaces bien definidas.

5.2.- ¿Qué tipos de API hay?

Hay básicamente **cuatro tipos de API en lo que se refiere a sus políticas de uso** compartido, como veremos a continuación.

❖ **1.-API según sus políticas de uso**

- **APIs públicas o abiertas:** Las APIs públicas también son conocidas como API abiertas y están disponibles para que otros usuarios o desarrolladores las empleen con mínimas restricciones o, en algunos casos incluso, están totalmente accesibles.
- **APIs privadas o internas:** Las APIs privadas o internas están ocultas de los usuarios externos y se exponen únicamente para los sistemas internos de una organización. Se emplean para el desarrollo interno de la empresa, optimizando la productividad y la reutilización de servicios.
- **APIs de aliados comerciales :** Las APIs de aliados comerciales son aquellas que se exponen entre los miembros de una alianza comercial. Como no están disponibles para todos, se necesita una autorización especial para usarlas.
- **APIs compuestas :** Las APIs compuestas utilizan distintos datos o diversas APIs de servicio y permiten que los desarrolladores puedan acceder a varios terminales. Asimismo, podemos también dividir las APIs en cuatro según lo que ofrecen o casos de uso, como verás ahora.



❖ **2.- APIs según sus casos de uso**

- **API de datos :** Las APIs de datos proporcionan a varios bancos de datos o proveedores **SaaS** (*Software as a Service* o Software como Servicio) **acceso CRUD** (*Create, Read, Update, Delete*) a conjuntos de datos subyacentes, permitiendo la comunicación entre una aplicación y un sistema de gestión de bases de datos.
- **API de sistemas operativos :** Este grupo de APIs definen cómo las aplicaciones usan los recursos disponibles y servicios del sistema operativo. Por lo que cada OS (*Operative System*) posee un conjunto de APIs, por ejemplo, Windows API o Linux API tienen el *kernel-user space API* y *kernel internal API*.
- **APIs remotas :** Este grupo define los estándares de interacción que las aplicaciones tienen en diferentes dispositivos, es decir, un software accede a ciertos recursos ubicados fuera del dispositivo que los solicita, como dice su nombre. Como dos aplicaciones se conectan de forma remota a través de una red, las APIs remotas usan protocolos para lograr la conexión.
- **APIs web :** Esta clase de API es la más común, dado que las APIs web proporcionan datos que los dispositivos pueden leer y transferirlos entre sistemas basados en la web o arquitectura cliente-servidor.

5.3.-¿Cuáles son las ventajas de las APIs?

Como casi cualquier elemento o solución de la **Industria 4.0**, las APIs son capaces de entregar diversos beneficios gracias a su uso.

A continuación, los principales:

- **Aplicaciones**: el acceso a APIs garantiza mayor flexibilidad en procesos de transferencia de información.
- **Alcance:** a través de ellas es posible crear capas de aplicaciones con el objetivo de distribuir información a diferentes audiencias.
- **Personalización:** asimismo, puede servir como solución para crear experiencias diferenciadas hacia el usuario, permitiendo adaptar protocolos, funciones y comandos según requerimientos específicos.
- **Eficiencia:** al tener contenido que se publica de forma automática y se hace disponible en diversos canales simultáneamente, las APIs permiten distribuir más eficientemente los datos.
- **Adaptabilidad:** uno de los grandes beneficios de las APIs es la capacidad que tienen de adaptarse a cambios a través de la migración de datos y la flexibilidad de servicios.

Ejemplos de API

A continuación, te mostraremos algunos de los ejemplos de API más conocidas:

- **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.
- **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.

6.-¿Qué es Postman?

Es una herramienta de desarrollo que se utiliza principalmente para probar y desarrollar **APIs** (Interfaces de Programación de Aplicaciones). Permite a los desarrolladores enviar solicitudes HTTP (como GET, POST, PUT, DELETE) a un servidor y recibir respuestas, lo que facilita la interacción con las API durante su desarrollo y prueba.

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.

6.1.-Métodos más utilizados y posibles errores

Postman cuenta con una serie de métodos que nos permiten tomar acción ante nuestras peticiones, los más utilizados son:

- **GET**: Obtener información
- **POST**: Agregar información
- **PUT**: Reemplazar la información
- **PATCH**: Actualizar alguna información
- **DELETE**: Borrar información

En cuanto a los posibles errores que podemos apreciar en la respuesta que nos ofrece la herramienta, lo resumiremos en que si la respuesta dada se encuentra en el rango de “200” quiere decir que toda la petición ha salido sin inconvenientes; mientras que el rango de los códigos de error “400” hacen referencia a errores con el cliente y aquellos errores en la línea de los “500” tienen que ver con fallos en el servidor.

7.-¿Qué es el polimorfismo?

El polimorfismo se puede definir como una condición que se presenta de muchas formas diferentes. Es un concepto en Python programación en la que un objeto definido en Python se puede utilizar de diferentes maneras. Permite al programador definir múltiples métodos en una clase derivada y tiene el mismo nombre que el presente en la clase principal. Estos escenarios admiten la sobrecarga de métodos en Python.

En Python, el polimorfismo se puede lograr de varias maneras:

1. **Métodos de la misma firma en diferentes clases**: Puedes tener métodos con el mismo nombre en diferentes clases que actúan de manera diferente según la clase del objeto que lo llama.

```python
#Poliformismo
class Perro:
    def hablar(self):
        return "Guau"

class Gato:
    def hablar(self):
        return "Miau"

def haz_hablar(animal):
    print(animal.hablar())

perro = Perro()
gato = Gato()

haz_hablar(perro)  # Imprime "Guau"
haz_hablar(gato)   # Imprime "Miau"
```

2. **Sobrecarga de operadores:** Puedes definir cómo deben comportarse los operadores (como +, -, *, etc.) para tus propias clases mediante la implementación de métodos especiales, como __add__ para la suma.

```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, otro):
        return Vector(self.x + otro.x, self.y + otro.y)

    def __repr__(self):
        return f"Vector({self.x}, {self.y})"

v1 = Vector(2, 3)
v2 = Vector(4, 1)
v3 = v1 + v2

print(v3)  # Imprime "Vector(6, 4)"
```

3. **Interfaz común:** Utilizar una interfaz común, como una clase base, donde todas las clases derivadas implementan sus propias versiones de los métodos.

```python
class Animal:
    def hablar(self):
        raise NotImplementedError("Subclases deben implementar este método")

class Perro(Animal):
    def hablar(self):
        return "Guau"
```

El polimorfismo en Python permite que diferentes tipos de datos puedan ser utilizados de manera intercambiable, siempre y cuando implementen los métodos o interfaces esperados. Esto hace que el código sea más flexible y reutilizable.

8.-¿Qué es un método dunder?

En Python, "dunder" es una abreviatura de "double underscore" (doble guión bajo). Se refiere a los métodos y atributos especiales que tienen dos guiones bajos al principio y al final de su nombre. Estos métodos son también conocidos como métodos mágicos o métodos especiales.

Algunos ejemplos comunes de métodos dunder son:

- **__init__:** Es el constructor de la clase. Se llama cuando se crea una nueva instancia de la clase.
- **__str__:** Define cómo se convierte un objeto en una cadena de texto. Se usa cuando utilizas print() o str().
- **__repr__:** Define cómo se representa el objeto en una forma que puede ser evaluada por eval() (o al menos sea una representación válida). Se usa en la consola o para depuración.
- **__add__:** Permite definir el comportamiento del operador + para los objetos de la clase.
- **__getitem__:** Permite definir cómo se accede a los elementos de un objeto utilizandola sintaxis de corchetes, como en obj[key].

```python
#Ejmplos Dunder
class Punto:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return f"Punto({self.x}, {self.y})"
    def __repr__(self):
        return f"Punto({self.x}, {self.y})"
    def __add__(self, otro):
        return Punto(self.x + otro.x, self.y + otro.y)
    def __getitem__(self, index):
        if index == 0:
            return self.x
        elif index == 1:
            return self.y
        else:
            raise IndexError("Índice fuera de rango")

p1 = Punto(1, 2)
p2 = Punto(3, 4)
```

```python
print(p1)           # Imprime "Punto(1, 2)"
print(repr(p1))     # Imprime "Punto(1, 2)"
print(p1 + p2)      # Imprime "Punto(4, 6)"
print(p1[0])        # Imprime "1"
```

En este código, __init__ inicializa los atributos x e y, __str__ y __repr__ proporcionan representaciones en cadena del objeto, __add__ define el comportamiento del operador +, y __getitem__ permite el acceso a los atributos usando la sintaxis de corchetes.

Los métodos dunder permiten personalizar y extender el comportamiento de las instancias de tus clases en Python, haciéndolos una herramienta poderosa en la programación orientada a objetos.

9.-¿Qué es un decorador de python?

Los decoradores son una potente característica de Python que permite modificar o mejorar funciones, clases u otros objetos en tiempo de ejecución sin modificar directamente su código fuente.

Los decoradores se implementan utilizando funciones o clases y se utilizan para envolver o decorar el objeto de destino con funcionalidad adicional.

- **¿Por qué los decoradores son importantes en Python?**

Los decoradores juegan un papel crucial en la programación en Python por varias razones:

a) Reutilización del código:

Los decoradores permiten separar las preocupaciones transversales, como el registro, la validación, la autenticación o el almacenamiento en caché, de la lógica central de las funciones o clases. Esto favorece la reutilización del código y la modularidad.

b) Facilidad de lectura y mantenimiento:

Los decoradores proporcionan una forma limpia y concisa de añadir funcionalidad al código existente sin saturar la implementación original. Mejoran la legibilidad del código aislando comportamientos específicos y facilitando su comprensión y mantenimiento.

c) Metaprogramación y extensibilidad:

Los decoradores permiten la modificación dinámica del código en tiempo de ejecución, lo que posibilita técnicas avanzadas de metaprogramación. Proporcionan un mecanismo flexible para ampliar y personalizar el comportamiento de funciones o clases sin modificar su definición original.

- **Decoradores de Funciones en phyton**

Los decoradores de funciones en Python permiten modificar el comportamiento de una función sin modificar directamente su código fuente. Se implementan utilizando funciones o clases de orden superior. Cuando se decora una función, se pasa como argumento al decorador y éste devuelve una versión modificada de la función.

- **Sintaxis y uso de los decoradores:**

Los decoradores se implementan utilizando el símbolo "@" seguido del nombre de la función o clase decoradora. Se colocan antes de la definición de la función que se va a decorar.

La función/clase decoradora se encarga de modificar o mejorar el comportamiento de la función de destino. El decorador se aplica a una función o método usando el símbolo @ antes del nombre de la función decoradora. Esto ocurre justo encima de la función que se desea modificar. He aquí un ejemplo:

```python
#Ejemplo Decorador
def decorador(funcion_original):
    def nueva_funcion():
        print("Esto se ejecuta antes de la función original.")
        funcion_original()
        print("Esto se ejecuta después de la función original.")
    return nueva_funcion

@decorador
def funcion_principal():
    print("Esta es la función principal.")

# Llamando a la función
funcion_principal()
```

Output:

```
Esto se ejecuta antes de la función original.
Esta es la función principal.
Esto se ejecuta después de la función original.
```

Explicación:

1. **Decorador**: decorador es una función que toma otra función (funcion_original) como argumento.
2. **Función interna**: Dentro del decorador, definimos una nueva función (nueva_funcion) que ejecuta código antes y después de llamar a la función original.
3. **@decorador**: Aplicar el decorador a funcion_principal significa que cada vez que llamemos a funcion_principal, en realidad se ejecutará nueva_funcion que contiene código adicional.