M3C6 Python Assignment 4

A.- Preguntas teóricas.

1.- ¿Para qué usamos clases en Python?

Antes de indicar para que usamos las clases en Python veamos qué es Python.

Python es un lenguaje de programación orientado a objetos, en inglés “Object Oriented Programming” (OOP).

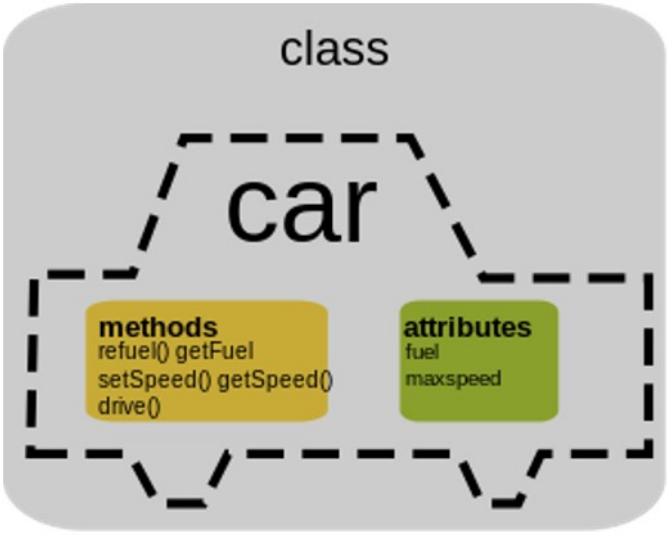
La programación orientada a objetos es un tipo de paradigma (modelo) de programación basado alrededor de clases de programación y ejemplos de clases llamadas objetos.

Pero, ¿que es una clase y qué es un objeto?

Una clase es una colección de objetos de tipo similar. Una vez que se define una clase, cualquier objeto nuevo creado puede pertenecer a esa clase.

Los objetos son las entidades de ejecución básica. Cuando se ejecuta un programa, los objetos interactúan entre sí mediante el envío de mensajes. Diferentes objetos también pueden interactuar entre sí sin conocer los detalles de sus datos o de código. Dicho de forma breve: un objeto es una cosa. Y, si una cosa es un sustantivo, entonces un objeto es un sustantivo. Además hay que saber que un objeto puede tener otros objetos.

Veamos un ejemplo gráfico con el ejemplo de un coche.



Fuente: cursosgis.com

Un objeto, un coche en este caso, puede tener color, un tamaño, etc. Estos son los atributos, definen al objeto.

De estos atributos se generan las cualidades de los atributos que son los adjetivos en sí: verde, grande, etc.

Este objeto puede tener otros objetos, atributo-objeto, por ejemplo ventana, que a su vez posee sus propios atributos y cualidades.

Además de lo indicado anteriormente, para entender mejor el concepto sobre las clases es necesario saber que son los métodos, también denominadas funciones. Los métodos o funciones representan acciones propias que puede realizar el objeto, y no otro.

En el siguiente gráfico podemos entender de una forma más clara el concepto de clase y su sentido respecto a un objeto:

Fuente: cursosgis.com

Volviendo a la pregunta de para qué usamos las clases en Python:

Las clases son una estructura de programación que permiten definir un conjunto de métodos y atributos que describen un objeto. Usamos las clases como modelos sobre los cuáles se construirán nuestros objetos, mientras que los objetos son instancias de esa clase.

Las clases nos permiten empaquetar o envolver datos y funcionalidad o comportamientos juntos.

Sobre el ejemplo del coche utilizado anteriormente, un ejemplo básico de sintaxis de clase sería la siguiente:

```
class Coches:

    def marca (self):

        return f"Este coche es rápido."

coche_uno = Coches()

print(coche_uno.marca())
```

El uso de las clases en Python tiene ventajas pero también desventajas. En cuanto a ventajas se pueden señalar las siguientes:

1. **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.

2. **Encapsulación**: permiten ocultar la complejidad de un objeto y exponer solo una interfaz simple y fácil de usar para interactuar con él.

3. **Modularidad**: pueden descomponer un programa en componentes más pequeños y manejables, lo que facilita el mantenimiento y la solución de problemas.

4. **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.

En cuanto a las desventajas se pueden observar las siguientes:

1. **Sobrecarga de complejidad:** las clases pueden agregar complejidad adicional a un programa y hacer que sea más difícil de entender y depurar.

2. **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.

3. **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.

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

Se ejecuta la denominada función contructora `__init__`.

El método `__init__` es un método de los llamados especiales. Es una función que está disponible para todas las clases.

El objetivo fundamental del método `__init__` es inicializar los atributos del objeto que creamos. Es una función que se llamará automáticamente cada vez que creamos una instanciación, que es el proceso de creación de un objeto.

Las ventajas de implementar el método `__init__` en lugar del método inicializar son:

1. Es el primer método que se ejecuta cuando se crea un objeto.
2. Se llama automáticamente. Es decir es imposible de olvidarse de llamarlo ya que se llamará automáticamente.
3. Quien utiliza POO (OOP) en Python (Programación Orientada a Objetos) conoce el objetivo de este método.

Otras características del método `__init__` son:

1. Se ejecuta inmediatamente tras crear un objeto.
2. No puede retornar dato.
3. Puede recibir parámetros que se utilizan normalmente para inicializar atributos.
4. Es un método opcional, pero en la creación de código es muy habitual declararlo.

La sintaxis de este constructor es la siguiente:

```
def __init__([parámetros]):
        [algoritmo]
```

3.- ¿Cuáles son los ~~(tres)~~ verbos de API REST?

Primero veamos que es una API REST:

Una API REST es una interfaz de comunicación entre sistemas de información que usa el protocolo de transferencia de hipertexto (hypertext transfer protocol o HTTP) para obtener datos o ejecutar operaciones sobre dichos datos en diversos formatos, como pueden ser XML o JSON.

Se basa en el modelo cliente-servidor donde el cliente es el que solicita obtener los recursos o realizar alguna operación sobre dichos datos, mientras que el servidor es aquel ente que entrega o procesa dichos datos a solicitud del cliente.

Para identificar si una API es REST o no lo es, existen algunos criterios como los siguientes:

1. Debe usar una arquitectura cliente-servidor.

2. Las ejecuciones de la API no deben considerar el estado del cliente, el estado de peticiones anteriores o algún indicador almacenado que haga variar su comportamiento. La comunicación debe ser sin estado (stateless).

3. Ha de estar orientada a recursos, usando las operaciones estándar de los verbos HTTP.

4. Hace uso de la URL como identificador único de los recursos.

5. Debe ser hipermedia: cuando se consulte un recurso, este debe contener links o hipervínculos de acciones o recursos que lo complementen.

La documentación donde se describe el comportamiento de una API se denomina especificación de una API (API Spec). La finalidad de dicha documentación es guiar al desarrollador que va a integrar el uso de la API en su sistema.

Los componentes primordiales que se describen en la especificación de una API son los llamdado **verbos propios del protocolo HTTP** que fueron tomados para definir operaciones muy puntuales y específicas sobre los recursos de la API.

Los verbos más utilizados en una API REST son los siguientes:

1. `GET`: listado de recursos. Detalle de un solo recurso.
2. `POST`: creación de un recurso.
3. `PUT`: modificación total de un recurso.
4. `PATCH`: modificación parcial de un recurso.
5. `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.

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

MongoDB es una base de datos NoSQL orientada a documentos y de código abierto.

MongoDB apareció a mediados de la década del año 2000. Se utiliza para almacenar volúmenes masivos de datos.

A diferencia de una base de datos relacional SQL tradicional (MySQL, PostgreSQL, MariaDB, Microsoft SQL Server u Oracle Database), MongoDB no se basa en tablas y columnas. Los datos se almacenan como colecciones y documentos.

Soporta datos SQL y, además, guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Cada base de datos MongoDB contiene colecciones, que a su vez contienen documentos. Cada documento es diferente y puede tener un número variable de campos. El tamaño y el contenido de cada documento también varían.

La estructura de un documento corresponde a la forma en que los desarrolladores construyen sus clases y objetos en el lenguaje de programación utilizado. En general, **las clases** no son filas y columnas, sino que **tienen una estructura clara formada por pares Value/key**.

En MongoDB los documentos no tienen un esquema predefinido y los campos pueden añadirse a voluntad. El modelo de datos disponible en MongoDB facilita la representación de relaciones jerárquicas u otras estructuras complejas.

El modelado de los datos y la estructura de los documentos sólo deben responder a las necesidades del usuario. Es importante tener en cuenta las necesidades de la aplicación y, por tanto, qué datos y tipos de datos se necesitarán.

Si se esperan muchas consultas, es muy recomendable utilizar índices en el modelo de datos para mejorar la eficiencia de las consultas.

Si se producen frecuentes adiciones, actualizaciones y eliminaciones de datos, conviene utilizar los índices y el sistema de fragmentación para mejorar la eficacia global del entorno.

Una característica importante de MongoDB es la elasticidad de sus entornos. Muchas empresas tienen clusters de más de 100 nodos para bases de datos que contienen millones de documentos.

Las **ventajas más destacables** para utilizar MongoDB son las siguientes:

Esta base de datos es **muy flexible** y se adapta a los casos de uso concretos de una empresa.

Las consultas adecuadas permiten encontrar campos específicos dentro de los documentos. También **es posible crear índices** para mejorar el rendimiento de las búsquedas. Se puede indexar cualquier campo.

Otra característica importante es la **posibilidad de crear conjuntos de réplicas** formados por dos o más instancias de MongoDB. Cada miembro puede actuar como réplica secundaria o primaria en cualquier momento:

> La **réplica primaria** es el servidor principal, que interactúa con el cliente y realiza todas las operaciones de lectura y escritura.
>
> Las **réplicas secundarias** mantienen una copia de los datos. Así, en caso de fallo de la réplica primaria, el cambio a la secundaria se realiza automáticamente. Este sistema garantiza una alta disponibilidad.

MongoDB también permite implementa el “sharding” (dividir una base de datos en múltiples fragmentos más pequeños llamados "shards"), esto es, **permite el escalado horizontal** al distribuir los datos entre múltiples instancias de MongoDB. La base de datos puede ejecutarse en varios servidores, y esto permite equilibrar la carga o duplicar los datos para mantener el sistema en funcionamiento en caso de fallo del hardware.

Debido a estas numerosas ventajas, MongoDB es ahora una herramienta muy utilizada en el campo de la ingeniería de datos. Es una solución imprescindible para los ingenieros de datos.

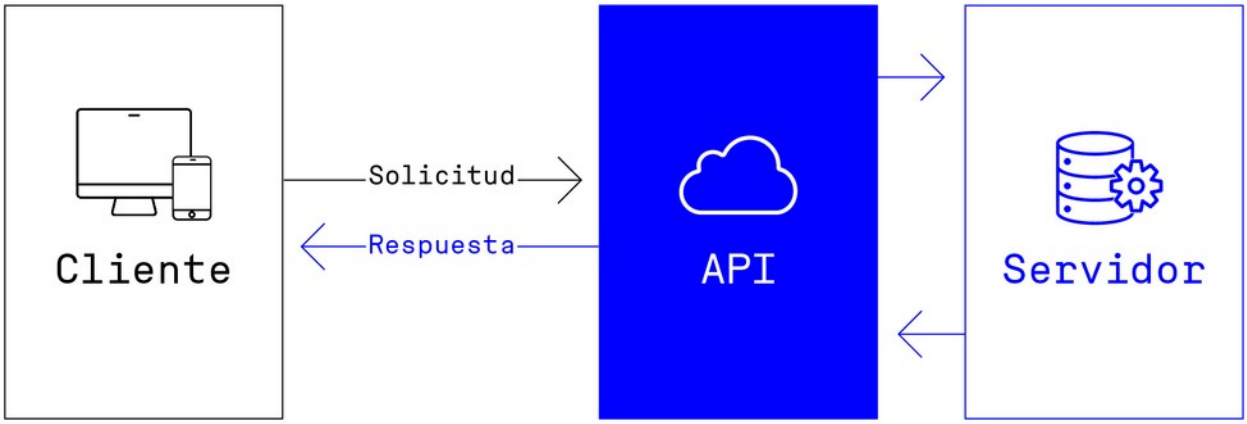
Los desarrolladores utilizan MongoDB por su flexibilidad, escalabilidad, rendimiento y ecosistema: capacidades fundamentales y necesarias para crear y potenciar aplicaciones modernas.

5.- ¿Qué es una API?

API significa interfaz de programación de aplicaciones. Es una forma en que podemos comunicarnos con una aplicación y sin necesidad de implementar un raspador, "scraper".

Nos da un conjunto de comandos que podemos usar y podemos hacer que nuestra aplicación se comunique con otro servidor en otra aplicación.

Lo que conseguimos con una API es un conjunto de endpoints (puntos finales). Un endpoint es sólo una URL que nos devuelve datos JSON que una aplicación externa pueda usar.



Fuente: gluo.mx

Por ejemplo, si estás creando un React (librería o biblioteca de Javascript que sirve para construir interfaces de usuario), puedes comunicarte con esa URL y recuperar los datos de una forma que puedas anilizarlos y después mostrarlos en pantalla.

En el curso se nos ha enseñado como crear nuestra primera API en la parte del servidor, en Flask, aprovechando el lenguaje Python.

Para ello hemos tenido que haber instalado previamente el entorno "pipenv", darnos de alta en Postman y utilizar un editor de texto, en mi caso, Visual Studio Code.

6.- ¿Qué es Postman?

Postman es una **plataforma que permite y hace más sencilla la creación y el uso de APIs**. Esta herramienta es muy útil para programar porque da la posibilidad hacer pruebas y comprobar el correcto funcionamiento de los proyectos que realizan los desarrolladores web.

Postman es una plataforma con funciones muy útiles para los desarrolladores web. Algunas de ellas son las enumeradas a continuación:

1. **Desarrollo de peticiones**: ofrece la posibilidad de desarrollar y crear peticiones HTTP a cualquier API a través de una interfaz gráfica. De esta manera, se convierte en una herramienta muy práctica para programar y realizar pruebas para comprobar que los desarrollos que están llevando a cabo los programadores se están ejecutando correctamente.

2. **Crear y probar colecciones de APIs**: podemos probar colecciones de APIs, ya sea para Frontend como Backend. Las colecciones de Postman ayudan a los desarrolladores a organizar las solicitudes de API que están relacionadas.

3. **Gestión del Ciclo de Vida de las API**: permite hacer una gestión del ciclo de vida de las APIs, de su conceptualización, desarrollo, ejecución y mantenimiento.

4. **Administrar documentación**: da la posibilidad de crear documentación fundamentada en las API y en las colecciones que se hayan desarrollado y, además hacerla pública.

5. **Trabajar con entornos de colaboración**: permite trabajar con entornos para después poder compartir la información con el resto de miembros del equipo que forman parte del desarrollo de la API.

6. **Establecer variables**: pueden establecerse variables de entorno y globales, que posteriormente se utilizarán en las pruebas de las APIs. Permite definir todas las variables que puedan necesitar los desarrolladores y organizarlas por entornos de trabajo. Esta es una funcionalidad muy útil cuando se llevan a cabo distintos proyectos o cuando quieres ejecutar varios entornos para un mismo proyecto.

7. **Automatizar test de integración**: permite automatizar de forma muy simple test de integración para los proyectos. La plataforma hace uso de Javascript para programar los test. Si no se tienen conocimientos de Javascript, la plataforma ofrece un grupo de snippets que permiten desarrollar los test muy fácilmente.

En cuanto a **ventajas destacables por el uso de esta herramienta** señalamos las siguientes:

1. Postman tiene una comunidad muy grande de usuarios.
2. Su interfaz es muy intuitiva y sencilla.
3. La plataforma puede integrarse con otras herramientas.
4. Ofrece la posibilidad de añadir scripts (JavaScript) para automatizar, configurar pruebas o agregar validaciones.
5. La herramienta permite realizar trabajos en los que se puede colaborar con el resto de miembros de un equipo.
6. Permite trabajar con colecciones.
7. Nos permite establecer comunicación con APIs externas y probarlas.

Este software fue creado en 2012 por Abhinav Asthana, Ankit Sobti y Abhijit Kane en Bangalore, India, para resolver el problema de compartir pruebas API.

7.- ¿Qué es el polimorfismo?

Polimorfismo es un concepto que quiere expresar que un elemento puede tener muchos cambios o muchas formas.

Dicho de otra forma: **es la capacidad de objetos de diferentes clases para responder al mismo mensaje.**

Dos objetos de diferentes clases pueden tener métodos con el mismo nombre, y ambos métodos pueden ser llamados con el mismo código, dando respuestas diferentes.

El polimorfismo es una técnica de la programación orientada a objetos que permite a distintos objetos responder de manera diferente a un mismo llamado de método. En Python esto se logra gracias al uso de clases y funciones.

Los objetos de diferentes clases pueden ser accedidos utilizando el mismo interfaz, mostrando un comportamiento distinto según cómo sean accedidos.

En Python no es necesario que los objetos compartan un interfaz, simplemente basta con que tengan los métodos que se quieren llamar.

Para entender de la mejor manera posible el concepto de poliformismo, hay un recurso al que se denomina “tipado de pato” (duck typing) y que indica lo siguiente: “si habla como un pato, si camina como un pato,... entonces pato será”.

Aplicado a la programación podemos decir: “si un objeto creado en Python se comporta de una determinada manera, no importa tanto el tipo de objeto como el método que utilizamos para llegar hasta él”. Esto es, expresándonos en “lenguaje patuno”, diremos que llegamos al “pato” (objeto) por su “parpeo” y sus “andares” (comportamiento, método).

En Python nos importan más los métodos que los tipos de objetos.

8.- ¿Qué es un método dunder?

Dunder son los **métodos que comienzan con guiones bajos dobles**, que tienen algunos nombres de métodos como “init” y luego terminan con dos guiones bajos más: `__init__`

Este método nos permite asignar atributos y realizar operaciones con el objeto en el momento de su creación. También es ampliamente conocido como el constructor.

También son llamados “métodos especiales” o “métodos mágicos” y, los más comunes, pueden ser clasificados, según su utilidad, de la forma siguiente:

1. Para operaciones aritméticas:

    `__add__` (suma)

    `__sub__` (resta)

    `__mul__` (multiplicación)

    `__div__` (división)

2. Para operaciones de comparación:

    `__lt__` (menor que)

    `__gt__` (mayor que)

    `__eq__` (igual que)

3. Para operaciones del “ciclo de vida”:

    `__init__` (crear objeto)

    `__del__` (eliminar objeto)

4. Para operaciones de representación:

    `__str__` (devuelve una representación de cadena legible por humanos)

    `__repr__` (devuelve una representación de cadena)

9.- ¿Qué es un decorador de python?

Es una función que recibe otra función como parámetro, le añade cosas y retorna otra función diferente.

Son muy útiles. Nos permiten envolver una función dentro de otra y modificar el comportamiento de esta última sin modificarla permanentemente.

Los decoradores se aplican utilizando el símbolo @ seguido del nombre de la función decoradora justo encima de la función que deseas decorar:

```
@decorador

def mi_funcion():

  # Tu función especifica

  return # El resultado que quieres devolver con tu función
```

B.- Ejercicio práctico.

1.- Cree una clase de Python llamada Usuario que use el método init y cree un nombre de usuario y una contraseña. Crea un objeto usando la clase.

Comprobar respuesta en el siguiente enlace:

https://github.com/VicenteHeredia/Checkpoint_6.git

```
class Usuario:
    def __init__(self, usuario, contraseña):
        self.usuario = usuario
        self.contraseña = contraseña

    def name(self):
        return f"Para el usuario {self.usuario} la contraseña es: {self.contraseña}"

usuario_uno = Usuario('Pepe', 'pepe0123')

print(usuario_uno.name())
```