1. ¿Para qué usamos Clases en Python?

Las clases en Python son estructuras fundamentales que nos permiten modelar y representar objetos del mundo real en nuestro código. Utilizamos clases para encapsular datos (propiedades) y comportamientos (métodos) relacionados en un solo objeto. Esto nos ayuda a organizar nuestro código de manera más modular y orientada a objetos, lo que a su vez facilita la reutilización del código y el mantenimiento del mismo a medida que nuestros programas crecen en complejidad.

Algunas de las razones principales para usar clases en Python son:

- **Organización del código**: Las clases permiten organizar el código de manera estructurada y modular, lo que facilita la comprensión y el mantenimiento del programa.

- **Reutilización del código**: Una vez definida una clase, se puede crear cualquier número de objetos (instancias) de esa clase, lo que permite reutilizar el código sin necesidad de volver a escribirlo.

- **Abstracción**: Las clases permiten abstraer los detalles de implementación de un objeto, lo que significa que los usuarios solo necesitan conocer cómo interactuar con el objeto a través de sus métodos públicos, sin necesidad de entender cómo funciona internamente.

- **Herencia**: La herencia permite crear nuevas clases basadas en clases existentes, lo que permite extender y modificar el comportamiento de las clases de manera eficiente.

- **Polimorfismo**: Las clases y los objetos en Python pueden exhibir comportamientos diferentes en función del contexto en el que se utilizan, lo que facilita la creación de código flexible y genérico.
  En resumen, las clases en Python proporcionan una forma poderosa y flexible de organizar y estructurar el código, lo que permite desarrollar programas más escalables, mantenibles y fáciles de entender.

Ejemplo:

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def Woof(self):
        return f"{self.name} is woofing."
```

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

Cuando creamos una instancia (es decir, un objeto) de una clase en Python, el método que se ejecuta automáticamente es el método **__init__()**, también conocido como el constructor de la clase. Este método se utiliza para inicializar las propiedades del objeto recién creado.

¿Por qué se utiliza? El método **__init__()** se utiliza para garantizar que el objeto tenga un estado inicial coherente al ser creado. Esto significa que podemos establecer valores predeterminados para las propiedades del objeto o realizar cualquier otra inicialización necesaria.

¿Para qué se utiliza? Se utiliza para asignar valores iniciales a las propiedades de un objeto, lo que garantiza que el objeto esté en un estado válido desde el momento de su creación.

Sintaxis: El método **__init__()** se define dentro de la clase y toma al menos un parámetro, **self**, que hace referencia al objeto mismo. Además, puede tomar otros parámetros para inicializar las propiedades del objeto según sea necesario.

Ejemplo:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
# Crear una instancia de la clase Person
person1 = Person("John", 25)
```

3. ¿Cuáles son los tres verbos de API?

Los tres verbos principales que se utilizan comúnmente en las API RESTful (Interfaz de Transferencia de Estado Representacional) son:

- **GET:** Se utiliza para recuperar datos de un recurso específico o una colección de recursos. Por ejemplo, al realizar una solicitud GET a una API de redes sociales, podríamos recuperar los detalles de un usuario específico o una lista de publicaciones en un feed.

- **POST:** Se utiliza para crear un nuevo recurso. Al realizar una solicitud POST a una API, se envían datos al servidor para crear un nuevo objeto o realizar alguna acción. Por ejemplo, al publicar un nuevo mensaje en una red social, se puede utilizar una solicitud POST para enviar el contenido del mensaje al servidor.

- **DELETE:** Se utiliza para eliminar un recurso específico en el servidor. Al realizar una solicitud DELETE a una API, se elimina el recurso identificado por el URI en la solicitud. Por ejemplo, al realizar una solicitud DELETE a una API de correos electrónicos, se podría eliminar un correo electrónico específico de la bandeja de entrada.

Estos tres verbos (GET, POST y DELETE) son esenciales en el diseño de API RESTful y se utilizan para realizar operaciones básicas de lectura, actualización y eliminación en recursos a través de HTTP.

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

MongoDB es una base de datos de tipo NoSQL que destaca por su enfoque en documentos y colecciones de documentos.

Algunas características clave de MongoDB son:

- Documentos BSON: Almacena datos en formato BSON (Binary JSON), que es una representación binaria de JSON (notación de objetos de Javascript). Cada registro es un documento flexible con pares clave-valor, independiente del resto de registros.

- Esquema dinámico: Permite esquemas dinámicos, lo que significa que cada documento en una colección puede tener un esquema diferente.

- Escalabilidad horizontal: Puede escalar horizontalmente de manera eficiente al distribuir los datos en clústeres y agregar más nodos según sea necesario.

- Buena para datos no estructurados: Ideal para almacenar y procesar datos no estructurados o semiestructurados, como documentos, logrando flexibilidad y velocidad.

En conclusión, MongoDB es una base de datos que destaca por su flexibilidad, escalabilidad horizontal y capacidad para manejar datos no estructurados, es decir, las características de una base de datos NoSQL, lo que la hace adecuada para aplicaciones modernas con requisitos de almacenamiento de datos dinámicos, tanto en tamaño como en tipo.

5. ¿Qué es una API?

Una API, Interfaz de Programación de Aplicaciones, es la infraestructura en código que permite que varias aplicaciones se comuniquen entre ellas compartiendo datos, elementos, información y funcionalidades. Proporciona una forma estandarizada para que los diferentes componentes de software se comuniquen y compartan datos entre sí de manera eficiente y segura.

Sería similar a un camarero que toma comandas y entrega platos en un restaurante. Por un lado, tenemos a los clientes y, por otro, a los cocineros, que se comunican entre ellos gracias al camarero (la API). En definitiva, las APIs son las intermediarias en el proceso de intercambio de información entre aplicaciones.

Las API son utilizadas para diferentes propósitos, incluyendo:

- **Acceso a servicios web:** Permiten a las aplicaciones acceder y consumir datos y funcionalidades ofrecidos por otros sistemas a través de la web, como redes sociales, servicios de pago, servicios de almacenamiento en la nube, etc.
- **Integración de sistemas:** Facilitan la comunicación y la interoperabilidad entre diferentes sistemas informáticos, permitiendo que se compartan datos y funcionalidades de manera eficiente.
- **Desarrollo de aplicaciones:** Las API proporcionan a los desarrolladores las herramientas necesarias para construir aplicaciones complejas al proporcionar acceso a funciones predefinidas y datos estructurados.
- **Automatización de procesos:** Permiten la automatización de tareas repetitivas o complejas al proporcionar acceso programático a funcionalidades y datos de sistemas externos.
  En resumen, una API actúa como una interfaz estandarizada que permite a diferentes aplicaciones y sistemas comunicarse entre sí de manera efectiva, permitiendo el intercambio de datos y la ejecución de acciones de manera controlada y segura.

6. ¿Qué es Postman?

Trabajar directamente con APIs puede ser algo complicado, considerando que los datos que se comparten entre aplicaciones, y la manera de hacerlo, suelen estar en formatos poco legibles por los humanos.

Por ese motivo se han creado programas como Postman que facilitan el desarrollo, prueba y documentación de APIs. Permite a los desarrolladores web enviar solicitudes HTTP a sus API, observar las respuestas y colaborar con otros miembros del equipo en el desarrollo de API.

Postman ofrece una interfaz gráfica intuitiva para los humanos, al contrario que el código en bruto de las APIs, para crear, enviar y gestionar solicitudes, así como la capacidad de automatizar pruebas y guardar colecciones de solicitudes para su reutilización/automatización.

Además, proporciona herramientas para la documentación de API, lo que hace que sea más fácil compartir información sobre cómo utilizar una API específica que hayamos creado.

En definitiva, Postman es una herramienta extremadamente útil para trabajar con APIs durante todo el ciclo de su desarrollo, desde las pruebas iniciales de manera local hasta su despliegue en la nube.

7. ¿Qué es el polimorfismo?

El polimorfismo es un concepto de la programación orientada a objetos que se refiere a la capacidad de diferentes objetos de responder de manera única a la misma llamada de método. Permite que un mismo método se comporte de manera diferente en función del objeto que lo llama.

Polimorfismo hace referencia a la capacidad de una función, método o, incluso, clase de trabajar con diferentes tipos de objeto y comportarse de manera distinta dependiendo de la situación.

En resumen, el polimorfismo en la programación orientada a objetos permite que diferentes objetos respondan de manera diferente a los mismos mensajes, lo que lleva a un código más flexible, genérico y fácil de mantener.

8. ¿Qué es un método dunder?

Un método dunder, también conocido como "método mágico" o "método especial", es un método en Python que utiliza doble guion bajo (___) al principio y al final de su nombre. Estos métodos tienen un significado especial en el contexto de las clases y objetos en Python.

Los métodos dunder son utilizados por el propio intérprete de Python para realizar operaciones especiales. Por ejemplo, el método **___init___()** se utiliza para inicializar un objeto cuando se crea una instancia de una clase, mientras que el método **___str___()** se utiliza para devolver una representación de cadena legible para humanos del objeto.

Estos métodos proporcionan una forma de personalizar el comportamiento de las clases y objetos en Python, lo que permite definir cómo se deben comportar los objetos en diferentes situaciones.

Los métodos dunder son una parte fundamental de la programación en Python y se utilizan en muchas bibliotecas y marcos de trabajo para definir el comportamiento de las clases y objetos de manera flexible y poderosa.

Ejemplo:

```
class Libro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor
    # Método dunder para representar el objeto como una cadena
    def __str__(self):
        return f"Libro: {self.titulo} - Autor: {self.autor}"
# Crear un objeto de la clase Libro
libro1 = Libro("El código Da Vinci", "Dan Brown")
# Imprimir el objeto utilizando el método dunder __str__()
print(libro1)
# Output: Libro: El código Da Vinci - Autor: Dan Brown
```

9. ¿Qué es un decorador de Python?

Un decorador es una función que toma otra función como argumento y agrega alguna funcionalidad adicional a esa función sin modificar su código. Los decoradores proporcionan una forma flexible de extender o modificar el comportamiento de las funciones o métodos existentes de manera transparente.

Los decoradores son especialmente útiles cuando queremos agregar características comunes a varias funciones o métodos sin tener que repetir el mismo código en cada una de ellas. Por ejemplo, podríamos usar un decorador para realizar el registro de actividad, validar permisos de acceso, medir el tiempo de ejecución, o manejar excepciones, entre otras cosas

Ejemplo:

```
class Factura:
    def __init__(self, cliente, total):
        self._cliente = cliente
        self._total = total
    def formateador(self):
       return f'{self._cliente} debe: ${self._total}'
    @property
    def cliente(self):
        return self._cliente
    @property
    def total(self):
        return self._total
# Crear una instancia de la clase Factura
empresa = Factura('empresa1', 100)
# Acceder a los atributos cliente y total
print(empresa.cliente)
print(empresa.total)
```