Documentacion

1. ¿Para qué usamos Clases en Python?

R/ Las usamos para crear paquetes que contienen datos y funcionalidades juntos; permiten definir un conjunto de métodos y atributos que describen un objeto.
Las clases son fundamentales en la programación orientada a objetos, los objetos creados a partir de una clase tienen el mismos comportamiento y características definidos por la clase.
En Python las clases se definen de la siguiente manera:

```
class Usuario:
      def __init__(self, nombre, contraseña):
            self.nombre = nombre                #atributo nombre
            self.contraseña = contraseña        #atributo contraseña

      def saludar(self):
           print("Hola mi nombre es" + self.nombre)
```

Ejemplo No. 1

En el ejemplo anterior la palabra "class" se usa para empezar a definir una clase seguido del nombre que se quiere asignar a la clase en este caso "Usuario" y después dos puntos (:) luego lo que viene dentro de la clase, que son los métodos y atributos que, en el ejemplo de la clase Usuario tiene dos métodos __init__ que es un método propio de Python y saludar, dentro del método __init__ tenemos los atributos nombre y contraseña.

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

R/ El método __init__, como se menciono en el ejemplo No. 1 este es un método propio de Python, este método crea los atributos que deben tener todos los objetos de la clase y por tanto contiene los parámetros necesarios para su creación, pero no devuelve nada.
Este método es llamado automáticamente cuando se ejecuta el código.

```
def __init__(self, nombre, contraseña):
            self.nombre = nombre                #atributo nombre
            self.contraseña = contraseña        #atributo contraseña
```

Ejemplo No. 2

Los atributos que se crean dentro del método __init__ se conocen como atributos del objeto, mientras que los que se crean fuera de él se conocen como atributos de la clase.

3. ¿Cuáles son los tres verbos de API?

R/

GET

Se utiliza para recuperar o consultar información desde un recurso. No se produce ningún efecto secundario cuando un cliente realiza esta llamada, no importa cuantas veces realicemos una petición get el resultado que obtenido siempre sera el mismo.

Cuando ingresamos a la dirección usando GET https:///guides/books/ estamos solicitando que se nos entregue el recurso identificado por /guides/books, este es un buen ejemplo de uso con GET.

POST

Se utiliza para crear un nuevo recurso. Cada llamada POST produce un nuevo recurso Normalmente, la acción POST se dirige a un recurso que representa una colección, para indicar que un nuevo recurso debe agregarse a dicha colección, por ejemplo POST /guides para agregar un nuevo recurso a la colección guides.

PUT

Son muy similares ya que los dos son utilizados para modificar o actualizar un recurso existente. Un escenario común para el uso de PUT sería para actualizar la información de una guía, por ejemplo:

PUT /guides/books

DELETE

Se utiliza para eliminar registros, bien pudiera ser para eliminar un recurso individual como en:

DELETE /guides/books

O para eliminar una colección completa:

DELETE /guides

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

R/ MongoDB es una base de datos multiplataforma de código abierto NoSQL orientada a documentos. Se diferencia de las bases de datos relacionales por su flexibilidad y rendimiento.

A diferencia de una base de datos relacional SQL tradicional como MySQL, MongoDB no se basa en tablas y columnas. Los datos se almacenan como colecciones y documentos.

Los documentos son pares value/key que sirven como unidad básica de datos. Las colecciones contienen conjuntos de documentos y funciones. Son el equivalente a las tablas en las bases de datos relacionales clásicas.

MondoDB usa JavaScript como lenguaje de consulta y representa los datos como documentos JSON. No requiere un esquema predefinido, lo que significa que los documentos de una misma colección pueden tener diferentes estructuras.

Para insertar un registro en una base de datos MongoDB se puede hacer de la siguiente manera:

```
db.books.insertOne({
  "name": "OOP Programming",
  "publishedDate": new Date(),
  "authors": [
    {"name": "Jon Snow"},
    {"name": "Ned Stark"}
  ]
})
```

Ejemplo No. 3

En el ejemplo No. 3 books seria la tabla y name, publishedDate, y authors serian las columnas en una base de datos SQL.

5. ¿Qué es una API?

R/ API (Interfaz de Programación de Aplicaciones) permite a dos componentes de software comunicarse entre si, funciona en términos de cliente y servidor, donde el cliente envía una petición y recibe una respuesta del servidor.

Existen cuatro formas de funcionamiento de las APIs que son:

API de 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.

API de 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.

API de 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.

API de 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. Este tipo de API son como la que vemos en el ejemplo de la imagen No.1.

6.¿Qué es Postman?

R/ Es una aplicación que dispone de herramientas propias que nos permiten realizar peticiones de forma simple para probar APIs propias o de terceros. Puede se utilizada en sistemas operativos Windows, Linux y Mac.

Se puede testear bajo los verbos mencionados en el numeral 3 de guía, la plataforma también nos ofrece la oportunidad de organizar en carpetas, funcionalidades y módulos los servicios web, generar documentacion de nuestras aplicaciones, entre muchas otras cosas.

Cabe resaltar que hay una versión gratuita y versiones de pago con diferentes planes según el uso que se le vaya a dar, un ejemplo de uso de postman en la versión gratuita seria el siguiente, donde se realiza una petición POST para añadir un nuevo registro a guides, al dar click en send lo que se ejecutara sera poner una guía con el titulo de update cindy guide y que el contenido sea content, si al ejecutarse todo va bien la repuesta de la Postman debería ser un 200 OK.

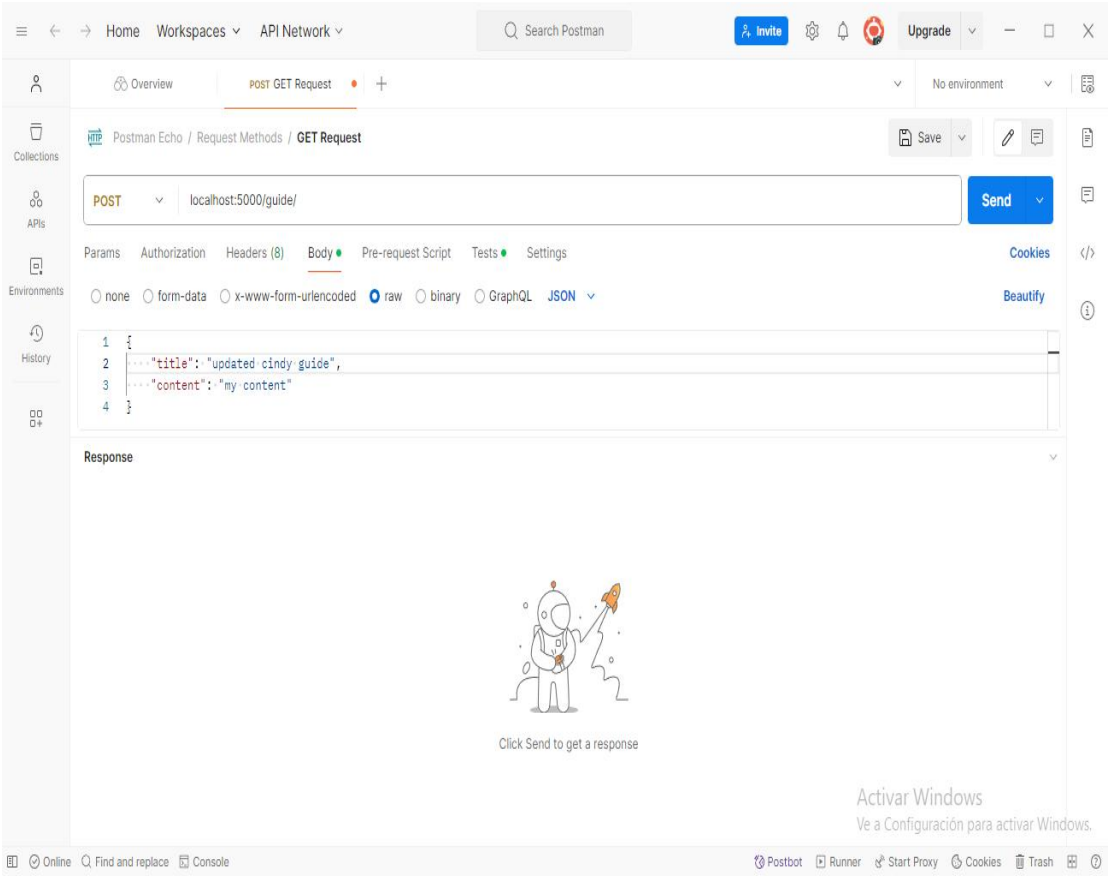

Imagen No. 1

7.¿Qué es el polimorfismo?

R/ La palabra polimorfismo viene del griego poli(muchos) y morfo(forma), en Python hace referencia a que los objetos pueden tomar diferentes formas, es decir objetos de diferentes clases pueden ser accedidos utilizando la misma interfaz y mostrar un comportamiento distinto.

En el siguiente ejemplo tenemos tres clases Perro(), Ave() y Serpiente(), donde cada una tiene un comportamiento distinto y tenemos el método mover con el argumento animal en este método es donde aplicamos el polimorfismo ya que se usara con las tres clases pero con cada una de ellas el comportamiento sera diferente dependiendo del animal que se le pase, ya que cada animal tiene definido un método avanzar diferente, con lo cual cuando al método mover se le pase perro va a funcionar con el método avanzar de la clase perro y cuando se le pase ave va a funcionar con el método avanzar de la clase ave y de igual forma lo hará con serpiente.

Asi el resultado sera que para perro va a imprimir caminar, para ave va a imprimir volar y para serpiente va a imprimir arrastrar.

Código	Salida
class Perro(): def avanzar(self):	Caminar

print("Caminar")
 	
class Ave():
  def avanzar(self):
  print("Volar")
 	
class Serpiente():
  def avanzar(self):
  print("Arrastrar")
 	
def mover(animal):
  animal.avanzar()
 	
perro = Perro()
ave = Ave()
serpiente = Serpiente()
 	
mover(perro)
mover(ave)
mover(serpiente)
	Volar
	Arrastrar

Ejemplo No. 4

8.¿Qué es un método dunder?

R/ los métodos dunder son los métodos que comienzan con __, el nombre del método y termina con __. son métodos privados y protegidos dentro de sus clases. Los métodos dunder se proporcionan directamente desde el lenguaje de programación Python, se pueden usar pero no se puede cambiar de ninguna manera.

Entre los métodos dunder tenemos:

__init__ : Se trata de un inicializador de clase o también conocido como constructor. Cuando una instancia de una clase es creada, el método __init__ es llamado.

Código	Salida

class Prueba(object):
  def __init__(self):
  print('Hola Mundo!')
 	
  def otro_método(self):
  print('Hola otro método!')
 	
a = Prueba()
a.otro_método() #no es llamado automáticamente
	Hola Mundo!
	
	Hola otro método!

Ejemplo No. 5

__string__: dunder string se usa para devolver una representación en cadena de una instancia de clase, se llama para definir como se debe imprimir o cuando se una interpolacion, Asi:

Código	Salida

class Invoice:
  def __init__(self, client, total):
  self.client = client
  self.total = total
 	
  def __str__(self):
  return f'Invoice from {self.client} for {self.total}'
 	
inv = Invoice('Google', 100)
print(str(inv))
	Invoice from Google for 100

Ejemplo No. 6

__repr__: se utiliza para devolver una representación de cadena legible de un objeto. Este método se define dentro de una clase y se llama cuando se usa la función repr() en un objeto de esa clase.

Código	Salida

class Invoice:
  def __init__(self, client, total):
  self.client = client
  self.total = total
 	
  def __repr__(self):
  return f'Invoice <value: client: {self.client}, total: {self.total}>'
 	
inv = Invoice('Google', 100)
print(repr(inv))
	<value: client: Google, total: 100>

Ejemplo No. 7

El método __str__() devuelve una representación de cadena legible para humanos del objeto, mientras que el método __repr__() devuelve una representación de cadena que es suficiente para recrear el objeto.

9. ¿Qué es un decorador de Python?

R/ Son funciones que agregan funcionalidades a otras funciones que recibe como argumento, en resumen nos permite agregar nuevas funcionalidades a otra función sin necesidad de cambiarla. Los decoradores de Python se definen escribiendo el signo @ seguido del nombre del decorador, por ejemplo:

Codigo	Salida

def mi_decorador(funcion):
  def nueva_funcion(a, b):
  print("Se va a llamar")
  c = funcion(a, b)
  print("Se ha llamado")
  return c
  return nueva_funcion
 	
@mi_decorador
def suma(a, b):
  print("Entra en la funcion suma")
  return a + b
	Se va a llamar
	Entra en la funcion suma
	Se ha llamado

Ejemplo No.8	

El código en decorador se ejecutara cuando se llame la función “mi_decorador”, cualquier función que use @mi_decorador tendrá dos print, uno al principio print("Se va a llamar") y otro al final print("Se ha llamado"), dando igual lo que realmente haga la función.

Python tiene un decorador propio que es @property lo que hace es que envuelve la propiedad con la que queremos trabajar,lo que nos dice @property es que los atributos que están allí pueden ser llamados directamente, se usa para getters, setters o deleters. A continuación un ejemplo:

@property para los getter y @client.setter para anular el cliente ya existente en la clase y crear otro google.client = 'Yahoo'.

Nota: Si pones un _ antes del atributo quiere decir que es protegido self._client = client, protegido quiere decir que las clases hijas tendrán acceso a esos datos es uno de los mas comunes.

Codigo	Salida

class Invoice:
 	
  def __init__(self, client):
  self._client = client
 	
  def formater(self):
  return f'{self._client}'
 	
  @property
  def client(self):
  return self._client
 	
  @client.setter
	Google
	
	Yahoo

```
    def client(self, client):
        self._client = client

google = Invoice('Google')
print(google.client)

google.client = 'Yahoo'
print(google.client)
```

Ejemplo No.9