APIs práctica

```
if __name__ == '__main__':
url = 'https://www.google.com.ar/'
response = requests.get(url)
if response.status_code == 200:
content = response.content
file = open('google.html', 'wb')
file.write(content)
file.close()
```

1. Prácticas recomendadas:

```
import requests

# Definir la URL base de la API y el endpoint
base_url = 'http://api.openweathermap.org/data/2.5/weather'
api_key = 'your_api_key'

# Definir parámetros de la solicitud
params = {
    'q': 'London,uk',  # Ciudad y código de país para obtener el clima
    'appid': api_key  # Clave de API para autenticación
}

try:
    # Realizar la solicitud GET a la API
    response = requests.get(base_url, params=params)

    # Verificar si la solicitud fue exitosa (código de estado 200)
    if response.status_code == 200:
        # Obtener los datos de la respuesta en formato JSON
        data = response.json()

        # Acceder a los datos relevantes de la respuesta
        temperature = data['main']['temp']
        weather_description = data['weather'][0]['description']

        # Mostrar los resultados
        print('Temperatura:', temperature)
        print('Descripción del clima:', weather_description)
    else:
        # Mostrar el código de estado en caso de error
        print('Error en la solicitud:', response.status_code)
```

```
except requests.exceptions.RequestException as e:
    # Capturar errores de solicitud
    print('Error de solicitud:', e)
```

En este ejemplo, se utiliza la OpenWeatherMap API para obtener datos del clima de una ciudad específica. Se siguen algunas buenas prácticas:

1. Manejo de errores: Se utiliza un bloque `try-except` para capturar y manejar posibles errores durante la solicitud. Esto ayuda a evitar que la aplicación se bloquee si ocurren errores.
2. Control de límites de tasa (rate limiting): Algunas API pueden tener restricciones sobre el número de solicitudes que se pueden realizar en un cierto período de tiempo. Para respetar estas limitaciones, es importante revisar la documentación de la API para conocer cualquier límite de tasa establecido y ajustar el código en consecuencia.
3. Uso de versiones de API: Algunas API pueden tener múltiples versiones disponibles. Es una buena práctica especificar la versión de la API que se está utilizando en la URL o en los encabezados de la solicitud, para asegurarte de que tu código funcione de manera consistente incluso si la API se actualiza en el futuro.

REST APIs:

REST (Representational State Transfer) es un estilo arquitectónico para el diseño de sistemas distribuidos basados en la web. Se utiliza ampliamente en el desarrollo de API (Application Programming Interfaces) para permitir la comunicación y transferencia de datos entre diferentes aplicaciones o servicios. A continuación, se desarrollan los aspectos clave de las REST API:

1. Estructura de una REST API:
    - URL base: Define la dirección base de la API, a través de la cual se accede a los recursos.
    - Recursos: Representan las entidades que se pueden acceder y manipular mediante la API. Cada recurso tiene una URL única.

- Métodos HTTP: Se utilizan para realizar operaciones sobre los recursos. Los métodos comunes son GET (obtener), POST (crear), PUT (actualizar) y DELETE (eliminar).
- Códigos de estado HTTP: Indican el resultado de una solicitud. Algunos ejemplos son 200 (éxito), 201 (creado), 400 (solicitud incorrecta) y 404 (recurso no encontrado).
- Parámetros: Permiten personalizar las solicitudes mediante la inclusión de información adicional, como filtros, paginación o ordenamiento.

2. Beneficios de las REST API:
    - Flexibilidad: Las REST API permiten una comunicación flexible y escalable entre diferentes sistemas, ya que utilizan estándares web ampliamente adoptados.
    - Independencia de plataforma: Las REST API son independientes de la plataforma y el lenguaje de programación, lo que facilita la integración entre aplicaciones desarrolladas en diferentes tecnologías.
    - Separación entre cliente y servidor: La arquitectura cliente-servidor de las REST API permite una clara separación de responsabilidades y promueve la modularidad y la reutilización del código.
    - Alta interoperabilidad: Al utilizar protocolos web estándar como HTTP y formatos de datos comunes como JSON, las REST API son compatibles con una amplia gama de tecnologías y pueden interactuar con diferentes sistemas.
3. Buenas prácticas al diseñar una REST API:
    - Utilizar URLs semánticas y significativas para los recursos.
    - Emplear los métodos HTTP de manera adecuada y coherente según las operaciones realizadas sobre los recursos.
    - Utilizar códigos de estado HTTP apropiados para indicar el resultado de las solicitudes.

Manipulación de los datos de respuesta para extraer información relevante.

En Python, el manejo de archivos se realiza utilizando la función `open()`. Esta función se utiliza para abrir un archivo en diferentes modos, como lectura, escritura o ambos. Aquí está la sintaxis básica de la función `open()`:

```
open(nombre_archivo, modo)
```

- `nombre_archivo`: Es una cadena de caracteres que especifica el nombre o la ruta del archivo que deseas abrir.
- `modo`: Es una cadena de caracteres que especifica el modo en el que se abrirá el archivo. Algunos modos comunes son:
  - `'r'`: Lectura (por defecto). Abre el archivo en modo de solo lectura.
  - `'w'`: Escritura. Abre el archivo en modo de escritura, y si el archivo no existe, lo crea. Si el archivo ya existe, lo sobrescribe.
  - `'a'`: Agregar. Abre el archivo en modo de escritura, pero no sobrescribe el archivo existente. En su lugar, escribe al final del archivo.
  - `'b'`: Modo binario. Utilizado junto con otros modos para abrir el archivo en modo binario en lugar de modo de texto. Esto es útil para trabajar con archivos que no son de texto, como imágenes o archivos binarios.

En el ejemplo de la página de google, `'google.html'` es el nombre del archivo que deseamos abrir y `'wb'` es el modo en el que se abre el archivo. `'wb'` significa escritura en modo binario, lo cual es útil si estás trabajando con archivos que no son de texto puro, como archivos HTML o imágenes.

```
import requests

if __name__ == '__main__':
  url = 'https://www.google.com.ar/'
  response = requests.get(url)

  if response.status_code == 200:
    content = response.content

    file = open('google.html', 'wb')
    file.write(content)
    file.close()
```

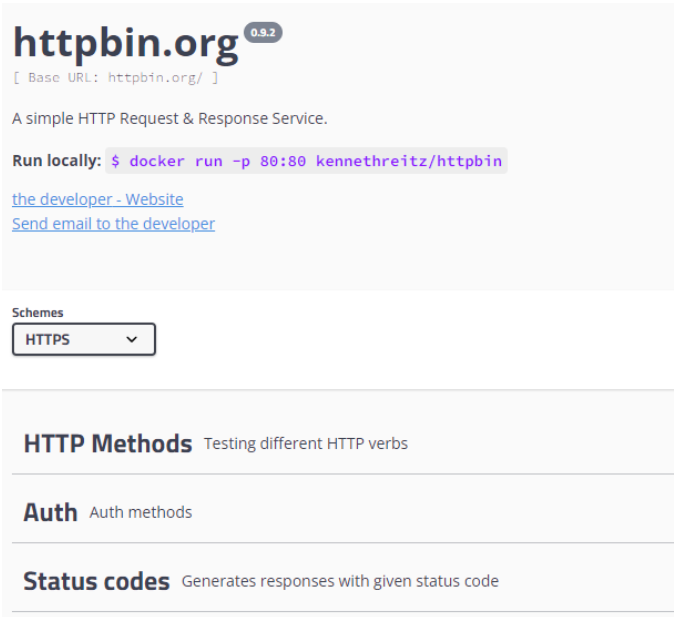
Aquí tienes un ejemplo completo de cómo usar `open()` para abrir un archivo en modo binario y escribir en él:

```
archivo = open('google.html', 'wb')
archivo.write(b'<html><body><h1>Hello, World!</h1></body></html>')
archivo.close()
```

En este ejemplo, hemos abierto el archivo llamado `'google.html'` en modo binario de escritura (`'wb'`). Luego, hemos utilizado el método `write()` para escribir una cadena en el archivo. Nota que hemos utilizado `b''` antes de la cadena para convertirla en una secuencia de bytes, ya que estamos en modo binario. Finalmente, hemos cerrado el archivo utilizando el método `close()`.

Dirección web para prácticar métodos de respuesta:

https://httpbin.org/



Link al colab trabajado en clase:
https://colab.research.google.com/drive/1UqkHM6N8HkTDGb0zUuQzfvhUEsVO0vx-?

usp=sharing

Guía para la creación de REST APIs:

Paso 1: Definir los recursos:

- Identifica los recursos que deseas exponer a través de tu API. Los recursos son entidades o conjuntos de datos que los clientes pueden solicitar o manipular.

Paso 2: Diseñar las rutas (endpoints) de la API:

- Cada recurso debe tener una ruta única que los clientes puedan utilizar para acceder a ellos. Diseña las rutas de acuerdo con las convenciones de REST, utilizando nombres descriptivos y jerarquías si es necesario.

Paso 3: Seleccionar los métodos HTTP adecuados para cada ruta:

- Los métodos HTTP (GET, POST, PUT, DELETE, etc.) se utilizan para indicar la operación que se desea realizar en un recurso. Asigna los métodos adecuados a cada ruta de acuerdo con las acciones permitidas.

Paso 4: Configurar el entorno de desarrollo:

- Elige un lenguaje de programación y un framework adecuado para desarrollar tu API REST. Algunos ejemplos populares incluyen Flask (Python), Express (Node.js), Django (Python) y Ruby on Rails (Ruby). Instala las dependencias necesarias.

Paso 5: Crear las rutas y controladores:

- Define las rutas en tu framework y asigna controladores o funciones para manejar las solicitudes a cada ruta. En los controladores, implementa la lógica de negocio correspondiente para cada operación.

Paso 6: Implementar la lógica de negocio y la interacción con la base de datos:

- Si tu API requiere interactuar con una base de datos u otros servicios externos, implementa la lógica de negocio y las consultas necesarias para recuperar o manipular los datos. Utiliza los modelos y ORM correspondientes si es aplicable.

Paso 7: Validar y procesar los datos de entrada:

- Asegúrate de validar y procesar los datos de entrada de forma adecuada. Realiza comprobaciones de seguridad, validación de formatos y cualquier otra verificación necesaria antes de procesar las solicitudes.

Paso 8: Manejar los códigos de estado y las respuestas:

- Asegúrate de enviar las respuestas adecuadas con los códigos de estado HTTP correspondientes (200, 201, 400, 404, etc.) según el resultado de la solicitud. Formatea las respuestas en el formato adecuado, como JSON o XML.

Paso 9: Documentar la API:

- Documenta tu API de manera clara y concisa. Proporciona detalles sobre las rutas, los parámetros requeridos, los códigos de estado, los formatos de respuesta, la autenticación si es necesario, y cualquier otra información relevante.

Paso 10: Probar y depurar la API:

- Realiza pruebas exhaustivas de tu API utilizando herramientas como Postman o cURL. Verifica que todas las rutas y operaciones funcionen correctamente. Maneja los errores y realiza pruebas de rendimiento si es necesario.

Paso 11: Implementar autenticación y seguridad (opcional):

- Si tu API requiere autenticación o medidas de seguridad adicionales, implementa los mecanismos correspondientes, como tokens de acceso, OAuth, SSL/TLS, entre otros.

Paso 12: Implementar límites de tasa (rate limiting) (opcional):

- Si deseas controlar el número de solicitudes que los clientes pueden realizar a tu API, implementa límites de tasa para evitar abusos o sobrecargas. Define una política de límites de acuerdo a tus necesidades.

  Paso 1: Conoce los límites de la API:
  Investiga la documentación de la API que estás utilizando para comprender los límites de tasa establecidos por el proveedor. Esto puede incluir la cantidad máxima de solicitudes permitidas por unidad de tiempo (por ejemplo, 100 solicitudes por minuto) o restricciones más específicas.

  Paso 2: Establece una estructura de control:
  Determina cómo deseas controlar el ritmo de tus solicitudes. Una opción común es utilizar un temporizador o un contador para asegurarte de que no excedas el límite permitido. Puedes utilizar las herramientas proporcionadas por Python, como el módulo `time`, para implementar esta estructura de control.

Paso 3: Controla el ritmo de las solicitudes:
Antes de enviar una solicitud a la API, verifica si se ha alcanzado el límite de tasa. Si se ha alcanzado, espera un período de tiempo determinado antes de realizar la próxima solicitud. Esto se puede lograr utilizando la función `time.sleep()` para pausar la ejecución de tu código por un cierto intervalo de tiempo.

Aquí tienes un ejemplo básico de cómo implementar límites de tasa en Python:

```
import requests
import time

# Configura los parámetros de límite de tasa
limite_solicitudes = 10  # Máximo de 10 solicitudes por minuto
tiempo_espera = 60  # Espera de 60 segundos entre las solicitudes

# Realiza las solicitudes
for i in range(10):
    # Realiza la solicitud a la API
    response = requests.get("https://api.example.com/endpoint")

    # Verifica si se ha alcanzado el límite de tasa
    if i > 0 and i % limite_solicitudes == 0:
        # Espera el tiempo de espera antes de realizar la próxima solicitud
        time.sleep(tiempo_espera)

    # Procesa la respuesta de la API
    # ...
```

En este ejemplo, se establece un límite de 10 solicitudes por minuto y un tiempo de espera de 60 segundos entre las solicitudes. Después de cada solicitud, se verifica si se ha alcanzado el límite de tasa y, si es así, se espera el tiempo de espera especificado antes de realizar la próxima solicitud.

Es importante tener en cuenta que los límites de tasa pueden variar dependiendo de la API que estés utilizando. Asegúrate de ajustar los valores de límite de tasa y tiempo de espera de acuerdo con las especificaciones proporcionadas por el proveedor de la API.