Q1. Explain GET and POST methods.

Ans: The GET and POST methods are two common HTTP (Hypertext Transfer Protocol) request methods used in web development to send data between a client (such as a web browser) and a server. Here's an explanation of each method:

1. GET Method:

The GET method is used to retrieve or fetch data from a specified resource on the server. When a client makes a GET request, it sends a request to the server to retrieve the content of a particular URL. This method is considered "safe" and "idempotent," meaning it should not have any side effects on the server and can be repeated without causing any additional changes.

GET requests have the following characteristics:

- The data is sent in the URL parameters of the request.

- GET requests can be cached by the browser and other intermediaries.

- GET requests are visible in the browser's address bar, so the parameters are exposed.

GET requests are typically used for operations that do not modify the server's state, such as retrieving a web page, fetching data, or performing searches. For example, when you enter a URL in your web browser, a GET request is made to retrieve the webpage associated with that URL.

2. POST Method:

The POST method is used to submit data to be processed by the server. When a client makes a POST request, it sends data as part of the request body to the server. This method is not considered "safe" or "idempotent" because it can have side effects on the server, such as storing or updating data.

POST requests have the following characteristics:

- The data is sent in the request body, separate from the URL.

- POST requests are not cached by the browser or intermediaries.

- POST requests are not visible in the browser's address bar, so the parameters are not exposed.

POST requests are typically used when you need to create, update, or delete resources on the server. For example, when submitting a form on a web page, the form data is sent to the server using a POST request.

In summary, the GET method is used for retrieving data from a server, while the POST method is used for sending data to a server to be processed and potentially stored. The choice between GET and POST depends on the nature of the operation you want to perform and the requirements of the application you are building.

Q2. Why is request used in Flask?

Ans: In the Flask web framework, the `request` object is used to handle incoming client requests and retrieve data sent by the client. It provides access to the details of the current HTTP request, including the headers, URL parameters, form data, and more. The `request` object is an instance of the `Request` class in Flask, which encapsulates all the information related to the incoming request.

Here are a few reasons why the `request` object is used in Flask:

1. Accessing Request Data: The `request` object allows you to access data sent by the client in various ways. It provides properties and methods to access the request's URL parameters, form data, JSON data, headers, cookies, and more. You can use these features to extract relevant information from the request and use it in your application logic.

2. Handling Different HTTP Methods: The `request` object helps in handling different HTTP methods like GET, POST, PUT, DELETE, etc. It provides attributes like `request.method` to determine the method used in the request. Based on the method, you can execute specific code blocks or perform different actions.

3. Routing and URL Parameters: Flask allows you to define routes with URL parameters, which are dynamic parts of the URL. The `request` object enables you to access these parameters through its `args` attribute. It provides methods like `request.args.get()` to retrieve specific URL parameters passed by the client.

4. File Uploads: If your application involves file uploads, the `request` object helps handle the uploaded files. It provides the `request.files` attribute, which allows you to access the uploaded files, save them to the server, or process them as required.

Overall, the `request` object in Flask is an essential component that facilitates the handling of incoming requests, extracts data sent by the client, and provides access to various aspects of the request. It allows you to build dynamic and interactive web applications by processing and responding to client requests effectively.

Q3. Why is redirect() used in Flask?

Ans: In Flask, the `redirect()` function is used to redirect the client's browser to a different URL. It is a convenient way to direct users to a different route or page within your Flask application or to an external URL.

Here are a few reasons why the `redirect()` function is used in Flask:

1. Route Redirection: When a user visits a specific URL in your Flask application, you may want to redirect them to a different route or page. This could be useful for handling redirects after form submissions, successful login/logout actions, or when certain conditions are met. The `redirect()` function allows you to specify the URL to which the client should be redirected, triggering a new request and rendering the corresponding route or page.

2. Handling Invalid URLs: If a user enters an invalid URL or accesses a route that doesn't exist in your application, you can use the `redirect()` function to redirect them to a custom error page or a predefined route. This helps improve the user experience by gracefully handling such situations and providing meaningful feedback.

3. External URL Redirection: Besides redirecting within your own application, the `redirect()` function can also be used to redirect users to external URLs. For example, you may want to redirect users to a different website, a social media profile, or an external service. The `redirect()` function allows you to specify any valid URL, including external ones, as the target for redirection.

Here's an example usage of `redirect()` in Flask:

```python

from flask import Flask, redirect, url\_for

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

# Redirect to the about page

return redirect(url\_for('about'))

@app.route('/about')

def about():

return 'This is the about page.'

@app.route('/external')

def external():

# Redirect to an external URL

return redirect('https://www.example.com')

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

In the above example, accessing the root URL `'/'` will trigger a redirection to the `'about'` route using `redirect(url\_for('about'))`. The `'about'` route will be rendered, displaying the message "This is the about page." Similarly, accessing `'/external'` will redirect the user's browser to `https://www.example.com`.

Overall, the `redirect()` function in Flask provides a convenient way to control the flow of your application by redirecting users to different routes or external URLs. It enhances the user experience and allows you to handle various redirection scenarios within your Flask application.

Q4. What are templates in Flask? Why is the render\_template() function used?

Ans: In Flask, templates are files that contain HTML markup with placeholders for dynamic content. They are used to separate the presentation logic from the application logic, allowing developers to build web pages that can be dynamically generated based on data from the server.

Here's an explanation of templates in Flask and the purpose of the `render\_template()` function:

Templates in Flask:

- Templates are typically written in HTML, with the ability to include Flask-specific template tags and expressions.

- They allow for dynamic content insertion using placeholders, which are usually represented by double curly braces (`{{ }}`).

- Templates support control structures like loops and conditionals, enabling the generation of different content based on specific conditions or iterating over data.

The `render\_template()` function:

- The `render\_template()` function is used to render a template file and return it as a response to the client's request.

- It takes the name of the template file as an argument and can also accept additional data to be passed to the template.

- The function searches for the specified template in a predefined templates directory within your Flask application.

- It then processes the template, replacing the placeholders with the provided data or evaluating the expressions within the template tags.

- The rendered HTML is sent back to the client's browser as the response.

Here's an example usage of the `render\_template()` function in Flask:

```python

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

# Pass data to the template and render it

return render\_template('index.html', title='Welcome', name='John')

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

```

In the above example, when the user accesses the root URL `'/'`, the `home()` function is triggered. It calls `render\_template()` with the name of the template file `'index.html'` and additional data (`title` and `name`). The template file `index.html` is located in the templates directory. Flask processes the template, replaces the placeholders (`{{ title }}` and `{{ name }}`) with the provided data, and returns the rendered HTML as the response.

By using templates and the `render\_template()` function, Flask allows developers to separate the presentation logic from the application code, promoting a more maintainable and scalable web application architecture. It enables dynamic generation of HTML pages by combining the power of Flask's routing and data processing capabilities with the flexibility of HTML templates.

Q5. Create a simple API. Use Postman to test it. Attach the screenshot of the output in the Jupyter Notebook.

from flask import Flask, jsonify, request

app = Flask(\_\_name\_\_)

@app.route('/api/hello', methods=['GET'])

def hello():

return jsonify({'message': 'Hello, world!'})

@app.route('/api/sum', methods=['POST'])

def sum\_numbers():

data = request.get\_json()

num1 = data['num1']

num2 = data['num2']

result = num1 + num2

return jsonify({'result': result})

if \_\_name\_\_ == '\_\_main\_\_':

app.run()