**Flask -2**

**Assignment**

# Q1. Explain GET and POST methods.

In the context of web development and HTTP (Hypertext Transfer Protocol), GET and POST are two commonly used methods for sending data from a client (such as a web browser) to a server.

1. GET method:

\*The GET method is used to retrieve data from a server.

\*When a client makes a GET request, it appends the data to be sent as query parameters in the URL.

\*The data is visible in the URL, making it less secure for sensitive information.

\*GET requests are idempotent, meaning that making the same request multiple times should have the same effect as making it once. It should not have any side effects on the server or data.

\*GET requests can be cached by the client and intermediaries (like proxies), as they are intended to be safe and repeatable.

\*This method is commonly used for retrieving web pages, images, or other static content from a server.

2. POST method:

\*The POST method is used to send data to a server to create or update a resource.

\*When a client makes a POST request, the data is sent in the body of the HTTP request, not in the URL.

\*The data is not directly visible in the URL, which provides better security for sensitive information.

\*POST requests are not idempotent, meaning that making the same request multiple times may have different effects or create multiple resources.

\*POST requests are not typically cached by clients or intermediaries because they can have side effects.

\*This method is commonly used when submitting forms, sending user input, or performing operations that modify server-side data.

In summary, the GET method is used for retrieving data, while the POST method is used for sending data to create or update resources. GET requests append data in the URL, while POST requests send data in the body of the HTTP request. GET requests are idempotent and can be cached, while POST requests are not idempotent and should not be cached.

# Q2. Why is request used in Flask?

In Flask, the request object is used to handle incoming HTTP requests made by clients to the Flask application. It provides access to various attributes and methods that allow the application to retrieve data from the request, process it, and generate an appropriate response.

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 in the request, such as form data, query parameters, and request headers. You can retrieve this data using attributes and methods provided by the request object, like request.form, request.args, and request.headers.
2. Handling different HTTP methods: The request object helps you handle different HTTP methods like GET, POST, PUT, DELETE, etc. You can check the request method using the request.method attribute and perform specific actions based on the method.
3. File uploads: When handling file uploads, the request object provides access to files sent with the request through the request.files attribute. You can retrieve and process the uploaded files using this attribute.
4. Cookies and sessions: The request object allows you to access cookies sent by the client through the request.cookies attribute. It also provides methods to work with session data, such as request.session or request.session['key'], for storing and retrieving user-specific information.
5. Request context: The request object is part of the Flask request context, which allows you to access request-related information throughout the application. It provides a way to share data across different parts of the application during a single request.

By using the request object, Flask provides a convenient way to handle and process incoming HTTP requests, enabling you to build dynamic and interactive web applications.

# Q3. Why is redirect( ) used in Flask?

In Flask, the redirect() function is used to perform a redirect operation, which means it sends an HTTP redirect response to the client's web browser. It is commonly used in web applications to redirect users to a different URL or route within the application.

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

1. Changing the URL: When a user performs a certain action or accesses a specific URL, you may want to redirect them to a different URL or route within your application. This could be to direct them to a different page, route them to a login page after authentication, or to handle a specific error condition.
2. Handling form submissions: After a user submits a form on a webpage, it is common to redirect them to a different page to display the results or provide feedback. This helps prevent resubmission of the form if the user refreshes the page.
3. Managing authentication and authorization: In web applications that require authentication, the redirect() function can be used to redirect unauthenticated users to a login page or redirect authorized users to a specific page after login.
4. Implementing URL aliases or shortcuts: You can use redirect() to create shorter or more user-friendly URLs for specific routes or resources. For example, you can redirect a long URL to a shorter and more memorable one for marketing purposes.

a simple example that demonstrates the usage of redirect() in Flask:

from flask import Flask, redirect, url\_for

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

@app.route('/')

def index():

# Redirecting to the '/hello' route

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

@app.route('/hello')

def hello():

return 'Hello, World!'

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

app.run()

example, when a user accesses the root URL '/', the index() function is called, which redirects the user to the 'hello' route using redirect(url\_for('hello')). This results in the user being redirected to the hello() function, which returns the "Hello, World!" message.

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

In Flask, templates are files written in HTML or other template languages that allow you to separate the presentation logic from the application logic. Templates are used to generate dynamic HTML content by combining them with data provided by the Flask application.

Flask uses a templating engine called Jinja2, which is a powerful and flexible engine that allows you to build dynamic templates. Templates can contain placeholders, called template variables, which are replaced with actual values when the template is rendered. Template variables are enclosed in double curly braces, like {{ variable\_name }}.

The render\_template() function is used in Flask to render a template and return the corresponding HTML content as the response to a client's request. It takes the name of the template file as its argument, along with optional additional arguments that represent the variables you want to pass to the template.

Here's an example of how render\_template() is used in a Flask route function:

from flask import Flask, render\_template

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

@app.route('/')

def index():

name = 'Riya'

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

In this example, when a user accesses the root URL of the Flask application, the index() function is executed. It calls render\_template() with the template file name 'index.html' and the variable name set to 'Riya'. The function retrieves the content of the index.html template, replaces the {{ name }} placeholder with the value 'Riya', and returns the resulting HTML content as the response.

By using templates and the render\_template() function, you can separate the presentation logic from the application logic, making your Flask application more modular, maintainable, and easier to work with.

# Q5. Create a simple API. Use Postman to test it.

from flask import Flask, jsonify

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

# Sample user data

users = [

{

'id': 1,

'name': 'Vasuki Roi',

'email': 'vasuki@example.com'

},

{

'id': 2,

'name': 'Jane Jadhav',

'email': 'jane@example.com'

}

]

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

def get\_users():

return jsonify(users)

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

app.run(debug=True)