**Flask -1**

Assignment

# Q1. What is Flask Framework? What are the advantages of Flask Framework?

Flask is a popular Python web framework used for building web applications. It is considered a micro-framework because it focuses on simplicity and extensibility, providing only the essentials needed to build a web application.

Advantages of Flask Framework:

1. Lightweight and Flexible: Flask is designed to be minimalistic and lightweight. It does not impose any specific project structure or dependencies, allowing developers the freedom to choose the components they need. This flexibility makes Flask suitable for both small and large-scale projects.
2. Easy to Get Started: Flask has a simple and intuitive API, making it easy for developers, including beginners, to get started quickly. Its syntax is clean and readable, which aids in writing clean and maintainable code.
3. Modular and Extensible: Flask follows the "micro" philosophy, which means it provides only the core functionality. However, it is highly extensible through its modular design. Developers can add additional functionality by using various Flask extensions or third-party libraries based on their project requirements.
4. Integrated Development Server: Flask includes a built-in development server, which allows developers to quickly test their applications without requiring a separate web server setup. This server automatically reloads the application whenever changes are detected, making the development process more efficient.
5. Flask-Werkzeug and Jinja Templating: Flask uses the Werkzeug WSGI toolkit and the Jinja templating engine. Werkzeug provides a solid foundation for handling HTTP requests and responses, while Jinja enables powerful and flexible HTML templating. These components enhance the development experience and help maintain a separation of concerns.
6. Large Community and Ecosystem: Flask has a vibrant and active community of developers. This means there is a wealth of resources, tutorials, and documentation available, making it easier to find help and support when needed. Additionally, Flask integrates well with other popular Python libraries and frameworks, expanding its capabilities and possibilities.
7. Scalability: Although Flask is lightweight, it can handle scaling needs effectively. With the help of extensions and additional libraries, Flask can be used to build complex and high-performance web applications. It allows developers to scale their applications as needed, ensuring they can handle increased traffic and user demands.

Overall, Flask offers a balance between simplicity and extensibility, making it a versatile choice for developing web applications in Python.

# Q2. Create a simple Flask application to display ‘Hello World!!’. Attach the screenshot of the output in Jupyter Notebook.

To create a simple Flask application that displays "Hello World!!" we ,can follow these steps:

1. Install Flask: Run !pip install flask in a code cell to install the Flask framework.
2. Import the necessary modules: Import the Flask class from the flask module and create an instance of it.
3. Define a route and view function: Use the @app.route() decorator to define a route ("/") for the home page. Then, create a view function that returns the "Hello World!!" message.
4. Run the Flask application: Use the app.run() method to run the Flask application.

Here's an example of how we can create this Flask application in a Jupyter Notebook:

from flask import Flask

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

@app.route('/')

def hello\_world():

return 'Hello World!!'

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

app.run()

To attach a screenshot of the output in a Jupyter Notebook, we can follow these steps:

1. Run the Flask application: Execute the code cell containing the Flask application by clicking on it and pressing the "Run" button or using the keyboard shortcut (e.g., Shift+Enter).
2. Take a screenshot: Capture a screenshot of the output. You can use the built-in screenshot functionality of your operating system or a screenshot tool.
3. Attach the screenshot: In the Jupyter Notebook, click on the "Edit" menu, select "Insert Image," and browse to select the screenshot file. Alternatively, you can drag and drop the screenshot file directly into the Jupyter Notebook interface.

# Q3. What is App routing in Flask? Why do we use app routes?

In Flask, app routing refers to the process of mapping URLs to specific functions or view handlers within your application. It allows you to define different routes or endpoints that your application can respond to based on the incoming request's URL.

When a user sends a request to your Flask application, the app routing mechanism determines which function should handle that request based on the URL. Each route is associated with a particular function that is executed when the corresponding URL is accessed. These functions are commonly referred to as view functions or route handlers.

Here's an example of defining a route in Flask:

from flask import Flask

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

@app.route('/')

def index():

return 'Hello, World!'

@app.route('/about')

def about():

return 'This is the about page.'

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

app.run()

n the above example, the @app.route() decorator is used to associate a URL with a specific function. The '/' route is associated with the index() function, which returns the string 'Hello, World!' when accessed. The '/about' route is associated with the about() function, which returns the string 'This is the about page.'.

App routes are used for several reasons:

1. URL Mapping: App routes allow you to map specific URLs to different parts of your application, enabling you to create a logical structure for handling different requests. This helps in organizing and managing the functionality of your application.
2. Request Handling: By defining app routes, you can specify how your application responds to different types of requests. For example, you can have routes for handling GET requests, POST requests, or handling specific types of data.
3. View Functions: App routes allow you to define view functions that generate responses for specific URLs. These view functions can dynamically generate content based on the incoming request parameters or perform other operations required to handle the request.
4. RESTful APIs: If you're building a RESTful API, app routes are crucial for defining the different endpoints that your API supports. Each route can correspond to a specific resource or functionality provided by your API.

# Q4. Create a “/welcome” route to display the welcome message “Welcome to ABC Corporation” and a “/” route to show the following details: Company Name: ABC Corporation Location: India Contact Detail: 999-999-9999

from flask import Flask

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

@app.route('/welcome')

def welcome():

return 'Welcome to ABC Corporation'

@app.route('/')

def company\_details():

details = """

Company Name: ABC Corporation

Location: India

Contact Detail: 999-999-9999

"""

return details

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

app.run()

Output:

Company Name: ABC Corporation

Location: India

Contact Detail: 999-999-9999

# Q5. What function is used in Flask for URL Building? Write a Python code to demonstrate the working of the url\_for() function.

In Flask, the url\_for() function is used for URL building. It generates a URL for the specified endpoint (view function) based on the route patterns defined in your Flask application.

Here's a Python code example to demonstrate the working of the url\_for() function:

from flask import Flask, url\_for

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

@app.route('/')

def index():

return 'Welcome to the homepage!'

@app.route('/user/<username>')

def user\_profile(username):

return f'Welcome, {username}!'

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

with app.test\_request\_context():

# Using the url\_for() function to generate URLs

home\_url = url\_for('index')

user\_url = url\_for('user\_profile', username='Riya')

print('Homepage URL:', home\_url)

print('User Profile URL:', user\_url)

Output:

Homepage URL: /

User Profile URL: /user/Riya

Flask application with two routes. The index() function is associated with the root URL ("/"), and the user\_profile() function is associated with the "/user/<username>" URL pattern.Within the if \_\_name\_\_ == '\_\_main\_\_': block, we create a test request context using app.test\_request\_context(). This allows us to use the url\_for() function outside of a request context. then use url\_for() to generate URLs for the index and user\_profile endpoints, passing any necessary arguments. Finally, we print the generated URLs for demonstration purposes.