***FLASK-1***

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

Flask is a Python-based micro web framework that is designed to make it easy to build web applications. It is called a "micro" framework because it does not require particular tools or libraries and keeps the core of the application simple and lean.

The advantages of using Flask framework are:

1. Lightweight: Flask is a lightweight framework, which means it doesn't come bundled with a lot of unnecessary features. This makes it easy to get started and reduces the overhead of your application.
2. Flexibility: Flask is very flexible and allows developers to build applications in their own way. Flask doesn't dictate how an application should be structured, which gives developers the freedom to build the application the way they want.
3. Easy to learn: Flask has a simple and easy-to-understand syntax, which makes it easy for developers who are new to web development to get started. The Flask documentation is also well-organized and makes it easy to learn.
4. Modular: Flask is designed to be modular, which means that you can add third-party extensions to extend its functionality. There are a lot of extensions available that make it easy to add features like authentication, database integration, and more.
5. Testing: Flask makes it easy to test your application using built-in testing support. Flask provides a testing client that makes it easy to simulate requests to your application and verify the responses.
6. Scalability: Flask is scalable and can be used to build applications of any size. Flask allows you to start small and then scale up as your application grows.

Overall, Flask is a great choice for building web applications due to its simplicity, flexibility, and extensibility.

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

**Jupyter Notebook.**

Here is an example of a simple Flask application that displays "Hello World!!" when you visit the root URL ("/"):

**from flask import Flask**

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

**@app.route('/')**

**def hello\_world():**

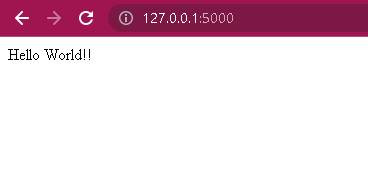
**return 'Hello World!!'**

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

**app.run()**

This code imports the Flask class from the flask module, creates a new instance of the Flask class, and defines a route that maps to the root URL ("/"). When a user visits the root URL, the hello\_world function is called and returns the message "Hello World!!". Finally, the app.run() method starts the Flask application.

You can save this code in a Python file (e.g. app.py) and run it in your terminal or command prompt using the command:



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

In Flask, app routing refers to the process of mapping URL patterns to view functions. When a user requests a URL, Flask uses the routing information to determine which view function should handle the request and generate a response.

App routes are used in Flask to define URL patterns and map them to view functions. An app route is created using a decorator (@app.route()) that is applied to a view function. The decorator specifies the URL pattern that should map to the view function.

For example, here is a simple app route that maps the root URL ("/") to a view function named hello\_world:

**from flask import Flask**

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

**@app.route('/')**

**def hello\_world():**

**return 'Hello World!!'**

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

**app.run()**

In this example, the @app.route('/') decorator specifies that the root URL ("/") should map to the hello\_world function. When a user visits the root URL in their web browser, Flask calls the hello\_world function and returns the message "Hello World!!".

App routing in Flask is useful because it allows you to define the structure of your application's URLs and map them to specific functions that generate the appropriate response. This makes it easy to create dynamic web applications with multiple pages and URL patterns.

Additionally, app routing helps to keep your code organized and maintainable. By separating the different views of your application into individual functions with their own URL patterns, you can more easily make changes to specific parts of your application without affecting the rest of it.

***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():**

**return 'Company Name: ABC Corporation<br>Location: India<br>Contact Detail: 999-999-9999'**

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

**app.run()**

**Q5. What function is used in Flask for URL Building? Write a Python code to demonstrate the working of the**

**url\_for() function.**

The url\_for() function in Flask is used for URL building. It takes the name of a Flask view function as the first argument and generates a URL based on the rules defined for that view.

Here is an example of how to use the url\_for() function in Flask:

**from flask import Flask, url\_for**

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

**@app.route('/')**

**def home():**

**return 'This is the homepage.'**

**@app.route('/about')**

**def about():**

**return 'This is the about page.'**

**@app.route('/contact')**

**def contact():**

**return 'This is the contact page.'**

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

**with app.test\_request\_context():**

**print(url\_for('home'))**

**print(url\_for('about'))**

**print(url\_for('contact'))**

In this code, we have defined three routes using the @app.route() decorator: home, about, and contact. The url\_for() function is used to generate URLs for these views.

When the Flask application is running, the url\_for() function will generate URLs based on the rules defined for each view. For example, url\_for('home') will generate a URL for the home view, which in this case is simply "/", and url\_for('about') will generate a URL for the about view, which is "/about".

When we run the code, it will print the generated URLs to the console:

**/**

**/about**

**/contact**

Note that we have wrapped the calls to url\_for() inside a with app.test\_request\_context() block. This is because the url\_for() function requires a Flask application context to work properly. By using the test\_request\_context() function, we can create a temporary application context for testing purposes.