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

Flask is a lightweight web framework written in Python. It is designed to be simple, yet powerful, and is often referred to as a microframework because it does not require or impose a particular set of tools or libraries. Flask provides a solid foundation for building web applications and APIs, allowing developers to focus on writing their application logic rather than dealing with the complexities of the underlying infrastructure.

Here are some advantages of using Flask:

1. Simplicity: Flask is known for its simplicity and minimalistic design. It has a small core framework and offers only the essential features needed for web development. This makes it easy to learn and use, especially for beginners or those who prefer a more lightweight framework.

2. Flexibility: Flask provides a flexible architecture that allows developers to choose the components they want to use. It does not force any specific database, templating engine, or form validation library, giving developers the freedom to integrate the tools they prefer.

3. Extensibility: Flask is highly extensible and allows developers to add additional functionalities through third-party libraries or extensions. There is a wide range of Flask extensions available for tasks like database integration, authentication, form handling, and more. This modular approach enables developers to customize their applications based on their speciic needs.

4. Python integration: Being written in Python, Flask leverages the power and simplicity of the Python language. Developers can utilize the vast ecosystem of Python libraries and tools to enhance their Flask applications. Flask also supports the use of popular Python features and idioms, making it a natural choice for Python developers.

5. Testing and debugging: Flask provides built-in support for testing, which makes it easier to write unit tests for your application. Additionally, Flask's debugging features allow developers to easily identify and fix issues during development, providing a smoother debugging experience.

6. Documentation and community: Flask has comprehensive and well-organized documentation, making it easy for developers to get started and find solutions to common problems. It also has a large and active community, with numerous tutorials, blog posts, and open-source projects available. The community support ensures that developers can find help and resources whenever they need assistance.

Overall, Flask's simplicity, flexibility, extensibility, Python integration, testing/debugging capabilities, and supportive community make it a popular choice for developing web applications and APIs.

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

from flask import Flask

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

@app.route('/')

def hello\_world():

return 'Hello World!!'

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

app.run()

Save this code in a Python file, for example, app.py. Make sure you have Flask installed (pip install flask) and run the file using python app.py. Flask will start a development server, and you should see output indicating that the application is running. You can then access the application by opening a web browser and navigating to http://localhost:5000/, where you will see the "Hello World!!" message displayed.

This is a basic example that demonstrates the fundamental structure of a Flask application. The @app.route('/') decorator associates the hello\_world function with the root URL ("/"), and when accessed, it returns the "Hello World!!" string. You can modify and expand upon this example to create more complex Flask applications.

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

In Flask, app routing refers to the process of associating a URL endpoint with a specific function or view in your application. It allows you to map different URLs to different functions, enabling your application to handle various requests and provide appropriate responses.

The @app.route() decorator is used in Flask to define routes. It is applied to a function and specifies the URL pattern that should trigger the execution of that function. The URL pattern can include dynamic elements, such as variable parts or placeholders, which can be extracted and used in the function.

Here's an example to illustrate app routing in Flask:

Python Code

from flask import Flask

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

@app.route('/')

def home():

return 'This is the home page.'

@app.route('/about')

def about():

return 'This is the about page.'

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

def profile(username):

return f'This is the profile page of {username}.'

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

app.run()

In this example, we have three different routes defined:

The '/' route is associated with the home function. When a user visits the root URL (e.g., http://localhost:5000/), the home function is executed, and the message "This is the home page." is returned.

The '/about' route is associated with the about function. When a user visits the /about URL (e.g., http://localhost:5000/about), the about function is executed, and the message "This is the about page." is returned.

The '/user/<username>' route is associated with the profile function. It includes a dynamic element <username> which acts as a placeholder for any value provided in the URL. For example, when a user visits /user/johndoe, the profile function is executed, and the message "This is the profile page of johndoe." is returned.

App routing is crucial because it allows you to define the behavior of your application based on different URLs. It enables you to create different views, handle different types of requests (e.g., GET, POST), and pass dynamic values from the URL to your functions. By using app routes effectively, you can create a structured and organized web application that responds to user requests appropriately.

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

Ans: Certainly! Here's an updated Flask application with the `/welcome` and `/` routes to display the requested details:

```python

from flask import Flask

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

@app.route('/welcome')

def welcome():

return 'Welcome to ABC Corporation'

@app.route('/')

def company\_details():

company\_name = 'ABC Corporation'

location = 'India'

contact\_detail = '999-999-9999'

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

app.run()

```

In this updated example:

- The `/welcome` route is associated with the `welcome` function. When a user visits the `/welcome` URL (e.g., `http://localhost:5000/welcome`), the `welcome` function is executed, and the message "Welcome to ABC Corporation" is returned.

- The `/` route is associated with the `company\_details` function. When a user visits the root URL (e.g., `http://localhost:5000/`), the `company\_details` function is executed, and the details of the company are displayed. The company name, location, and contact details are stored in variables and rendered as an HTML-formatted string using line breaks (`<br>`).

You can run this Flask application, and when you visit `http://localhost:5000/welcome`, you will see the welcome message "Welcome to ABC Corporation". And when you visit `http://localhost:5000/`, you will see the company details displayed as follows:

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

Ans: In Flask, the `url\_for()` function is used for URL building. It is a helpful utility function provided by Flask that generates a URL for a specific function or endpoint based on its name. This function takes the name of the function or endpoint as an argument and returns the corresponding URL.

Here's an example to demonstrate the working of the `url\_for()` function:

```python

from flask import Flask, url\_for

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

@app.route('/')

def home():

return 'This is the home page.'

@app.route('/about')

def about():

return 'This is the about page.'

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

def profile(username):

return f'This is the profile page of {username}.'

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

with app.test\_request\_context():

print(url\_for('home')) # Output: /

print(url\_for('about')) # Output: /about

print(url\_for('profile', username='johndoe')) # Output: /user/johndoe

```

In this example, we import the `url\_for()` function from Flask and define three routes (`home`, `about`, and `profile`) as shown in the previous examples.

Within the `if \_\_name\_\_ == '\_\_main\_\_':` block, we use `app.test\_request\_context()` to create a temporary request context, which allows us to access the `url\_for()` function without running the actual Flask application. This is useful for testing and demonstration purposes.

Inside the `print()` statements, we pass the function or endpoint names as arguments to `url\_for()` to generate the corresponding URLs. For example, `url\_for('home')` returns `/`, `url\_for('about')` returns `/about`, and `url\_for('profile', username='johndoe')` returns `/user/johndoe`.

When you run this code, it will print the generated URLs to the console. Note that the actual URLs may vary depending on your Flask application's configuration and URL structure.

The `url\_for()` function is handy when you want to generate URLs dynamically, especially when dealing with routes that have dynamic elements or when you want to avoid hard-coding URLs in your templates or code.