**1. What is Flask?**

Flask is a micro web framework for Python designed for building web applications quickly and with minimal overhead. It is classified as a "micro-framework" because it doesn't require particular tools or libraries, and it keeps the core simple but extensible.

* **Lightweight and Simple**: Focuses on keeping the core simple, with optional add-ons.
* **Extensible**: You can add plugins or libraries for more functionality (e.g., database support, form handling).
* **WSGI-compliant**: Flask is built to be compatible with the Web Server Gateway Interface (WSGI), a standard for Python web applications.

**2. Setting Up Flask**

To start using Flask, you need to install it using pip:

pip install flask

Once installed, you can create a basic Flask app:

from flask import Flask

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

@app.route('/')

def home():

return 'Hello, World!'

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

app.run(debug=True)

**3. Flask Routing**

* **Route Decorator**: Routes in Flask map URLs to functions. The @app.route() decorator is used to link a function to a URL.
* **Dynamic Routes**: You can define dynamic URL parameters using <parameter> syntax.

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

def show\_user(username):

return f'User: {username}'

**4. Flask HTTP Methods**

Flask routes can handle different HTTP methods (GET, POST, etc.). Use the methods parameter in the route to specify the methods.

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

def submit():

return 'Form submitted!'

* **GET**: Default method, retrieves data.
* **POST**: Sends data to be processed.
* **PUT**: Updates existing data.
* **DELETE**: Deletes data.

**5. Flask Templates (Jinja2)**

Flask uses **Jinja2** templating engine to render HTML dynamically.

* **Rendering Templates**: Use render\_template() to render HTML files with dynamic content.

from flask import render\_template

@app.route('/profile')

def profile():

return render\_template('profile.html', name="John")

In profile.html, you can use Jinja2 syntax:

<h1>Welcome, {{ name }}!</h1>

**6. Flask Static Files**

Flask provides a folder named static for serving static files (e.g., images, CSS, JavaScript). Files placed inside the static folder can be accessed via /static/filename.

Example:

<img src="{{ url\_for('static', filename='image.png') }}" alt="image">

**7. Flask Forms**

You can handle HTML forms in Flask by using the request object to retrieve form data.

* **GET vs. POST**: Form data is sent via GET (for simple data) or POST (for sensitive data).
* **Handling POST Requests**:

from flask import request

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

def submit():

name = request.form['name']

return f"Hello, {name}!"

**8. Flask Sessions**

Flask uses cookies to store session data on the client-side, ensuring data is preserved across requests. To use sessions, you need to set a secret key.

app.secret\_key = 'your\_secret\_key'

To store and retrieve session data:

from flask import session

@app.route('/set\_session')

def set\_session():

session['username'] = 'john'

return 'Session is set!'

@app.route('/get\_session')

def get\_session():

return f"Logged in as: {session.get('username')}"

**9. Flask Redirects and URL For**

Flask provides redirect() to redirect users to another route and url\_for() to dynamically generate URLs.

* **Redirect**: Redirects to another URL.

from flask import redirect, url\_for

@app.route('/login')

def login():

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

**10. Flask Error Handling**

You can define custom error pages for different HTTP error codes (like 404 or 500).

@app.errorhandler(404)

def not\_found(error):

return 'Page not found!', 404

app.register\_blueprint(mod, url\_prefix='/mod')

**11. Flask Database Integration**

Flask doesn’t include a database by default, but it can be easily integrated with databases like **SQLite**, **PostgreSQL**, or **MySQL** using libraries such as **SQLAlchemy** or **Flask-SQLAlchemy**.

**Flask-SQLAlchemy Example**:

pip install flask-sqlalchemy

from flask\_sqlalchemy import SQLAlchemy

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///mydatabase.db'

db = SQLAlchemy(app)

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(80), unique=True, nullable=False)

@app.route('/add\_user')

def add\_user():

user = User(username="john")

db.session.add(user)

db.session.commit()

return f'User {user.username} added!'