Python - Flask

Introduction to Flask

- **Flask** is a lightweight and flexible Python web framework.
- It allows developers to build web applications and REST APIs quickly.
- Known as a **microframework** because it doesn't include unnecessary tools.

✅ Simple
✅ Fast
✅ Scalable

Why Flask?

- Minimal setup required
- Built-in development server
- Easy routing system
- Jinja2 templating support
- Compatible with databases (SQLite, MySQL)
- Ideal for microservices and APIs

Installing Flask

```
pip install flask
```

✅ Flask installs automatically with its dependencies.
You can verify installation by running:

```
python -m flask --version
```

First Flask Application

Program: `app.py`

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Hello, Flask!"

if __name__ == '__main__':
    app.run(debug=True)
```

Explanation:

- `Flask(__name__)` → Creates the Flask app.
- `@app.route('/')` → Defines route for home page.
- `app.run()` → Starts the development server.

Running the Flask App

- Save the file as `app.py`
- Run the following command:

```
python app.py
```

- Visit **http://localhost:5000/**
  ✅ Output → *Hello, Flask!*

Flask Application Structure

```
project/
 |
 ├──── app.py
 ├──── templates/
 |     ├──── index.html
 |     ├──── about.html
 ├──── static/
 |     ├──── style.css
 |     ├──── script.js
 └──── database.db
```

Explanation:

- `templates/` → HTML files (Jinja2 templates)
- `static/` → CSS, JS, images
- `app.py` → main application

Routing in Flask

```
@app.route('/')
def home():
    return "Welcome Home"

@app.route('/about')
def about():
    return "About Page"
```

Explanation:

- Each `@app.route()` defines a unique URL path.
- Flask maps URLs to Python functions.

URL Parameters

```
@app.route('/user/<name>')
def user(name):
    return f"Hello, {name}!"
```

✅ Example:

Visiting `/user/Praveen` → *Hello, Praveen!*

Using HTML Templates

File: `templates/index.html`

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Welcome to Flask!</h1>
    <p>Hello, {{ username }}</p>
  </body>
</html>
```

Python Code:

```
from flask import render_template

@app.route('/welcome/<username>')
def welcome(username):
    return render_template('index.html', username=username)
```

Passing Data to Template

```
@app.route('/data')
def data():
    users = ['John', 'Mary', 'Peter']
    return render_template('users.html', users=users)
```

templates/users.html

```
<h2>User List</h2>
<ul>
  {% for user in users %}
    <li>{{ user }}</li>
  {% endfor %}
</ul>
```

✅ {% ... %} → Control statements
✅ {{ ... }} → Display variables

Introduction to Jinja2

- **Jinja2** is a **templating engine** for Python, used mainly with **Flask**.
- It allows you to write **HTML pages that can dynamically display data** from Python code.
- Flask uses Jinja2 by default for rendering templates.

✅ Templating Engine
✅ Dynamic HTML Generation
✅ Easy Integration with Flask

Why Jinja2?

- Simplifies HTML page creation with dynamic content.
- Separates **business logic (Python)** from **presentation (HTML)**.
- Prevents repetitive HTML code using **templates and inheritance**.
- Enables variables, loops, conditions inside HTML.

How Jinja2 Works

Python (Backend)
↓
Data passed via render_template()
↓
Jinja2 Template (HTML + Jinja2 syntax)
↓
Dynamic HTML Output to Browser

✅ Flask passes data →
✅ Jinja2 processes it →
✅ Output = Dynamic webpage

Setting up Jinja2 with Flask

Folder Structure:

```
project/
 |
 ├──── app.py
 └──── templates/
    └──── index.html
```

Python Code (app.py)

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('index.html', username="Praveen")

if __name__ == '__main__':
    app.run(debug=True)
```

HTML Template (templates/index.html)

```
<h1>Welcome {{ username }}!</h1>
```

✅ Output → *Welcome Praveen!*

Jinja2 Syntax Overview

Syntax	Purpose	Example
`{{ ... }}`	Output variable	`{{ name }}`
`{% ... %}`	Logic/Control statements	`{% for user in users %}`
`{# ... #}`	Comment	`{# This is ignored #}`

Using Variables

Python Code

```
@app.route('/user')
def user():
    return render_template('user.html', name="Alice", age=25)
```

Template (`user.html`)

```
<p>Name: {{ name }}</p>
<p>Age: {{ age }}</p>
```

✅ Output:

```
Name: Alice
Age: 25
```

Using Loops

Python Code

```
@app.route('/users')
def users():
    user_list = ['John', 'Mary', 'Steve']
    return render_template('users.html', users=user_list)
```

Template (`users.html`)

```
<h2>User List</h2>
<ul>
  {% for user in users %}
    <li>{{ user }}</li>
  {% endfor %}
</ul>
```

✅ Output:

```
User List
• John
• Mary
• Steve
```

Flask + Jinja2 Example: Display Product List

Flask Application (`app.py`)

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def products():
    product_list = [
        {'id': 1, 'name': 'Laptop', 'price': 55000},
        {'id': 2, 'name': 'Mobile', 'price': 25000},
        {'id': 3, 'name': 'Tablet', 'price': 30000},
        {'id': 4, 'name': 'Smart Watch', 'price': 12000}
    ]
    return render_template('products.html', products=product_list)

if __name__ == '__main__':
    app.run(debug=True)
```

Jinja2 Template (`templates/products.html`)

```
<!DOCTYPE html>
<html>
 <head>
  <title>Product List</title>
   </head>
 <body>
  <h1>Available Products</h1>
  <table>
   <tr>
    <th>ID</th>
    <th>Product Name</th>
    <th>Price (₹)</th>
   </tr>
   {% for p in products %}
   <tr>
    <td>{{ p.id }}</td>
    <td>{{ p.name }}</td>
    <td>{{ p.price }}</td>
   </tr>
   {% else %}
   <tr>
    <td colspan="3">No products available</td>
   </tr>
   {% endfor %}
  </table>
 </body>
</html>
```

Using Conditional Statements

Python Code

```
@app.route('/check')
def check():
    return render_template('check.html', marks=85)
```

Template (`check.html`)

```
{% if marks >= 50 %}
  <p>Result: Pass</p>
{% else %}
  <p>Result: Fail</p>
{% endif %}
```

✅ Output → *Result: Pass*

Using Filters

Filters modify variable output inside templates.

Filter	Description	Example
`	upper`	Converts to uppercase
`	lower`	Converts to lowercase
`	title`	Converts to title case
`	length`	Returns length
`	default('N/A')`	Default value

Example:

```
<p>{{ "flask jinja"|upper }}</p>
```

✅ Output: `FLASK JINJA`

Loop with Index and Filters

```
<ul>
  {% for item in items %}
    <li>{{ loop.index }} - {{ item|title }}</li>
  {% endfor %}
</ul>
```

Features:

- `loop.index` → Current index (1-based).
- `loop.first`, `loop.last` → Boolean flags.

Template Inheritance

Purpose: To avoid code repetition by defining a base layout.

base.html

```
<!DOCTYPE html>
<html>
 <head><title>My Website</title></head>
 <body>
  <header><h1>Welcome to My Site</h1></header>
  <hr>
  {% block content %}{% endblock %}
  <hr>
  <footer>Copyright 2025</footer>
 </body>
</html>
```

home.html

```
{% extends 'base.html' %}
{% block content %}
 <h2>This is the Home Page</h2>
{% endblock %}
```

✅ Output → *Dynamic page inside common layout.*

Including Templates

```
<!-- main.html -->
<html>
  <body>
    {% include 'header.html' %}
    <h2>Main Content</h2>
    {% include 'footer.html' %}
  </body>
</html>
```

✅ `include` allows modular template design.