■ Python Flask Tutorial - Beginner to Advanced

1. Introduction to Flask

Flask is a lightweight and flexible web framework written in Python. It is often called a "micro-framework" because it does not require particular tools or libraries. It provides the essentials to build web applications and APIs, while allowing developers to add extensions when needed. Key Features: - Lightweight and modular design. - Built-in development server and debugger. - Jinja2 template engine for dynamic HTML rendering. - RESTful request dispatching for building APIs. - Large ecosystem of extensions (authentication, database, etc.).

2. Installation

To install Flask, use pip:

```
pip install flask
```

Verify installation by running:

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

3. Your First Flask Application

```
from flask import Flask

app = Flask(__name__)

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

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

4. Routing in Flask

Flask uses decorators to define routes.

Example:

```
@app.route("/about")
def about():
    return "This is the About Page."
```

Dynamic routes:

```
@app.route("/user/<username>")
```

```
def user_profile(username):
    return f"Welcome, {username}!"
```

5. Using Templates (Jinja2)

Flask uses Jinja2 for templates.

Example: Create a file templates/index.html

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Hello {{ name }}!</h1>
  </body>
</html>
```

In app.py:

```
from flask import render_template

@app.route("/hello/<name>")
def hello(name):
    return render_template("index.html", name=name)
```

6. Handling Forms

Flask can handle form submissions using POST.

HTML (templates/form.html):

```
<form method="POST">
  <input type="text" name="username">
  <input type="submit" value="Submit">
</form>
```

Flask Code:

```
from flask import request

@app.route("/submit", methods=["GET", "POST"])
def submit():
    if request.method == "POST":
        username = request.form["username"]
        return f"Hello {username}"
    return render_template("form.html")
```

7. Building REST APIs

```
from flask import Flask, jsonify
app = Flask(__name__)
```

```
@app.route("/api/data", methods=["GET"])
def get_data():
    return jsonify({"message": "Hello API", "status": "success"})

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

8. Working with Databases (SQLAlchemy)

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///test.db"
db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100))

@app.route("/add/<name>")
def add_user(name):
    user = User(name=name)
    db.session.add(user)
    db.session.commit()
    return f"User {name} added!"

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

9. Project Structure Best Practices

Recommended Flask Project Structure:

```
my_flask_app/
■■■ app.py
■■■ requirements.txt
■■■ static/          # CSS, JS, images
■■■ templates/       # HTML files
■■■ instance/        # Config, database
■■■ models.py        # Database models
■■■ routes.py        # Application routes
■■■ config.py        # Configuration settings
```

10. Conclusion & Next Steps

Flask is simple yet powerful for creating web apps and APIs. You can start small and scale your application by adding extensions when needed. Next Steps: - Learn about Flask Blueprints for modular apps. - Use Flask-Login for authentication. - Deploy apps with Gunicorn and Nginx or use cloud services (Heroku, AWS, etc.).