```
/*****************************************************************
*   Flask - COMPLETE MASTERY GUIDE (10-PAGE FORMAT)              *
*   GOAL: 100% knowledge, 10-page structure, readable, no waste.    *
*   ORDER: Most Used First -> Advanced Topics                    *
*****************************************************************/

/*
```

PAGE 1
INTRODUCTION & CORE CONCEPTS

```
*/
//--- WHAT IS FLASK? ---
// Flask ek micro web framework hai Python me. "Micro" ka matlab hai ki ye
simple aur lightweight hai.
// Aap features (jaise database, forms) extensions ke through add karte hain.
// Key Features: Simple, flexible, easy to start, lots of extensions.

//--- INSTALLATION ---
// Flask install karein.
// Terminal Command:
pip install Flask

//--- YOUR FIRST FLASK APP (app.py) ---
// 1. Flask class import karein.
from flask import Flask

// 2. Flask ka instance banayein. `__name__` se Flask ko pata chalta hai ki
resources kahan hain.
app = Flask(__name__)

// 3. Ek "route" decorator banayein jo URL ko function se jodta hai.
@app.route('/')
// 4. Ek "view function" likhein jo response (HTML) return karta hai.
def hello_world():
    return '<h1>Hello, World!</h1>'

//--- RUNNING THE DEVELOPMENT SERVER ---
// Method 1: `flask` command (Recommended)
// Terminal:
// set FLASK_APP=app.py  (On Windows)
// export FLASK_APP=app.py (On Mac/Linux)
// flask run
//
// Method 2: Python script
if __name__ == '__main__':
    app.run(debug=True) // `debug=True` se server auto-restarts aur debugger on
hota hai.

/*
```

PAGE 2
ROUTING & URLS

```
*/
//--- BASIC ROUTING ---
// `app.route()` decorator URL ko view function se link karta hai.
@app.route('/about')
def about():
    return 'This is the about page.'

//--- DYNAMIC ROUTES (VARIABLE RULES) ---
// URL se variable parts capture karein.
@app.route('/user/<username>')
```

```
def show_user_profile(username):
    return f'User: {username}'

//--- CONVERTERS ---
// Aap variable ka type specify kar sakte hain.
@app.route('/post/<int:post_id>')
def show_post(post_id):
    return f'Post ID: {post_id}' // post_id ek integer hoga.
// Common Converters: string (default), int, float, path (accepts slashes),
uuid.

//--- HTTP METHODS ---
// By default, routes sirf GET requests handle karte hain.
from flask import request
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return 'Handling POST request'
    else:
        return 'Showing login form'

//--- URL BUILDING (url_for) ---
// Hardcode karne ke bajaye URLs dynamically generate karein.
from flask import url_for
@app.route('/admin')
def admin_page():
    # `url_for` function ka naam leta hai, URL nahi.
    # Example: url_for('show_user_profile', username='John Doe') ->
/user/John%20Doe
    return 'Admin Page'

/*
```



```
*/
// Flask, Jinja2 template engine use karta hai HTML generate karne ke liye.
// By default, Flask `templates` naam ke folder me templates dhoondhta hai.
// Project Structure:
// /app.py
// /templates/
//   - index.html
//   - base.html

//--- RENDERING TEMPLATES ---
from flask import render_template

@app.route('/hello/<name>')
def hello(name):
    # `render_template` template file ko render karta hai.
    # Aap variables ko keyword arguments ke roop me pass kar sakte hain.
    return render_template('index.html', user_name=name)

//--- JINJA2 SYNTAX IN HTML (index.html) ---
// Variables: {{ user_name }}
// Statements: {% for item in items %} ... {% endfor %}
/*
<!doctype html>
<title>Hello from Flask</title>
<h1>Hello, {{ user_name }}!</h1>
*/

//--- TEMPLATE INHERITANCE ---
```

```
// Code repeat se bachne ke liye.
// `base.html` (Parent Template)
/*
<!doctype html>
<html>
  <head><title>{% block title %}{% endblock %}</title></head>
  <body>
    <div id="content">{% block content %}{% endblock %}</div>
  </body>
</html>
*/

// `index.html` (Child Template)
/*
{% extends "base.html" %}
{% block title %}Home Page{% endblock %}
{% block content %}
  <h1>Welcome to the Home Page!</h1>
{% endblock %}
*/

/*
```




PAGE 4
REQUEST & RESPONSE

```
*/
//--- THE REQUEST OBJECT ---
// Current request ka data access karne ke liye `request` object use karein.
from flask import request

@app.route('/search')
def search():
    # Query Parameters (e.g., /search?q=flask)
    query = request.args.get('q', 'default_value')
    return f'Searching for: {query}'

@app.route('/login', methods=['POST'])
def login_post():
    # Form Data
    username = request.form['username']
    password = request.form['password']
    return f'Username is {username}'

//--- FILE UPLOADS ---
// HTML form me `enctype="multipart/form-data"` hona chahiye.
@app.route('/upload', methods=['POST'])
def upload_file():
    uploaded_file = request.files['file']
    if uploaded_file.filename != '':
        uploaded_file.save(f'uploads/{uploaded_file.filename}')
        return 'File uploaded successfully'
    return 'No file selected'

//--- COOKIES ---
from flask import make_response, request
@app.route('/set_cookie')
def set_cookie():
    resp = make_response('Cookie has been set')
    resp.set_cookie('userID', 'flask_user')
    return resp

@app.route('/get_cookie')
def get_cookie():
```

```
    user_id = request.cookies.get('userID')
    return f'User ID from cookie is: {user_id}'

//--- REDIRECTS & ERRORS ---
from flask import redirect, abort
@app.route('/old_url')
def old_url():
    return redirect(url_for('hello_world')) // `redirect` user ko doosre URL par
bhejta hai.

@app.route('/user_page/<int:user_id>')
def user_page(user_id):
    if user_id == 0:
        abort(404) // 404 Not Found error generate karein.
    return f'Page for user {user_id}'

/*
```



```
*/
// Flask-WTF extension se forms handle karna aasan ho jaata hai.
// Installation: pip install Flask-WTF

//--- SETUP ---
app.config['SECRET_KEY'] = 'a-very-secret-key' // CSRF protection ke liye
zaroori.

//--- CREATING A FORM CLASS ---
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, Email

class LoginForm(FlaskForm):
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Log In')

//--- VIEW FUNCTION ---
@app.route('/login_wtf', methods=['GET', 'POST'])
def login_wtf():
    form = LoginForm()
    if form.validate_on_submit(): // POST request aur valid data check karta
hai.
        # Process data
        return f'Logged in with email: {form.email.data}'
    return render_template('login.html', form=form)

//--- RENDERING FORM IN TEMPLATE (login.html) ---
/*
<form method="POST" novalidate>
    {{ form.hidden_tag() }} <!-- CSRF Token -->

    <p>
        {{ form.email.label }}<br>
        {{ form.email(size=32) }}
        {% for error in form.email.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.password.label }}<br>
        {{ form.password(size=32) }}
```

```
    </p>
    <p>{{ form.submit() }}</p>
</form>
*/

/*
```

PAGE 6
DATABASES (FLASK-SQLALCHEMY)

```
*/
// Flask-SQLAlchemy extension se database operations aasan ho jaate hain.
// Installation: pip install Flask-SQLAlchemy

//--- SETUP ---
from flask_sqlalchemy import SQLAlchemy
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///test.db' // Database
location.
db = SQLAlchemy(app)

//--- DEFINING MODELS ---
// Model ek Python class hai jo database table ko represent karti hai.
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)

//--- CREATING TABLES ---
// Python shell me tables create karein.
// from app import db
// db.create_all()

//--- ADDING DATA ---
@app.route('/add_user')
def add_user():
    new_user = User(username='john', email='john@example.com')
    db.session.add(new_user)
    db.session.commit()
    return 'User added!'

//--- QUERYING DATA ---
@app.route('/users')
def list_users():
    all_users = User.query.all() // Sabhi users get karein.
    first_user = User.query.first() // Pehla user get karein.
    user_by_id = User.query.get(1) // ID se get karein.
    filtered_user = User.query.filter_by(username='john').first() // Filter
karke get karein.

    output = ""
    for user in all_users:
        output += f'ID: {user.id}, Username: {user.username}<br>'
    return output

/*
```

PAGE 7
BLUEPRINTS & APP STRUCTURE

```
*/
// Blueprints se aap apne Flask application ko components me organize kar sakte
hain.
// Ye bade applications ke liye bahut useful hai.
```

```
//--- PROJECT STRUCTURE ---
// /yourapp/
//   - __init__.py
//   - main/
//     - __init__.py
//     - routes.py
//   - auth/
//     - __init__.py
//     - routes.py

//--- CREATING A BLUEPRINT (auth/routes.py) ---
from flask import Blueprint

auth_bp = Blueprint('auth', __name__, template_folder='templates')

@auth_bp.route('/login')
def login():
    return "This is the blueprint login page."

//--- REGISTERING THE BLUEPRINT (yourapp/__init__.py) ---
from flask import Flask

def create_app():
    app = Flask(__name__)

    from .auth.routes import auth_bp
    app.register_blueprint(auth_bp, url_prefix='/auth') // Blueprint ko register
karein.

    return app

// Ab aap `/auth/login` par blueprint ka route access kar sakte hain.

/*
```


PAGE 8
SESSIONS & FLASHING


```
*/
//--- SESSIONS ---
// Session se aap user ka data requests ke beech store kar sakte hain.
// Flask me sessions cookies ka use karte hain aur cryptographically signed hote
hain.
from flask import session, redirect, url_for

app.config['SECRET_KEY'] = 'another-secret-key' // Sessions ke liye zaroori.

@app.route('/login_session', methods=['POST'])
def login_session():
    session['username'] = request.form['username'] // Session me data store
karein.
    return redirect(url_for('index'))

@app.route('/logout')
def logout():
    session.pop('username', None) // Session se data remove karein.
    return redirect(url_for('index'))

//--- FLASHING ---
// Flashing se aap user ko one-time messages dikha sakte hain.
from flask import flash

@app.route('/login_flash', methods=['POST'])
def login_flash():
```

```
    flash('You were successfully logged in!') // Message flash karein.
    return redirect(url_for('index'))

//--- DISPLAYING FLASH MESSAGES IN TEMPLATE ---
/*
{% with messages = get_flashed_messages() %}
  {% if messages %}
    <ul class=flashes>
    {% for message in messages %}
      <li>{{ message }}</li>
    {% endfor %}
    </ul>
  {% endif %}
{% endwith %}
*/

/*
```



```
*/
//--- CUSTOM ERROR PAGES ---
@app.errorhandler(404)
def page_not_found(error):
    return render_template('404.html'), 404

//--- APPLICATION CONTEXT & REQUEST CONTEXT ---
// Flask me do context hote hain:
// 1. Application Context: `current_app` (current app instance), `g` (global
object for one request).
// 2. Request Context: `request` (request object), `session` (session object).
// Ye contexts view function ke andar automatically available hote hain.

//--- CONTEXT PROCESSORS ---
// Context processors automatically variables ko templates me inject karte hain.
@app.context_processor
def inject_user():
    # Ye dictionary har template ke context me add ho jayegi.
    return dict(user_type='Admin')
// Ab aap template me `{{ user_type }}` use kar sakte hain.

//--- MIDDLEWARE (WSGI) ---
// Aap WSGI middleware se request/response ko modify kar sakte hain.
class SimpleMiddleware:
    def __init__(self, app):
        self.app = app
    def __call__(self, environ, start_response):
        # Modify request/response here
        return self.app(environ, start_response)

// app.wsgi_app = SimpleMiddleware(app.wsgi_app)

/*
```



```
*/
//--- TESTING WITH PYTEST ---
// Installation: pip install pytest
// Create a `tests` folder.

//--- EXAMPLE TEST (tests/test_app.py) ---
```

```
import pytest
from yourapp import create_app // Apne app factory se app import karein

@pytest.fixture
def client():
    app = create_app()
    app.config['TESTING'] = True
    with app.test_client() as client:
        yield client

def test_home_page(client):
    """Test the home page."""
    rv = client.get('/')
    assert rv.status_code == 200
    assert b"Hello, World!" in rv.data

//--- RUNNING TESTS ---
// Terminal:
// pytest

//--- DEPLOYMENT ---
// Development server (flask run) production ke liye suitable nahi hai.
// Production me ek production-ready WSGI server use karein.

//--- GUNICORN ---
// Gunicorn ek popular WSGI server hai.
// Installation: pip install gunicorn
// Run command: gunicorn -w 4 'yourapp:create_app()'
// -w 4: 4 worker processes.

//--- DOCKER ---
// Aap apni Flask app ko Docker container me package kar sakte hain.
// 1. `Dockerfile` banayein.
// 2. Image build karein: `docker build -t my-flask-app .`
// 3. Container run karein: `docker run -p 8000:8000 my-flask-app`

/*****************************************************************
*                       END OF MASTERY GUIDE                     *
*****************************************************************/
```