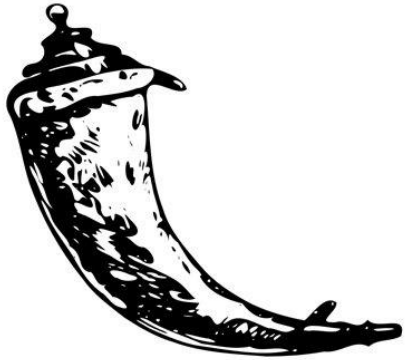
PYTHON FOR WEB DEVELOPMENT

FLASK



Deschamps Marc

Session Objectives

- Understand what Flask is.
- Learn the advantages of Flask compared to other frameworks.
- Install Flask and set up a working environment.
- Write a simple Flask application.

What is Flask?

- Flask is a micro web framework for Python.
- Lightweight and minimalist.
- Extensible with modules (e.g., Flask-SQLAlchemy, Flask-WTF).
- Built on Werkzeug (web server) and Jinja2 (template engine).

Features	Flask	Django
Type	Micro-framework	Full-stack framework
Flexibility	Very high	Moderate
Configuration	Minimalist	Convention over configuration
Best Use Case	Lightweight, modular apps	Complex, large-scale apps

Installing Flask

- Create a virtual environment
  - python -m venv env
  - source env/bin/activate # On Linux/Mac
    env\Scripts\activate # On Windows
- Install Flask
  - pip install flask

```
project/
│
├── app.py          # Main application file
├── templates/      # Folder for HTML files (optional)
└── static/         # Folder for CSS, JS, images (optional)
```

Flask application structure

```python
from flask import Flask

app = Flask(__name__)

@app.route("/")
def home():
    return "Welcome to Flask!"

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

- Flask: The main class for creating the app.
- @app.route: Decorator to define a route.
- debug=True: Enables automatic reloading during development.

Minimal Flask example

Running the app

- Python app.py
  - run --debug
- Flask run -> if app.py
  - -- export FLASK_APP=app.py

Managing Routes and Parameters

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

@app.route("/contact")
def contact():
    return "Contact us at support@example.com."
```

What are Routes?

- Routes define the URLs that your Flask application responds to.
- You assign a function to a specific URL using the `@app.route()` decorator.

POSTMAN

URL Parameters

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

- <int:var>: Accepts integers.
- <float:var>: Accepts floats.
- <path:var>: Accepts full paths.
- <string:var>: Accepts strings.

- If the parameters in the URL don't match the expected types, Flask returns a 404 error.

```
@app.route("/add/<int:a>/<int:b>")
def add(a, b):
    return f"The sum of {a} and {b} is {a + b}."
```

Next

```
@app.route('/', methods=['GET'])
def index():
    return "HW"
```

```
@app.route('/home2', methods=['GET', 'POST'])
def home2():
    return '<h1>YES, You\'re THERE <h1>'
```

Next*

```
@app.route('/home', methods=['GET'], defaults={'name' : 'default'})
@app.route('/home/<string:name>', methods=['GET'])
def home(name):
    return '<h1>YES {}, You\'re THERE <h1>'.format(name)

@app.route('/home2', methods=['GET', 'POST'])
def home2():
    return '<h1>YES, You\'re THERE <h1>'
```

Request

```
@app.route('/query')
def query():
    name = request.args.get('name')
    location = request.args.get('location')
    return '<h1>Yes {}, you are in {}</h1>'.format(name, location)
```

http://127.0.0.1:5000/query?name=name&location=location

Activity

- Implement a Flask application that:
  - Has endpoints for each arithmetic operation: `/add`, `/subtract`, `/multiply`, and `/divide`.
  - Use query parameters and url parameters.
  - Returns the result of the operation in plain text.

  - Handle edge cases:
  - Prevent division by zero and return a meaningful error message.
  - Validate the parameters

Template

```
@app.route('/formE')
def formE():
    return '''<form method="POST" action="/process">
            <input type="text" name="name">
            <input type="text" name="location">
            <input type="submit" value="Submit">
        </form>'''
```

Template

```html
<!DOCTYPE html>
<html>

    <head>
        <title>{{ title }}</title>
    </head>

    <body>
        <h1>{{ message }}</h1>

    </body>

</html>
```

```python
@app.route('/home3', methods=['GET'])
def home3():
    return render_template("home3.html", title="Accueil", message="Bienvenue !")
```

Template

```html
<!DOCTYPE html>
<html>

    <head>
        <title>{{ title }}</title>
    </head>

    <body>
        <h1>{{ message }}</h1>
        {% block content %}{% endblock %}
    </body>

</html>
```

```jinja
{% extends "base.html" %}

{% block content %}
<p>Ceci est la page d'accueil.</p>
{% endblock %}
```

Template with form

```
{% extends "base.html" %}

{% block content %}
<p>This is a form.</p>
<form method="POST" action="/process">
    <input type="text" name="name">
    <input type="text" name="location">
    <input type="submit" value="Submit">
</form>
{% endblock %}
```

Activity

Building a To-Do List Application

- Users should be able to:
  - View the list of tasks.
  - Add new tasks.
  - Delete tasks.

  - Tasks will include:
  - A unique ID.
  - A description.
  - Data storage will be temporary (using a Python list). (No DB)
  - Bonus : persistence in a file

Activity

Building a To-Do List Application - 2

- Add an edit task page
- Allow users to mark tasks as "completed" or "incomplete."

DB with SQL ALCHEMY

SQL ALCHEMY -> ORM

- What is needed :
  - Model
  - DataBase configuration
  - CRUD and Commit

  - No need for SQL

Model

```
class Task(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    description = db.Column(db.String(200), nullable=False)
    completed = db.Column(db.Boolean, default=False)
```

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///tasks.db'
db = SQLAlchemy(app)
```

Database config

CRUD and Commit

```python
@app.route("/addtask")
def addTask():
    new_task = Task(description="Learn Flask", completed=False)
    db.session.add(new_task)
    db.session.commit()
```

```python
@app.route("/display")
def display():
    tasks = Task.query.all()
    return render_template("display.html", tasks=tasks)
```

```python
with app.app_context():
    db.create_all()
```

Oups...

Activity

To-Do List Application – Part 3

- Add a DB for every CRUD operation

Authentication and User Management

Sessions and Cookies

Cookies: Stored on the client side, they are used to persist small amounts of data.

Sessions: Built on cookies, they are used to persist user data securely on the server side.

Flask Sessions:

Flask provides a session object to store user-specific data during a session.

Requires a secret key to sign session data for security.

```python
app = Flask(__name__)
app.secret_key = "your_secret_key"

@app.route("/")
def index():
    username = session.get("username")
    if username:
        return f"Welcome back, {username}!"
    return "Welcome to the app! Please <a href='/login'>login</a>."

@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        session["username"] = request.form["username"]
        return redirect(url_for("index"))
    return '''
        <form method="post">
            <input type="text" name="username" placeholder="Enter your username">
            <button type="submit">Login</button>
        </form>
    '''
```

- **`session["username"]`:** Stores the username in the session.
- **`session.get("username")`:** Retrieves the username from the session.
- **`session.pop("username")`:** Deletes the username from the session.

And...

```python
@app.route("/logout")
def logout():
    session.pop("username", None)
    return redirect(url_for("index"))
```

```
session["role"] = "admin"
```

```
from datetime import timedelta
app.permanent_session_lifetime = timedelta(minutes=30)
```

And...

Activity

To-Do List Application – Part 4

- Add a user for each task
- Ask for the user name and display only the task of the user…using session
- Is name is admin, all the tasks will be displayed

Security with werkzeug

```
pip install werkzeug
```

```python
@app.route("/register", methods=["GET", "POST"])
def register():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        if username in users:
            return "Username already exists!"
        users[username] = generate_password_hash(password)
        return redirect(url_for("login"))
    return '''
        <form method="post">
            <input type="text" name="username" placeholder="Enter your username" required>
            <input type="password" name="password" placeholder="Enter your password" required>
            <button type="submit">Register</button>
        </form>
    '''
```

```python
@app.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
        user_hash = users.get(username)
        if user_hash and check_password_hash(user_hash, password):
            session["username"] = username
            return redirect(url_for("dashboard"))
        return "Invalid username or password!"
    return '''
        <form method="post">
            <input type="text" name="username" placeholder="Enter your username" required>
            <input type="password" name="password" placeholder="Enter your password" required>
            <button type="submit">Login</button>
        </form>
    '''
```

```python
@app.route("/dashboard")
def dashboard():
    if "username" not in session:
        return redirect(url_for("login"))
    return f"Welcome, {session['username']}! <a href='/logout'>Logout</a>"

@app.route("/logout")
def logout():
    session.pop("username", None)
    return redirect(url_for("login"))
```

Activity

To-Do List Application – Part 5

- Update the taskapp security

Final Activity

Build a Event Booking Application where users can register, log in, and book tickets for events. Each user will have a personal dashboard showing the events they've booked.

- Implement user authentication (registration, login, and logout).
- Manage session data to track logged-in users.
- List available events with ticket availability.
- Allow users to book tickets for events.
- Display booked events on a personal dashboard.

Final Activity – more!

Extras

- **Cancel Booking:**
  - Add a feature to cancel a booking and restore ticket availability.
- **Admin Role:**
  - Create an admin role to add new events or manage ticket availability.
- **Search Functionality:**
  - Add a search bar to filter events by name.
- **Styling:**
  - Use CSS to style your application and make it visually appealing.