HTML in Python

1️⃣ Introduction to Embedding HTML within Python Using Web Frameworks

- Python is primarily a backend programming language. To create **web applications**, we need a way to **serve HTML pages** to users through a browser.
- **Web frameworks** like **Django** and **Flask** allow Python to generate and manage **dynamic HTML content**.
- Instead of writing static HTML files manually, these frameworks let Python **embed logic and data into HTML**.
- Key advantages:
    - Separation of **logic** (Python) from **presentation** (HTML)
    - Ability to render dynamic content based on user input or database data
    - Simplifies building complex web applications

Example: Using Python to display a personalized greeting in HTML:

name = "Christian"

html_content = f"<h1>Welcome, {name}!</h1>"

2️⃣ Generating Dynamic HTML Content Using Django Templates

- Django uses a **template engine** to render HTML dynamically.
- Templates allow:
    1. **Embedding variables**:
    2. <h1>Hello, {{ username }}!</h1>
    3. **Conditional statements**:
    4. {% if user.is_authenticated %}
    5. <p>Welcome back!</p>
    6. {% else %}
    7. <p>Please log in.</p>
    8. {% endif %}
    9. **Loops**:
    10.

11. {% for doctor in doctors %}
12. {{ doctor.name }}
13. {% endfor %}
14.
15. **Template inheritance** for reusable layouts (headers, footers, navigation).

- **MVT Architecture in Django**:
  - **Model** → Handles data (database)
  - **View** → Handles logic (Python code)
  - **Template** → Handles presentation (HTML)
- Workflow:
  1. User requests a page (URL).
  2. Django calls the **view function**.
  3. The view passes data to the **template**.
  4. Template generates **dynamic HTML** and sends it to the browser.

This theory explains **how Python integrates with HTML** and why **Django templates** are used for dynamic web pages.