# HTML in Python Theory:

1. **Introduction to embedding HTML within Python using web frameworks like Django or Flask.**

Embedding HTML within Python is primarily accomplished using web frameworks like Django (full-stack) or Flask (micro-framework). These frameworks enforce a clean separation of concerns, ensuring Python code handles logic and data, while HTML handles presentation.

The key to this separation is the Template Engine.

* Django uses the Django Template Language (DTL) by default.
* Flask often uses Jinja2.

Instead of writing HTML inside Python strings, you write HTML files (templates) and use Python to load and fill them with data.

1. **Generating dynamic HTML content using Django templates.**

Django Templates are plain text files (usually ending in .html) that contain static HTML along with special syntax that allows the template engine to insert dynamic content.

1. The Components

| Component | Role | Example Syntax |
| --- | --- | --- |
| Variables | Used to insert dynamic content (data) passed from the Python View. | Hello, {{ user.username }} |
| Tags | Control the logic of the template, such as loops and conditional statements. | {% if user.is\_authenticated %} |
| Filters | Used to transform the values of variables for display (e.g., formatting dates). | {{ pub\_date | date:"F j, Y" }} |

2. The Process

The generation of dynamic HTML follows a clear request-response cycle:

1. Request Handling (View): A request hits a Django View function. The Python code in the View processes the request (e.g., retrieves data from the database using the ORM).
2. Context Creation: The View packages all the required dynamic data into a Python dictionary called the context.

Python

# views.py

def post\_list(request):

posts = Post.objects.all() # Data from the Model

context = {

'title': 'Blog Posts',

'posts': posts

}

# Step 3: Rendering

return render(request, 'blog/post\_list.html', context)

1. Rendering: The View calls the render() shortcut, passing the request, the path to the HTML template, and the context dictionary.
2. Dynamic Insertion: The DTL engine executes the logic in the template, replacing the variable placeholders ({{ title }}) and executing control structures ({% for post in posts %}) with the actual data from the context.
3. Response: The View returns the final, complete, dynamic HTML document as an HTTP response to the browser.

Example Template (DTL)

HTML

<!DOCTYPE html>

<html>

<head>

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

</head>

<body>

<h1>Welcome to the {{ title }} Page</h1>

{% for post in posts %}

<div>

<h2>{{ post.title }}</h2>

<p>Published on: {{ post.pub\_date|date:"F j, Y" }}</p>

<p>{{ post.content|truncatechars:100 }}</p>

</div>

{% empty %}

<p>No posts available yet.</p>

{% endfor %}

</body>

</html>