Name:- Vishvash Limbasiya

**Python - Db And Python Framework**

Q.1:- Introduction to embedding HTML within Python using web frameworks like Django or Flask.

Ans:-

Embedding HTML within Python using web frameworks like **Django** or **Flask** involves using **templates** to generate dynamic web pages. Instead of writing raw HTML in Python code, these frameworks use a **templating engine** (like **Jinja2** in Flask or **Django Template Language**) that lets you:

* Write HTML with placeholders ({{ variable }}) for dynamic content.
* Use control structures like loops and conditionals ({% for item in list %}, {% if condition %}).
* Separate logic (Python) from presentation (HTML), making the code cleaner and easier to maintain.

Q.2:- Generating dynamic HTML content using Django templates.

Ans:-

In Django, dynamic HTML content is generated using **templates** with the **Django Template Language**. You pass data from views (Python code) to templates using a **context dictionary**. In the template, you use placeholders like {{ variable }} and tags like {% for %} or {% if %} to insert and control dynamic content.

View.py:-

def show\_data(request):

data = {'name': 'Vishvash'}

return render(request, 'template.html', data)

Q.3:- Integrating CSS with Django templates.

Ans:-

To integrate CSS with Django templates, you need to manage static files effectively. Here's a breakdown of the process:

* **Create a Static Folder**: In your Django project's root directory, create a folder named static. Inside the static folder, create a css folder.
* **Add CSS Files**: Place your CSS files (e.g., style.css) inside the css folder.

Q.4:- How to serve static files (like CSS, JavaScript) in Django.

Ans:-

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [BASE\_DIR / "static"]

Q.5:- Using JavaScript for client-side interactivity in Django templates.

Ans:-

* Directly in <script> tags: You can include JavaScript code directly within <script> tags in your HTML templates. This is suitable for small amounts of code.

Code:-

<script>  
 document.addEventListener('DOMContentLoaded', function() {  
 // Your JavaScript code here  
 alert('Page loaded!');  
 });  
 </script>

Q.6:- Linking external or internal JavaScript files in Django.

Ans:-

1. Static Files Setup:

* Ensure you have the django.contrib.staticfiles app included in your INSTALLED\_APPS setting in settings.py.
* Define the STATIC\_URL setting, typically as /static/.
* Create a static directory within your app or project directory to store your static files.
* Inside the static directory, create subdirectories for different types of static files (e.g., js for JavaScript files).

2. Internal JavaScript:

* Use the <script> tag directly within your HTML template.
* Place your JavaScript code between the opening and closing <script> tags.
* This is suitable for small amounts of code specific to a single template.

Q.7:- Overview of Django: Web development framework.

Ans:-

* **MVT Architecture:**

Django follows the Model-View-Template (MVT) architectural pattern, which promotes separation of concerns and makes code more organized.

* **Model:**

Represents the data and database interactions, often using an Object-Relational Mapper (ORM) to simplify database operations.

* **View:**

Handles incoming requests, interacts with the model, and selects the appropriate template to display.

* **Template:**

Generates the HTML for the web page, allowing for dynamic content based on the data from the model.

* **Built-in Features:**

Django provides various pre-built features, including user authentication, admin panel, form handling, and more, which can be used out of the box.

* **Security:**

Django is designed with security in mind, helping to prevent common vulnerabilities like SQL injection and cross-site scripting

Q.8:- Advantages of Django (e.g., scalability, security).

Ans:-

Django includes strong security features to protect web applications from common vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF) attacks. It also supports SSL/HTTPS for encrypted data transmission, making applications more secure.

Q.9:- Understanding the importance of a virtual environment in Python projects.

Ans:-

**Isolation**: Keeps project dependencies separate from global Python setup.

**Avoid Conflicts**: Prevents version clashes between packages used in different projects.

**Reproducibility**: Ensures consistent environments across development, testing, and deployment.

**Cleaner Setup**: Keeps your system Python clean and reduces risk of breaking system tools.

python -m venv venv

source venv/bin/activate

venv\Scripts\activate

Q.10:- Using venv or virtualenv to create isolated environments.

Ans:-

* 1. python -m venv venv
  2. pip install virtualenv
  3. virtualenv venv
  4. venv\Scripts\activate

Q.11:- Understanding the role of manage.py, urls.py, and views.py.

Ans:-

**manage.py**:  
Command-line tool to manage the project (run server, migrate DB, create apps, etc.).

python manage.py runserver

* **urls.py**:  
  Maps URLs to views. Defines how different web addresses route to specific code.
* **views.py**:  
  Contains functions or classes that handle requests and return responses (like HTML, JSON).

Q.12:- Django’s MVT (Model-View-Template) architecture and how it handles request-response cycles.

Ans:-

**Django’s MVT Architecture & Request-Response Cycle :**

* **Model**: Handles database – defines data structure.
* **View**: Contains business logic – processes requests and fetches data.
* **Template**: Handles UI – displays data in HTML.

Q.13:- Introduction to Django’s built-in admin panel.

Ans:-

**Enable admin app** (default in INSTALLED\_APPS).

**Create superuser**:

python manage.py createsuperuser

Q.14:- Setting up URL patterns in urls.py for routing requests to views.

Ans:-

Import views:-

from django.urls import path

from . import views

Urls:-

urlpatterns = [

path('', views.home, name='home'), # root URL

path('about/', views.about, name='about'), # /about URL

]

Q.15:- integrating templates with views to render dynamic HTML content.

Ans:-

Views.py:-

from django.shortcuts import render

def home(request):

context = {'name': ‘Vishvash’}

return render(request, 'home.html', context)

Q.16:- Connecting Django to a database (SQLite or MySQL).

Ans:-

1.Default (SQLite):

python manage.py migrate

2.For MySQL:

pip install mysqlclient

Q17:- Perform CRUD operations using Django ORM.

Ans:-

* **Create:**

obj = MyModel(field1='value', field2=123)

obj.save()

* **Read:**

all\_objs = MyModel.objects.all()

one\_obj = MyModel.objects.get(id=1)

filtered = MyModel.objects.filter(field1='price')

* **Update:**

obj = MyModel.objects.get(id=1)

obj.field2 = 456

obj.save()