**Name:- Vishvash Limbasiya**

**Python DB and Framework**

Q.1:- Introduction to embedding HTML within Python using web frameworks like Django or Flask. from flask import Flask, render\_template.

app = Flask(\_\_name\_\_)

@app.route("/")

def home():

return render\_template("index.html", name="Alice")

Ans:-

Django:

from django.shortcuts import render

def home(request):

return render(request, "index.html", {"name": "vishvash"})

Flask:

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route("/")

def home():

return render\_template("index.html", name=" vishvash")

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

Ans:-

View (Python):

def home(request):

return render(request, 'index.html', {'name': vishvash})

Q.3:- Integrating CSS with Django templates.

Ans:-

{% load static %}

<link rel="stylesheet" href="{% static 'css/style.css' %}">

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

Ans:-

STATIC\_URL = '/static/'

{% load static %}

<script src="{% static 'js/script.js' %}"></script>

<link rel="stylesheet" href="{% static 'css/style.css' %}">

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

Ans:-

{% load static %}

<script src="{% static 'js/script.js' %}"></script>

Q.6:- Overview of Django: Web development framework.

Ans:-

**Django** is a high-level Python web framework that enables rapid development of secure and maintainable websites. It follows the.

**Model-View-Template (MVT)** architecture and includes built-in features like an admin interface, ORM (Object-Relational Mapping), authentication, and routing. Django emphasizes **reusability, scalability**, and the **“Don’t Repeat Yourself (DRY)”** principle, making it ideal for both simple and complex web applications.

Q.7:- Advantages of Django (e.g., scalability, security).

Ans:-

**Advantages of Django:**

* **Scalability**: Handles high traffic and large volumes of data efficiently.
* **Security**: Protects against common threats like SQL injection, CSRF, and XSS.
* **Rapid Development**: Built-in tools speed up development.
* **Versatile**: Suitable for all types of projects—from small apps to large systems.
* **Batteries-Included**: Comes with many built-in features (admin panel, auth system, etc.).
* **Community Support**: Large, active community with rich documentation.

Q.8:- Django vs. Flask comparison: Which to choose and why.

Ans:-

**Django**: Full-featured, batteries-included framework. Best for larger projects needing built-in tools (admin panel, ORM, auth).

* *Choose Django* if you want rapid development with lots of built-in functionality.

Flask: Lightweight and flexible. Best for smaller projects or when you want more control over components.

* *Choose Flask* if you prefer a minimal setup and want to customize everything.

Q.9:- Steps to create a Django project and individual apps within the project.

Ans:-

1.Create virtual environment:

python -m venv env && source env/bin/activate

2.Install Django:

pip install django

3.Create project:

django-admin startproject projectname

4.Create app:

cd projectname && python manage.py startapp appname

5.Register app:

Add 'appname' to INSTALLED\_APPS in settings.py.

6.Run migrations:

python manage.py makemigrations && python manage.py migrate

7.Start server:

python manage.py runserver

Q.10:- Understanding the role of manage.py, urls.py, and views.py.

Ans:-

1. **manage.py**:  
A command-line utility to interact with your Django project (e.g., run the server, make migrations, create apps).

2. **urls.py**:  
Maps URL paths to specific views. It controls how web requests are routed within the project or app.

3. **views.py**:  
Contains functions or classes that define what content to display for a given URL (i.e., the logic behind each page).

Q.11:- Introduction to Django’s built-in admin panel.

Ans:-

**Register models** in admin.py inside your app:

from django.contrib import admin

from .models import YourModel

admin.site.register(YourModel)

Q.12:- Customizing the Django admin interface to manage database records.

Ans:-

**Register models** in admin.py:

from django.contrib import admin

from .models import MyModel

admin.site.register(MyModel)

**Use ModelAdmin** to customize:

class MyModelAdmin(admin.ModelAdmin):

list\_display = ('field1', 'field2')

search\_fields = ('field1',)

list\_filter = ('field2',)

admin.site.register(MyModel, MyModelAdmin)

Q.13:- Setting up URL patterns in urls.py for routing requests to views.

Ans:-

Project-level urls.py:

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('', include('myapp.urls')), # Include app URLs

]

App-level urls.py (myapp/urls.py):

from django.urls import path

from . import views

urlpatterns = [

path('', views.home, name='home'),

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

]

Q.14:- Integrating templates with views to render dynamic HTML content.

Ans:-

Set up the view in views.py:

from django.shortcuts import render

def home(request):

return render(request, 'home.html', {'name': 'John'})

Q.15:- javaScript for front-end form validation.

Ans:-

<form onsubmit="return validateForm()">

<input type="text" id="name" required>

<span id="error"></span>

<button type="submit">Submit</button>

</form>

<script>

function validateForm() {

const name = document.getElementById("name").value;

if (name === "") {

document.getElementById("error").textContent = "Name is required.";

return false;

}

return true;

}

</script>

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

Ans:-

Update settings.py:

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'your\_db\_name',

'USER': 'your\_username',

'PASSWORD': 'your\_password',

'HOST': 'localhost',

'PORT': '3306',

}

}

Q.17:- Using the Django ORM for database queries.

Ans:-

1. **Create**

Book.objects.create(title="Django Basics", author="Vishvash ")

**2. Read**

books = Book.objects.all)

book = Book.objects.get(id=1)

books = Book.objects.filter(author="Vishvash ")

3. **Update**

book = Book.objects.get(id=1)

book.title = "Advanced Django"

book.save()

**4. Delete**

book = Book.objects.get(id=1)

book.delete()

Q.18:- Understanding Django’s ORM and how QuerySets are used to interact with the database.

Ans:-

All records:

Book.objects.all()

Q.19:- Using Django’s built-in form handling

Ans:-

1. **Define a form (in forms.py):**

from django import forms

class ContactForm(forms.Form):

name = forms.CharField()

email = forms.EmailField()

2. **Use in a view (views.py):**

from .forms import ContactForm

def contact\_view(request):

form = ContactForm(request.POST or None)

if form.is\_valid():

# Process form.cleaned\_data

...

return render(request, 'contact.html', {'form': form})

3. **Render in template (contact.html):**

<form method="post">

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit">Send</button>

</form>

Q.20:- Implementing Django’s authentication system (sign up, login, logout, password management).

Ans:-

1. **Sign Up (Custom View + UserCreationForm):**

from django.contrib.auth.forms import UserCreationForm

from django.shortcuts import render, redirect

def signup\_view(request):

form = UserCreationForm(request.POST or None)

if form.is\_valid():

form.save()

return redirect('login')

return render(request, 'signup.html', {'form': form})

2. **Login / Logout (Built-in Views):**

from django.contrib.auth import views as auth\_views

urlpatterns = [

path('login/', auth\_views.LoginView.as\_view(), name='login'),

path('logout/', auth\_views.LogoutView.as\_view(), name='logout'),

]

3. **Password Management (Built-in URLs):**

urlpatterns += [

path('password\_change/', auth\_views.PasswordChangeView.as\_view(), name='password\_change'),

path('password\_reset/', auth\_views.PasswordResetView.as\_view(), name='password\_reset'),

]