1. What is Django and what is its main purpose?

Django is a high-level Python web framework that enables rapid development and clean, pragmatic design for web applications. It follows the "Don't Repeat Yourself" (DRY) principle and includes built-in features like an ORM (Object-Relational Mapper), authentication, and an admin interface. Its main purpose is to simplify the development of database-driven web applications by providing reusable components and security features.

2. Explain the MTV (Model-Template-View) architecture in Django.

Django follows the MTV (Model-Template-View) architecture, which separates concerns for better organization:

* Model: Represents the database structure and defines data fields and relationships. It allows easy data management using Django’s ORM.
* View: Contains the application logic, handles user requests, interacts with the model, and returns the appropriate response.
* Template: Manages the presentation layer and renders HTML dynamically by embedding data.

This structure helps maintain code organization and reusability.

3. How do you install Django?

To install Django, follow these steps:

1. Create a Virtual Environment (optional but recommended):

python -m venv myenv

source myenv/bin/activate # On macOS/Linux

myenv\Scripts\activate # On Windows

1. Install Django using pip:

pip install django

1. Verify the installation:

django-admin --version

4. What is a Django project and how do you create one?

A Django project is a collection of settings, applications, and configurations that form a web application. To create a Django project, run:

django-admin startproject project\_name

This creates the following structure:

project\_name/

│── manage.py

│── project\_name/

│ │── \_\_init\_\_.py

│ │── settings.py

│ │── urls.py

│ │── wsgi.py

To start an app inside the project:

python manage.py startapp app\_name

5. What is the purpose of the settings.py file in a Django project?

The settings.py file contains the configuration settings of a Django project, including:

* Database configurations
* Installed applications
* Middleware settings
* Template and static file settings
* Security settings (e.g., allowed hosts, authentication)

It acts as the central configuration file for the project.

6. How do you start the Django development server?

To start the development server, navigate to the project directory and run:

python manage.py runserver

This will start the server at http://127.0.0.1:8000/, where you can access your application.

7. What is the role of the urls.py file in a Django project?

The urls.py file is responsible for routing URLs to views. It defines the URL patterns that direct requests to the appropriate view function or class.

Example:

from django.urls import path

from . import views

urlpatterns = [

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

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

]

8. How do you define URL patterns in Django?

URL patterns are defined in the urls.py file using the path() function:

from django.urls import path

from . import views

urlpatterns = [

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

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

]

Each pattern consists of:

URL path ('home/')

View function (views.home)

Optional name for the URL (name='home')

9. What is a Django model and how do you define one?

A Django model is a Python class that defines the structure of a database table. It allows interaction with the database using Django’s ORM.

Example:

from django.db import models

class Book(models.Model):

title = models.CharField(max\_length=200)

author = models.CharField(max\_length=100)

published\_date = models.DateField()

Each class attribute represents a field in the database table.

10. What are migrations in Django and how do you create and apply them?

Migrations track changes to database models and apply them to the database. They help manage schema changes efficiently.

To create and apply migrations:

1. Create migration files:

python manage.py makemigrations

1. Apply migrations to the database:

python manage.py migrate

11. How do you interact with the Django shell?

Django provides an interactive shell to execute Python code within the Django environment.

To open the shell:

python manage.py shell

Example usage inside the shell:

from myapp.models import Book

Book.objects.create(title="Django Guide", author="John Doe")

12. What is the Django admin interface and how do you use it?

The Django admin interface allows managing database records via a web-based UI.

To enable it:

1. Create a superuser:

python manage.py createsuperuser

1. Register models in admin.py:

from django.contrib import admin

from .models import Book

admin.site.register(Book)

1. Access the admin panel at http://127.0.0.1:8000/admin/.

13. How do you create views in Django?

Views handle the application logic. Example function-based view:

from django.http import HttpResponse

def home(request):

return HttpResponse("Welcome to Django!")

Example class-based view:

from django.views import View

from django.http import HttpResponse

class HomeView(View):

def get(self, request):

return HttpResponse("Welcome to Django!")

14. What is a template in Django and how do you create one?

A Django template is an HTML file with dynamic content using template tags.

Example:

<!DOCTYPE html>

<html>

<head><title>Home</title></head>

<body>

<h1>Welcome, {{ user\_name }}</h1>

</body>

</html>

15. How do you pass data from views to templates in Django?

Use the context dictionary in render():

from django.shortcuts import render

def home(request):

context = {'user\_name': 'John'}

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

16. Explain the purpose of context processors in Django.

Context processors automatically pass global variables to all templates. They are defined in settings.py under TEMPLATES.

Example:

'context\_processors': [

'django.template.context\_processors.request',

'django.template.context\_processors.auth',

]

17. What are static files in Django and how do you handle them?

Static files (CSS, JS, images) are managed in the static/ directory.

Example in settings.py:

STATIC\_URL = '/static/'

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

In templates:

{% load static %}

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

18. How do you handle forms in Django?

Django provides forms.py for handling forms. Example:

from django import forms

class ContactForm(forms.Form):

name = forms.CharField(max\_length=100)

email = forms.EmailField()

19. Explain Django templates and template inheritance.

Templates allow dynamic content. Template inheritance avoids repetition.

Base template (base.html):

{% block content %}{% endblock %}

Child template (home.html):

{% extends 'base.html' %}

{% block content %} Welcome Home! {% endblock %}

20. How to change the default server in django?

By default, Django runs its development server on http://127.0.0.1:8000/ using the command:

python manage.py runserver

However, you can change the host, port, or even use a different server.

1. Change the Host and Port

You can specify a different host and port when starting the server.

Change the Port

To run Django on a different port (e.g., 9000):

python manage.py runserver 9000

Change the Host

To allow external access (e.g., from other devices on the network), use 0.0.0.0:

python manage.py runserver 0.0.0.0:8000

Make sure to update ALLOWED\_HOSTS in settings.py:

ALLOWED\_HOSTS = ['\*'] # Allows all hosts (for development only)

# Or specify allowed hosts explicitly:

# ALLOWED\_HOSTS = ['yourdomain.com', '192.168.1.100']

2. Use a Different WSGI Server

For production, it's recommended to use a WSGI server like Gunicorn or Daphne (for ASGI applications).

Using Gunicorn (For WSGI)

1. Install Gunicorn:

pip install gunicorn

1. Run Django using Gunicorn:

gunicorn project\_name.wsgi:application --bind 0.0.0.0:8000

Using Daphne (For ASGI)

1. Install Daphne:

pip install daphne

1. Run Django using Daphne:

daphne -b 0.0.0.0 -p 8000 project\_name.asgi:application

3. Using Nginx and Gunicorn for Production

For production environments, use Nginx as a reverse proxy and Gunicorn as the WSGI server.

1. Install Gunicorn:

pip install gunicorn

1. Run Gunicorn:

gunicorn --workers 3 project\_name.wsgi:application --bind 0.0.0.0:8000

1. Configure Nginx to proxy requests to Gunicorn.