Django Syllabus From **18th February**

* **By Digital Pathshala**

**Contents**

[1. Day 1: Introduction to Django 4](#_Toc190968123)

[1.1. What is Django? Why use Django? 4](#_Toc190968124)

[1.2. Setting up a Django project 4](#_Toc190968125)

[1.2.1. Check if Python is Installed 4](#_Toc190968126)

[1.2.2. Install pip (Python Package Manager) 4](#_Toc190968127)

[1.3. Installing and using a virtual environment 5](#_Toc190968128)

[2. Day 2: Django Project Structure 7](#_Toc190968129)

[2.1. Installation of Django on virtual environment 7](#_Toc190968130)

[2.2. Creation of Django Project 7](#_Toc190968131)

[2.3. Running the development server 7](#_Toc190968132)

[2.4. Creating and exploring a Django app 7](#_Toc190968133)

[2.5. Understanding Django’s Folder Structure 8](#_Toc190968134)

[2.6. Role of Key Files 9](#_Toc190968135)

[2.7. Configuring Django Settings 12](#_Toc190968136)

[2.8. Creating and Managing Django Apps 12](#_Toc190968137)

[3. Day 3: URLs and Views 13](#_Toc190968138)

[3.1. How Django processes a request 13](#_Toc190968139)

[3.2. Mapping URLs with urls.py 14](#_Toc190968140)

[3.3. Function-based views (FBVs) vs. Class-based views (CBVs) 14](#_Toc190968141)

[3.4. Django Templates 15](#_Toc190968142)

[3.4.1. Template syntax ({{}} and {% %}) 15](#_Toc190968143)

[3.4.2. Rendering templates using render() 16](#_Toc190968144)

[3.4.3. Using template inheritance (base.html) 17](#_Toc190968145)

[3.4.4. Static files (CSS, JS, images) 17](#_Toc190968146)

**Week 1: Django Basics (Days 1–6)**

# Day 1: Introduction to Django

## What is Django? Why use Django?

## Setting up a Django project

### Check if Python is Installed

Before installing Django, make sure Python is installed on your system.

For Windows, open Command Prompt (cmd) and run **python --version** or **python3 --version.**

If Python is not installed, download it from the official Python website and install it.

### Install pip (Python Package Manager)

**For Windows:**

Download the pip installer script from: **https://bootstrap.pypa.io/get-pip.py**

Go to the download directory and run the command:

**python get-pip.py**

**or python3 get-pip.py**

Find the Scripts directory inside the Python installation folder and add it to Environment Variables.

**For Mac:**

Run the following command to download the pip installer script:

**curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py**

Execute the script using:

**python3 get-pip.py**

**For Linux:**

Install pip using:

**sudo apt-get install python3-pip python-dev**

Verify pip Installation:

Run the following command to check if pip is installed correctly:

**pip --version**

**or pip3 –version**

## Installing and using a virtual environment

After successfully installing pip, the next step is to set up a **virtual environment** using **venv**. A virtual environment allows you to manage dependencies separately for each project, making it useful for deployment or transferring projects to another system.

**Installing Virtual Environment**

Run the following command to install **venv**:

* **pip install virtualenv**

Each project should have its own **venv** to maintain package isolation.

**To create virtual environment file:**

**python -m venv env**

**Activating the Virtual Environment**

* On Windows (Command Prompt or VS Code Terminal)
  + \venv\_folder\_name\Scripts\activate.bat
  + \venv\_folder\_name\Scripts\activate
* On Mac/Linux (Terminal)
  + source venv\_folder\_name/bin/activate

**Note:** If the activation does not work in **PowerShell**, open **Command Prompt** in VS Code and try again.

Generating packages on requirement.txt file:

pip freeze > requirements.txt

# Day 2: Django Project Structure

## Installation of Django on virtual environment

Once the virtual environment is activated, install Django using:

* **pip install django**

## Creation of Django Project

To create a new Django project, run:

* **django-admin startproject** project\_name : blogapp

**Note: Replace project\_name with your preferred project name.**

## Running the development server

Navigate to the project directory and start the Django server:

* cd project\_name
* **python manage.py runserver**

## Creating and exploring a Django app

Inside your project folder, create a Django app using:

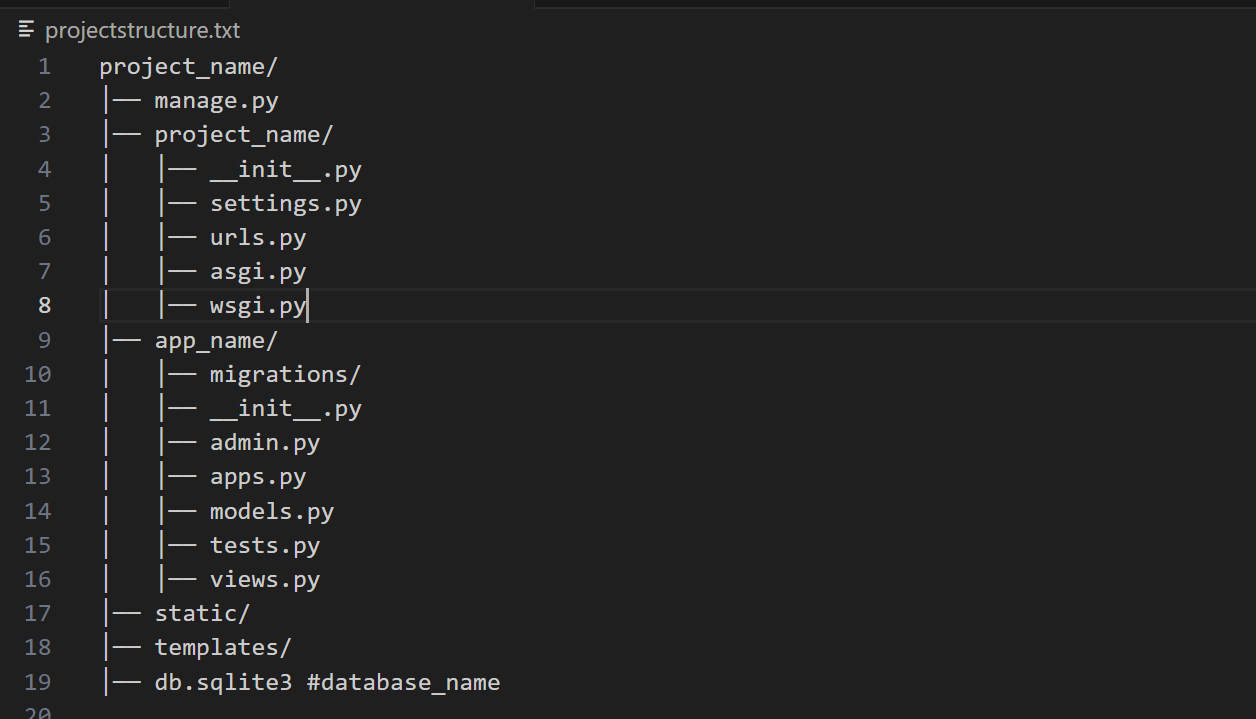
* python manage.py startapp app\_name

For example, to create an app for a **blog website**:

* **python manage.py startapp** blog
* **python manage.py startapp** blog # Handles blog posts
* **python manage.py startapp** users # Manages user authentication
* **python manage.py startapp** comments # Manages comments on blog posts
* **python manage.py startapp** categories # Manages categories for blog posts
* **python manage.py startapp** likes # Manages likes on blog posts and comments

## Understanding Django’s Folder Structure

After creating a Django project (**django-admin startproject** project\_name), the structure will look like this:



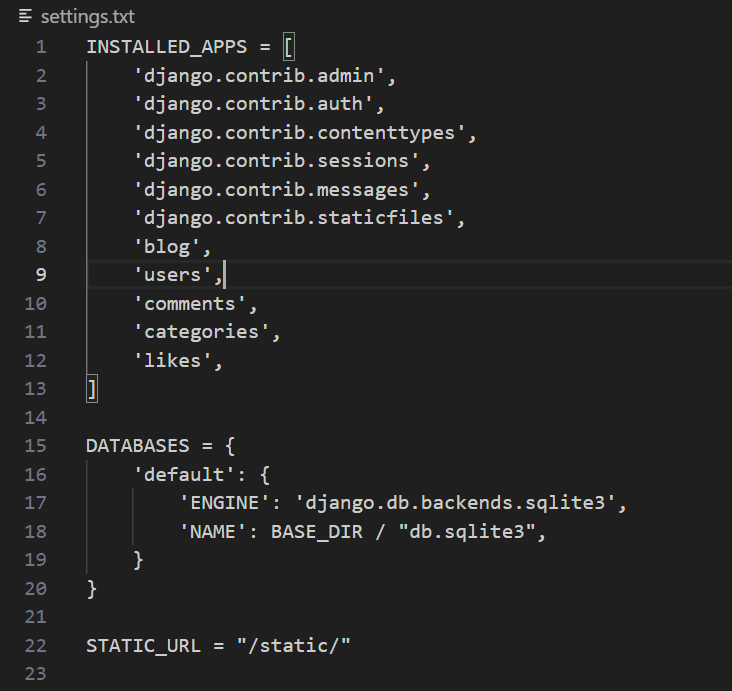
Each file serves a specific role:

* **manage.py**: A command-line utility for managing the Django project.
* **settings.py**: Configures settings like databases, middleware, installed apps, etc.
* **urls.py**: Defines URL patterns and routes to views.
* **wsgi.py / asgi.py**: Entry points for web servers (**Web Server Gateway Interface** (WSGI ) for synchronous apps**, Asynchronous Server Gateway Interface**( ASGI) for async apps).
* **models.py**: Defines database models (tables).
* **views.py**: Contains functions or classes that handle requests and return responses.
* **admin.py**: Registers models to the Django admin panel.
* **apps.py**: Configures app settings.
* **migrations/**: Stores database migration files.

## Role of Key Files

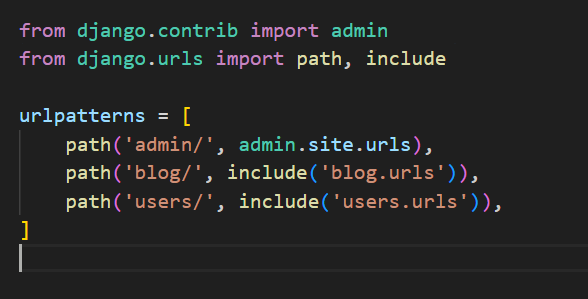
* **settings.py** (Configures Django Settings)

This file manages configurations like installed apps, middleware, databases, static files, etc.



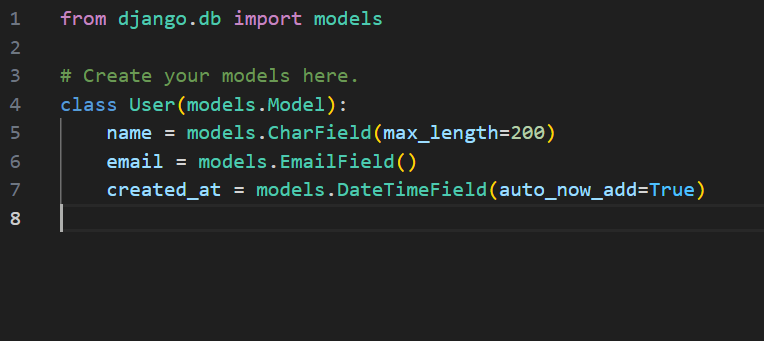
* **urls.py** (Defines URL Routing)

Maps URLs to views.



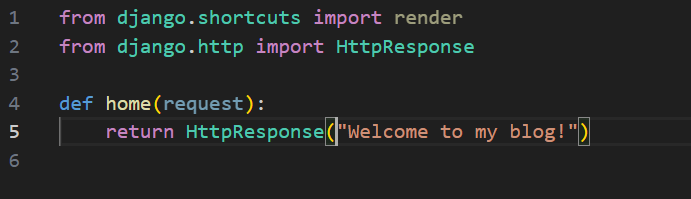
* **models.py** (Defines Database Tables)

Used to create and manage database tables using Django ORM.



* **views.py** (Handles Business Logic)

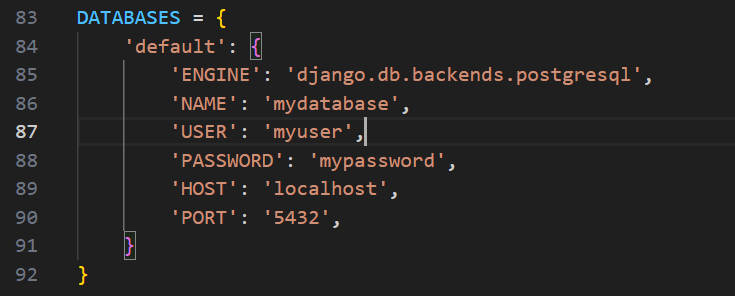
Defines what happens when a user visits a particular URL.



## Configuring Django Settings

Django settings can be modified inside settings.py. Common configurations include:

* Database Configuration

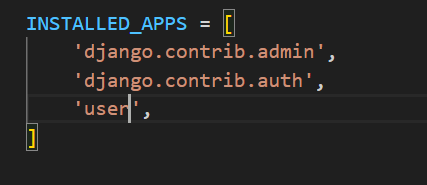


## Creating and Managing Django Apps

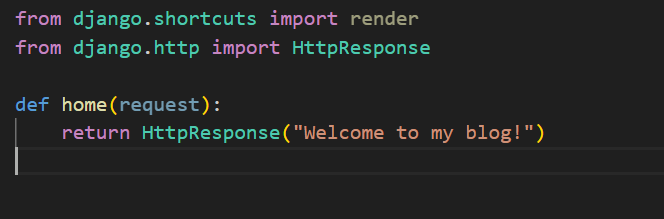
* **Creating an App**

python manage.py startapp app\_name

* Registering an App



* Creating Views



* Creating URLs

A screen shot of a computer code

AI-generated content may be incorrect.

# Day 3: URLs and Views

## How Django processes a request

Django follows a **request-response cycle** when handling HTTP requests. The key steps are:

* **User Makes a Request**
  + A user accesses a URL (e.g., http://localhost:8000/home).
* **Django URL Resolver**
  + Django searches for a matching pattern in urls.py.
* **View Execution**
  + If a matching URL pattern is found, the associated **view function** or **class-based view** (CBV) is executed.
* **Response Generation**
  + view processes the request and returns an HttpResponse (HTML, JSON, etc.).
* **Browser Displays Response**
  + The response is rendered in the user's browser

## Mapping URLs with urls.py

Django uses URL patterns to map requests to specific views. The main urls.py file is typically located in the project folder (next to settings.py).

## Function-based views (FBVs) vs. Class-based views (CBVs)

Django provides two ways to define views:

* **Function-Based Views (FBVs)**

FBVs use simple functions to handle requests.



* **Class-Based Views(CBVs)**

CBVs provide reusable and object-oriented views.

View: This is the base class for all CBVs. It dispatches HTTP requests to the appropriate method (get(), post(), etc.).

A screen shot of a computer screen

AI-generated content may be incorrect.

## Django Templates

### Template syntax ({{}} and {% %})

**{{ }}**: This is the syntax used to output variables and dynamic data into HTML. For example:



**{% %}**: This syntax is for control structures like loops and conditionals. For example:

This helps in rendering logic-based content.

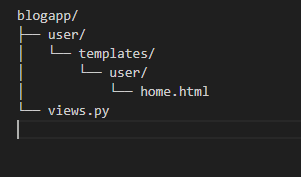
### Rendering templates using render()

The render() function in Django is used to generate an HTTP response by combining a template with context data. This is typically used in views to dynamically render HTML pages.

**Steps to render templates**:

**Create the Folder Structure**

* First, inside the user folder, create a folder called templates.
* Inside the templates folder, create another folder called user (this helps to organize templates by app).
* Inside the user folder, create the HTML file that you want to render (e.g., home.html).



**Create the Template (home.html)**:

* Now, in the home.html file, you can write your HTML content.
* You can use Django’s template syntax to display dynamic data passed from the view.

Example code:

A screenshot of a computer

AI-generated content may be incorrect.

**Create the View Function**:

* In your views.py file, create a function that will render the HTML template using the render() function.
* The render() function combines the template (home.html) with the context (data you want to pass to the template) and the request.

A screen shot of a computer code

AI-generated content may be incorrect.



Html file message displaying successfully:

### Using template inheritance (base.html)

* **Understanding Template Inheritance**

Template inheritance in Django allows us to define a common **base template (base.html)** that can be extended by other templates (**like home.html, dashboard.html**). This helps us **reuse** code instead of repeating it in multiple files.

In Django, we typically create **a base template** that includes common elements like:

* **Navigation bar**
* **Footer**
* **CSS & JS references**

Then, individual pages like **home, about, and contact pages** extend this base template.

**Types of Template Inheritance**

1. **App-Based Inheritance**

* Each app in a Django project has its own **templates directory**.
* This structure is useful when apps need their own styles/layouts.

1. **Project-Based Inheritance**

* A global **templates directory** is created at the **project level**, making it available to all apps.
* This is useful when multiple apps share the same layout.

Since base.html is in the **project-level templates/ folder**, it can be accessed from any app.

**Setting Up Template Inheritance**

**Step 1: Creating base.html (parent file)**

First, create a templates/ directory inside the project folder

Folder Structure:

A screenshot of a computer program

AI-generated content may be incorrect.

Now, inside **templates/base.html**, add the following code:

A screen shot of a computer

AI-generated content may be incorrect.

Notes:

* **{% block title %} ... {% endblock %}**
  + This allows child templates to override the page title dynamically.
  + If not overridden, "My Website" will be used as the default title.
* **{% url 'home' %}** dynamically generates the URL for the "About" page.
* **{% block content %}** is a placeholder for child templates.
* Any template that extends base.html can insert content here.

**Step 2: Registering the Templates Directory in settings.py**

Since base.html is in the templates/ folder, we need to configure Django to recognize it.

In settings.py, modify the TEMPLATES setting:

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 3: Extending base.html in home.html**

Now that we have base.html, let’s inherit it in **home.html**.

Inside **user/templates/user/home.html**, write:

A screen shot of a computer program

AI-generated content may be incorrect.

Rules to Follow While Extending Templates

* **{% extends "base.html" %}** must always be at the very top of the file (before any HTML or whitespace).
* Only define **{% block %}** ... **{% endblock %}** sections that exist in base.html.
* Use {% block %} to override only specific sections without modifying the entire structure.
* Ensure base.html includes **{% block content %}** and other placeholders where child templates can insert content.
* Avoid adding **<html>, <head>, or <body>** tags in a child template unless needed, as they are inherited from base.html.