What is Framework (Hindi) lec#1

This video by **Geeky Shows** provides a comprehensive explanation of what a software framework is, particularly for developers who are just starting out. Below is a detailed summary of the key concepts covered in the lesson:

**1. Core Definition of a Framework**

A framework is essentially a **foundation or a platform** that provides a pre-defined set of rules, libraries, and structures for building applications.

* **The Blueprint Analogy:** The video compares a framework to the pre-built structure of a house. Instead of making every single brick and beam yourself (writing everything from scratch), you are given a solid structure where you just need to design the rooms and finish the interiors.
* **Standardization:** It provides a standard way to build and deploy applications, ensuring that code is organized and maintainable.

**2. Why Use a Framework? (Key Advantages)**

The lesson highlights several reasons why professional developers rely on frameworks rather than writing "Vanilla" (pure) code:

* **Speed & Efficiency:** Since common tasks (like database connection, user authentication, or form handling) are pre-written, developers save a massive amount of time.
* **Security:** Frameworks often come with built-in security features to protect against common vulnerabilities like SQL injection or cross-site scripting (XSS).
* **Maintenance:** Because the framework follows a specific structure, it is much easier for other developers to understand and work on your code later.
* **Testing & Debugging:** Many frameworks include integrated tools for testing, making it easier to find and fix bugs.

**3. Framework vs. Library**

The video clarifies a common point of confusion: the difference between a **Library** and a **Framework**.

* **Control (Inversion of Control):** In a library, *you* are in control. You call the library whenever you need a specific tool. In a framework, the *framework* is in control. It tells you where to put your code and calls it when needed.
* **Scope:** A library is usually for a specific task (e.g., a math library), while a framework is for building the entire application.

**4. Real-World Examples Mentioned**

The tutorial lists popular frameworks across different programming languages to give context to your roadmap:

* **Python:** Django, Flask.
* **JavaScript:** React, Angular, Vue (Frontend) and Express.js (Backend).
* **PHP:** Laravel, CodeIgniter.
* **Java:** Spring, Hibernate.

**Conclusion**

The main takeaway is that while you *can* build a website without a framework, using one makes the process **faster, more secure, and more professional**. It allows you to focus on the unique features of your website (the "Business Logic") instead of the repetitive technical setup.

What is Web Framework (Hindi) lec#2

The video "What is Web Framework (Hindi)" by Geeky Shows provides a detailed explanation of web frameworks and their role in web application development. Here's a summary of the key points:

* **Definition**: A web framework (WF) or web application framework (WAF) is a software framework designed to support the development of web applications, including web services, web APIs, and web resources [[00:13](http://www.youtube.com/watch?v=_yinh8m3M78&t=13)].
* **Purpose**: Web frameworks aim to simplify common web development operations by providing tools and libraries for tasks like templating, database access, session management, and code reuse [[00:25](http://www.youtube.com/watch?v=_yinh8m3M78&t=25)].
* **Benefits**: They help developers save time, increase productivity, and serve clients more efficiently by offering pre-written code for common tasks [[01:31](http://www.youtube.com/watch?v=_yinh8m3M78&t=91)].
* **Architecture**: Over 80% of web frameworks rely on the Model View Controller (MVC) architecture, which separates the application into three interconnected parts [[01:49](http://www.youtube.com/watch?v=_yinh8m3M78&t=109)].
* **Examples**: Some popular web frameworks include Laravel, CodeIgniter, and Zend for PHP, Django for Python, and Spring for Java [[02:29](http://www.youtube.com/watch?v=_yinh8m3M78&t=149)].

The video serves as an introductory guide for beginners interested in understanding the fundamental concepts of web frameworks.

What is Model View Template

**What is MVT?**

MVT stands for Model – View – Template. It is an architectural pattern used by Django to organize web applications. Think of it as a way to separate your data, logic, and presentation so your code is clean, manageable, and scalable.

**Components of MVT**

1. **Model**
   * Represents the data layer of your application (database structure).
   * Defines what data you have and how it’s stored.
   * Example: A **Student** model may have fields like **name**, **age**, **grade**.
2. **View**
   * Handles business logic.
   * Receives user requests, interacts with models, and decides what data to show.
   * Example: A view function may fetch all students from the database and send it to the template.
3. **Template**
   * The presentation layer (HTML with dynamic placeholders).
   * Shows data to the user in a readable format.
   * Example: An HTML page displaying a table of students fetched from the view.

**MVT Flow in Django**

1. User sends a request via the browser.
2. Django’s View receives the request.
3. The View queries the Model (database) for data.
4. The View passes the data to a Template.
5. Template renders HTML and sends it back to the user.

**Analogy:**

* Model: Kitchen (ingredients / data)
* View: Chef (decides what to cook)
* Template: Plate & presentation (how food/data is displayed to customer)

**Model–View–Template pattern explained**

* **Model:** Defines the structure of your data and how it’s stored (database).
* **View:** Takes user requests, talks to the model, and returns a response.
* **Template:** HTML mixed with Django tags that generates the final web page sent to users.

**How Django uses MVT vs MVC**

* Django’s “View” is like the controller in traditional MVC.
* The “Template” is the part that creates the user‑facing UI.
* Models handle the database logic.

**Why MVT matters**

* Separates code by concerns (data, logic, presentation).
* Makes code easier to manage and scale.

**Examples**

* Simple demonstration of defining a model, writing a view function/class, and rendering a template with data.

**Scenario: A simple student management website**

**Goal:** Show a list of all students on a web page.

**Step 1: Model (Data Layer)**

# models.py

from django.db import models

class Student(models.Model):

name = models.CharField(max\_length=50)

age = models.IntegerField()

grade = models.CharField(max\_length=5)

* Here, Student is our **model**.
* It defines **what data exists**: name, age, grade.

**Step 2: View (Business Logic / Controller)**

# views.py

from django.shortcuts import render

from .models import Student

def student\_list(request):

1. Fetch all students from database

students = Student.objects.all()

2.# Pass data to template for rendering

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

**What’s happening here?**

1. The view **receives the user request** (request).
2. It **asks the model** for all students (Student.objects.all()).
3. It **decides** what data to send to the template.
4. It **renders** the template with the data.

Think of the view as the **chef in a restaurant**:

* Receives the order (request)
* Gathers ingredients (data from model)
* Prepares the dish (passes to template)

**Step 3: Template (Presentation Layer)**

<!-- students.html -->

<h1>Student List</h1>

<ul>

{% for student in students %}

<li>{{ student.name }} - {{ student.grade }}</li>

{% endfor %}

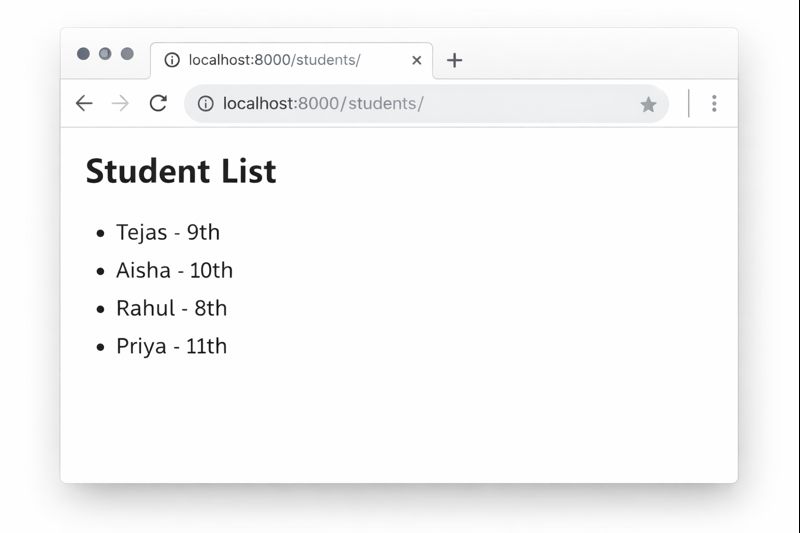
</ul>

* Shows the data in a nice list.
* View sent students to this template → template **renders it for the user**.

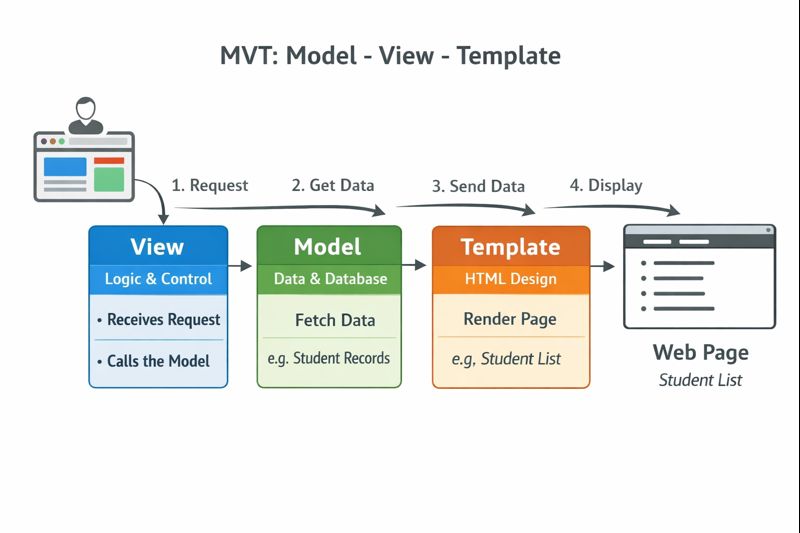
**Flow Recap**

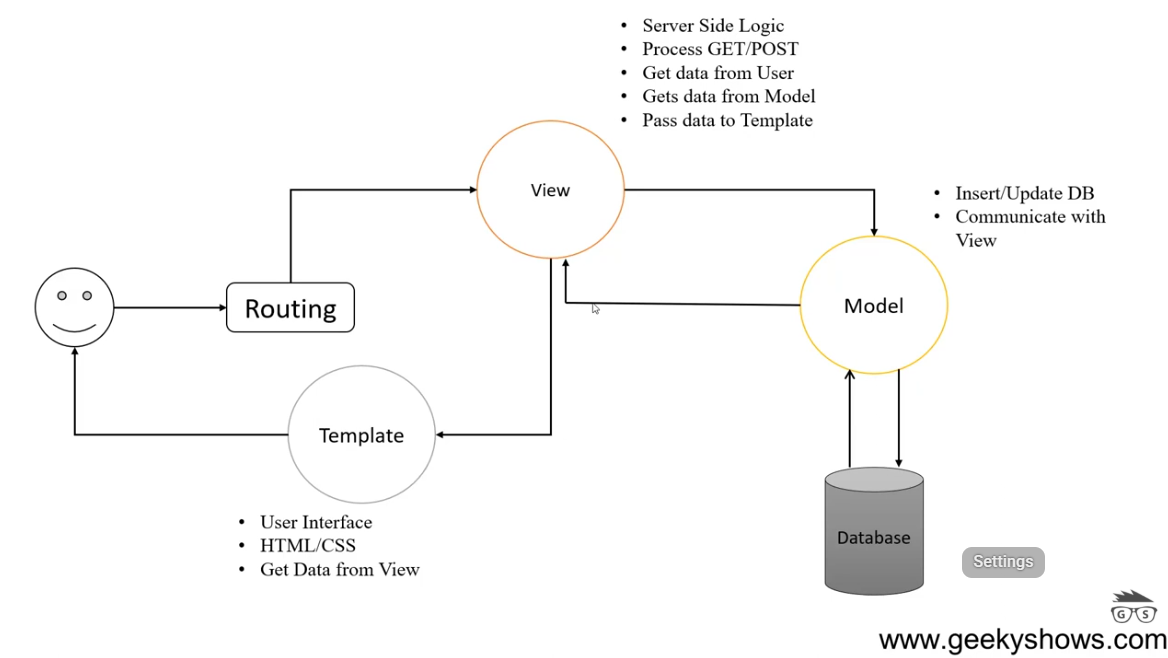
1. **User requests** /students/
2. **View** (student\_list) runs
3. **Model** fetches all students
4. **View** passes data to template
5. **Template** shows the list of students on the web page

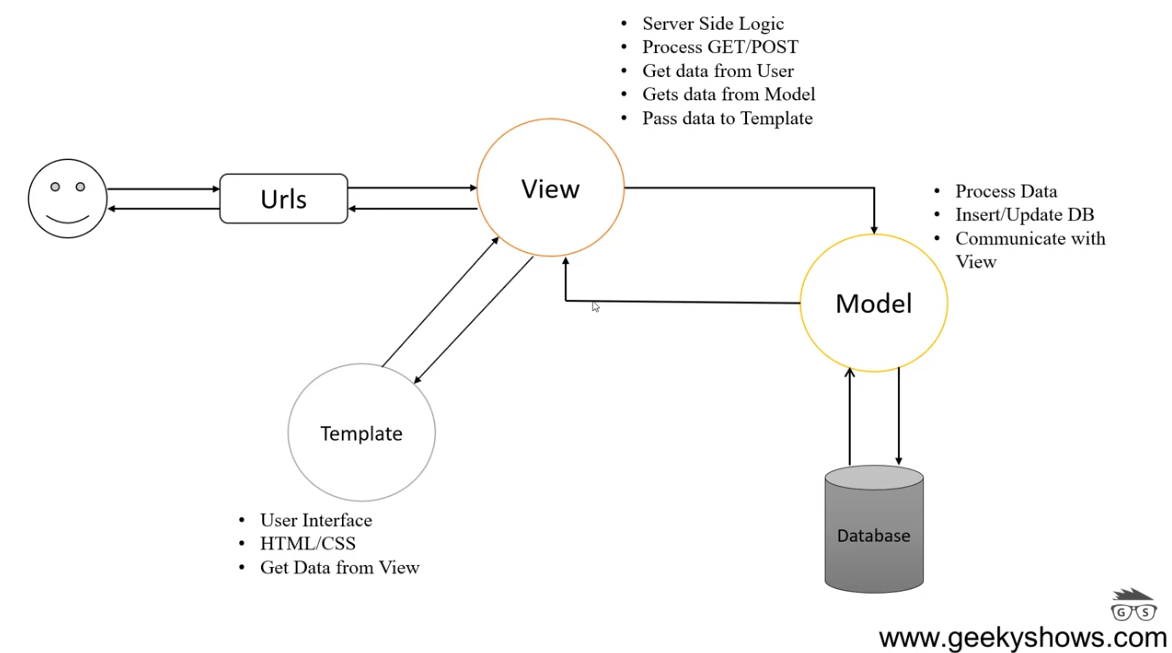
**OUTPUT:**



**simple digrams** showing **Model → View → Template flow:**



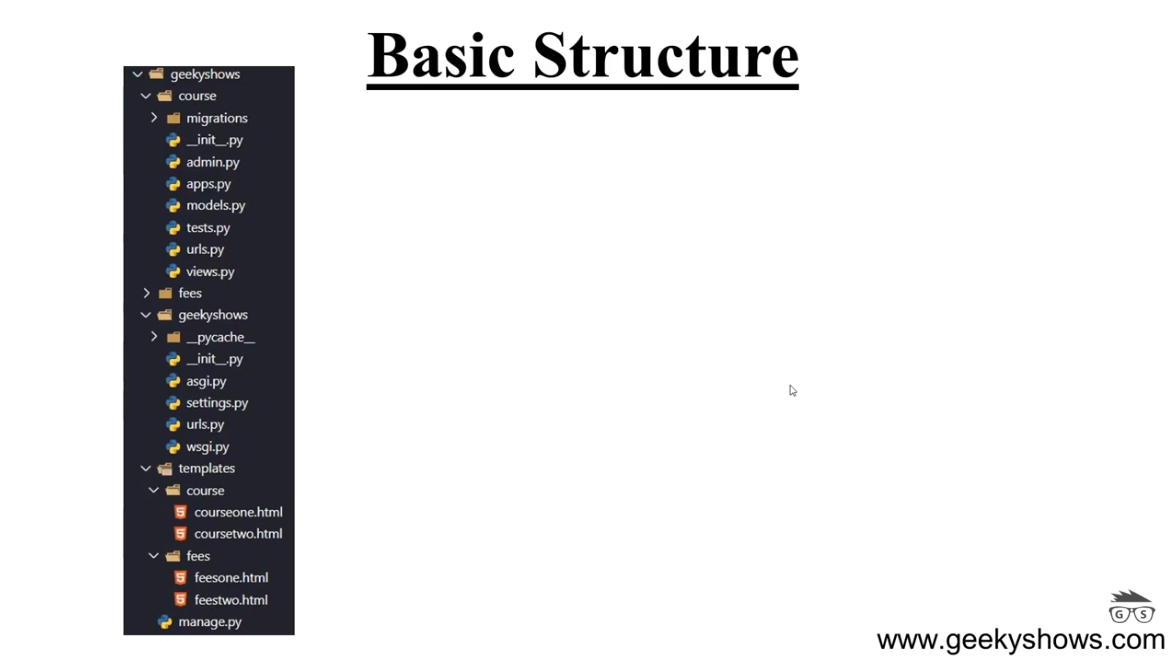




**Why we Use MVT:**

* Organized code.
* Independent block.
* Reduces the complexity of web Applications.
* Easy to modify.
* Easy to maintain.

**Basic Structure Of MVT:**

****

| **Feature** | **General Framework** | **Web Framework** |
| --- | --- | --- |
| **Purpose** | To provide a structure for *any* software. | To simplify the creation of *web* apps. |
| **Primary Focus** | Logic, performance, and architecture. | HTTP, Cookies, Sessions, and HTML. |
| **Environment** | Can run anywhere (OS, Embedded systems). | Runs on a Browser or a Web Server. |
| **Example** | **.NET** (General purpose) | **ASP.NET Core** (Web-specific) |

Django

**What is Django?**

Django is a **high-level Python web framework** designed for rapid development and clean, pragmatic design. Often described as a "batteries-included" framework, it provides almost everything you need to build a website "out of the box," including user authentication, an admin interface, and database management.

Its core philosophy is **DRY (Don't Repeat Yourself)**, which focuses on minimizing code duplication. Instead of reinventing the wheel for every project, Django gives you a solid foundation so you can focus on writing your app's unique features.

**Requirements to Learn Django**

To avoid "fighting" the framework, it is highly recommended to have a grasp of a few key areas before diving in.

**1. Python Fundamentals (Essential)**

Since Django is written in Python, you need a solid foundation. You don't need to be an expert, but you should understand:

* **Basic Syntax:** Variables, loops (for, while), and conditional statements (if, else).
* **Data Structures:** Lists, Dictionaries, and Tuples.
* **Functions:** How to define them and use arguments (\*args, \*\*kwargs).
* **Object-Oriented Programming (OOP):** This is crucial. You should understand **Classes**, **Inheritance**, and **Instances**, as Django uses them for almost everything (Models, Views, etc.).

**2. Basic Web Technologies**

You’ll be building websites, so you need to know what they are made of:

* **HTML & CSS:** You need to know how to structure a page and style it. Django handles the logic, but the browser still renders HTML.
* **HTTP Basics:** A general understanding of how a browser talks to a server (GET vs. POST requests) will make the "Request-Response" cycle in Django much clearer.

**3. Basic Database Concepts**

Django uses an **ORM (Object-Relational Mapper)**, which means you write Python code to interact with your database instead of SQL. However, it helps to understand:

* What a **Table**, **Row**, and **Primary Key** are.
* Relationships between data (One-to-One, One-to-Many, Many-to-Many).

**4. Command Line Basics**

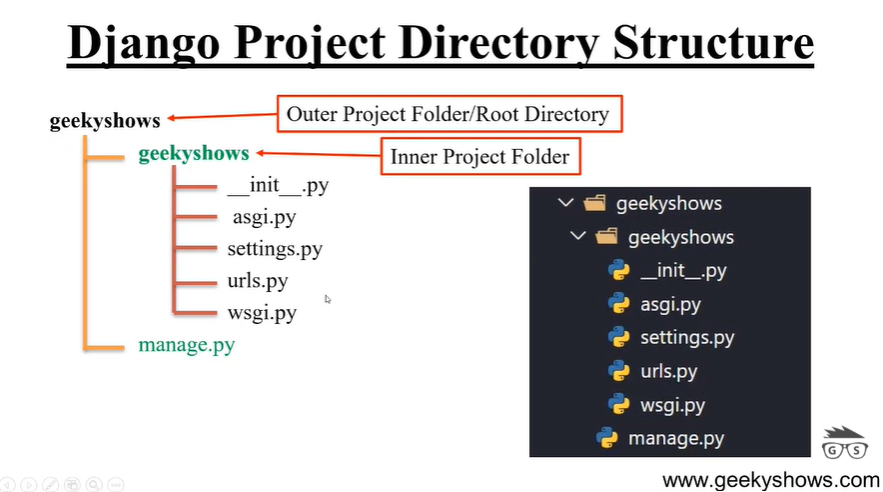
You will spend a fair amount of time in the terminal/command prompt to:

* Create projects and apps.
* Run the development server.
* Manage "Migrations" (updates to your database).

| **Task** | **Command Example** |
| --- | --- |
| **Start a Project** | django-admin startproject myproject |
| **Create an App** | python manage.py startapp blog |
| **Run the Server** | python manage.py runserver |
| **Apply Database Changes** | python manage.py migrate |

**Comparison of Learning Paths**

| **Feature** | **Beginner Path** | **Accelerated Path** |
| --- | --- | --- |
| **Python Prep** | 2-4 weeks (Fundamentals + OOP) | 1 week (Basics + syntax) |
| **Project Focus** | Simple Blog or To-Do list | Custom API or E-commerce |
| **Outcome** | Strong understanding of logic | Fast deployment of features |

**Django Project Directory Structure:**

how to install Django through seperate virtual envirnment and how to install it globally and cammands to create a project?

Installing Django using a virtual environment is the professional standard because it keeps your computer's "global" Python clean and prevents version conflicts between different projects.

Here is the breakdown of how to install it both ways and how to kickstart your first project.

**1. Installation via Virtual Environment (Recommended)**

This method creates an isolated "bubble" for your project.

**Step 1: Create the Virtual Environment**

Navigate to your project folder in your terminal and run:

* **Windows:** python -m venv .venv
* **macOS/Linux:** python3 -m venv .venv

**Step 2: Activate the Environment**

You must activate it so your terminal knows to use the "bubble" instead of the system Python.

* **Windows (CMD):** .venv\Scripts\activate
* **Windows (PowerShell):** .\.venv\Scripts\Activate.ps1
* **macOS/Linux:** source .venv/bin/activate

**Note:** Once activated, you will usually see (.venv) appear before your command prompt.

**Step 3: Install Django**

Now that the environment is active, install Django:

Bash

pip install django

**2. Global Installation (Not Recommended)**

Installing globally makes Django available everywhere on your system, but it can lead to issues if you ever need to use a different version of Django for a future project.

To install it globally, simply run the command **without** activating a virtual environment:

* **Windows:** pip install django
* **macOS/Linux:** pip3 install django (You may need to use sudo pip3 install django if you encounter permission errors).

**3. Commands to Create and Start a Project**

Once Django is installed (preferably in your active virtual environment), use these commands to build your project structure.

**Create the Project**

This creates a folder with the basic Django configuration files.

Bash

django-admin startproject myproject

*(Replace myproject with your desired name).*

**Navigate into the Project**

Bash

cd myproject

**Run the Development Server**

To see if it worked, start the built-in server:

Bash

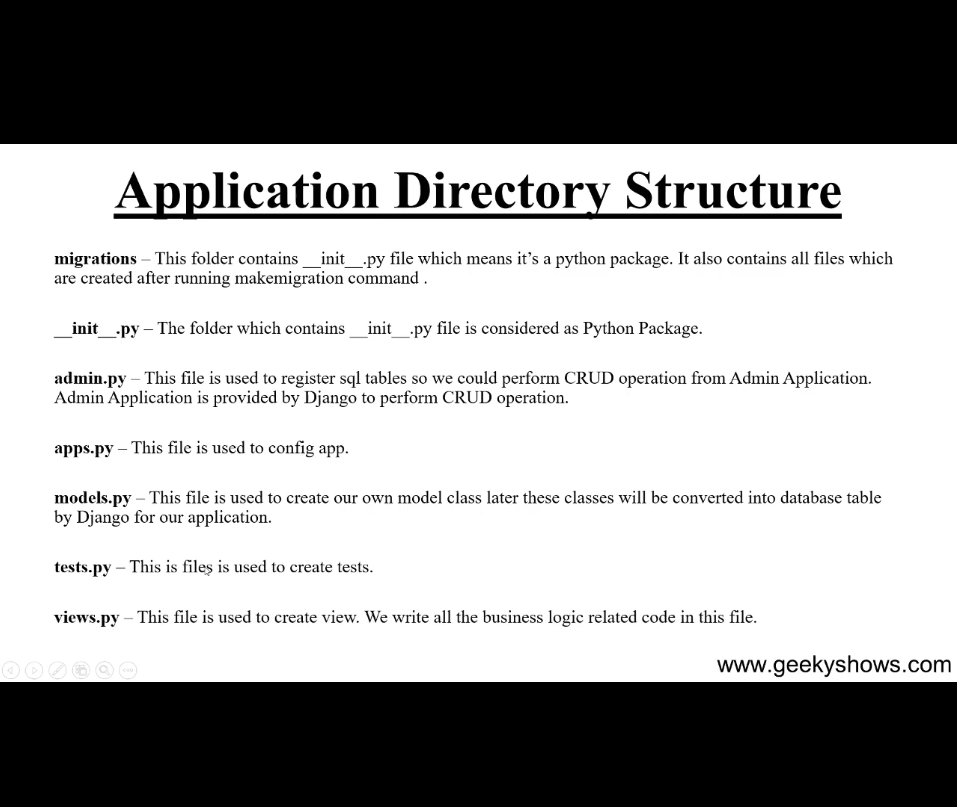
python manage.py runserver

Now, open your browser and go to http://127.0.0.1:8000/. You should see the Django "Rocket" success page!

GEEKY STEPS

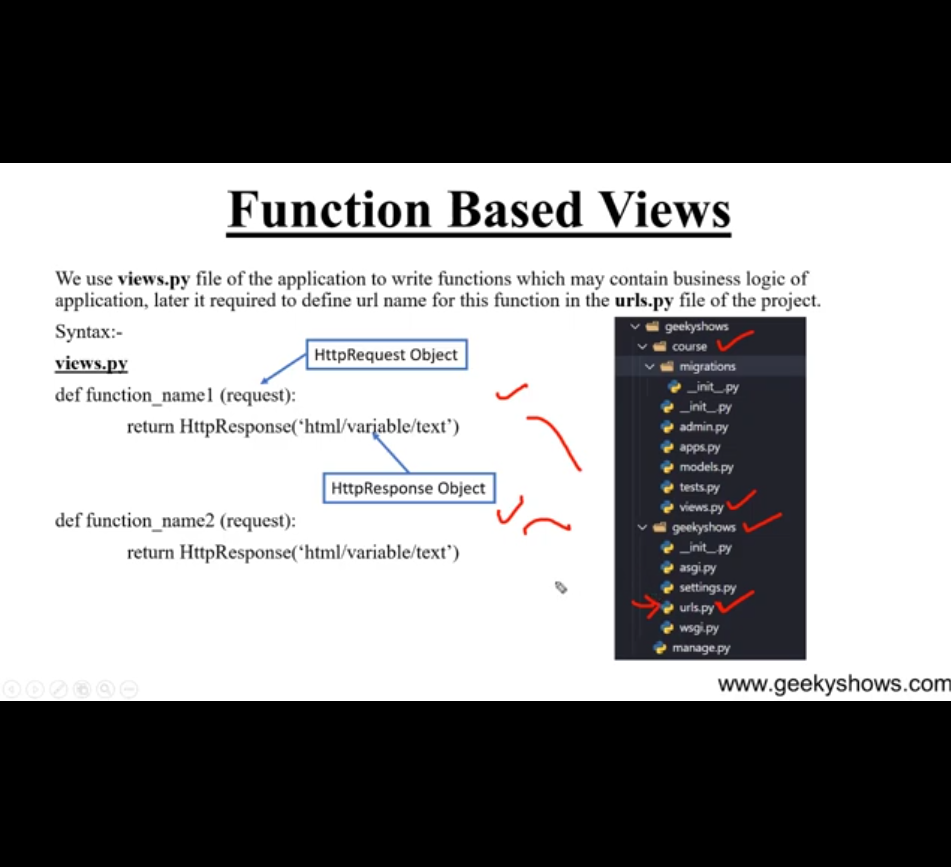


Application Directory Structure:

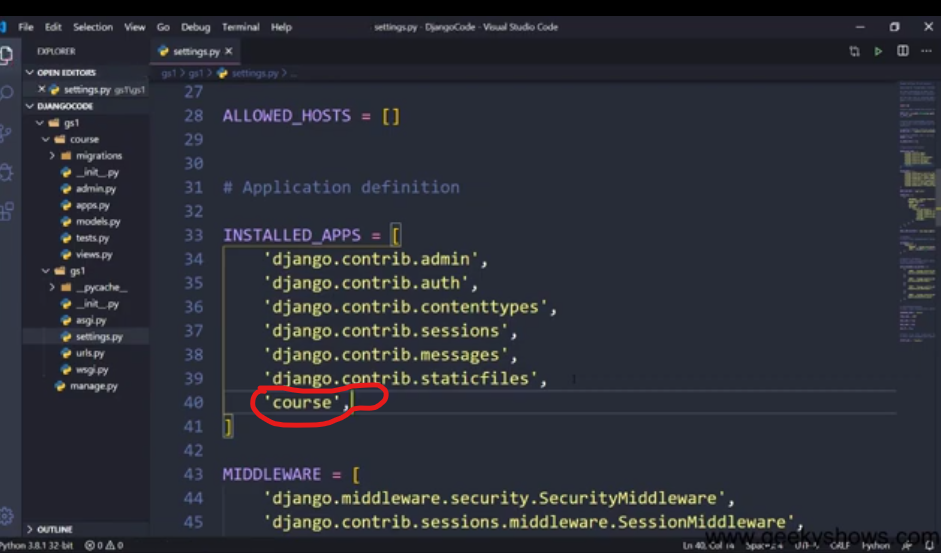


When we start editing and start create views:

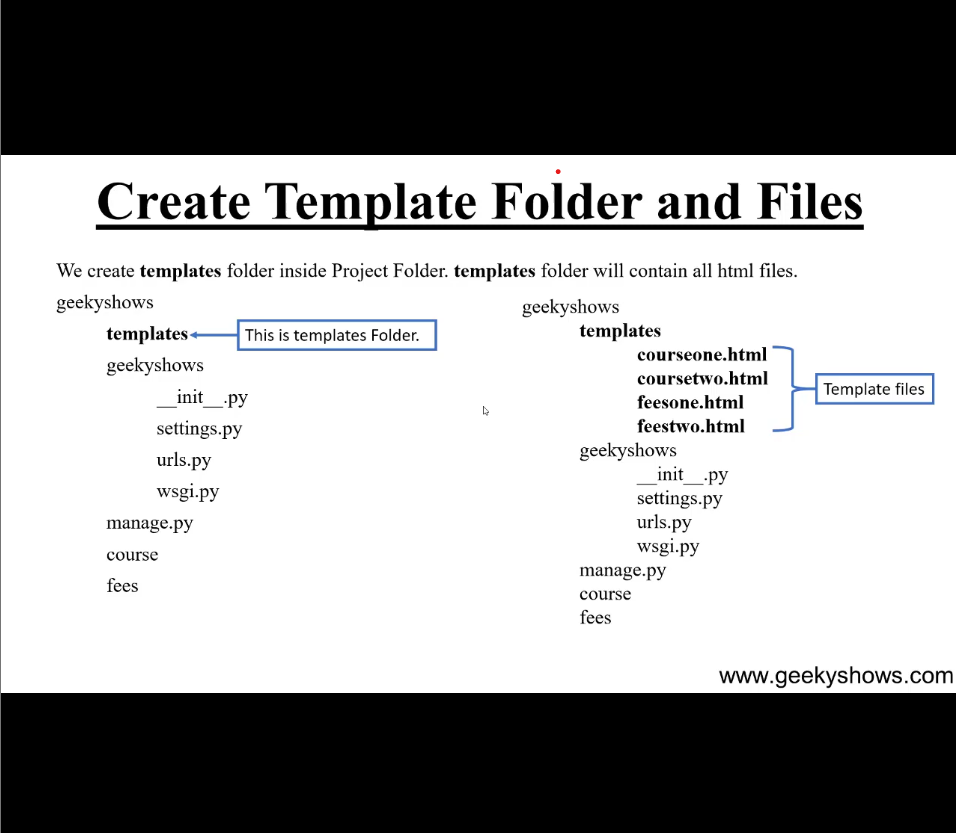
* 1. Functions based views:



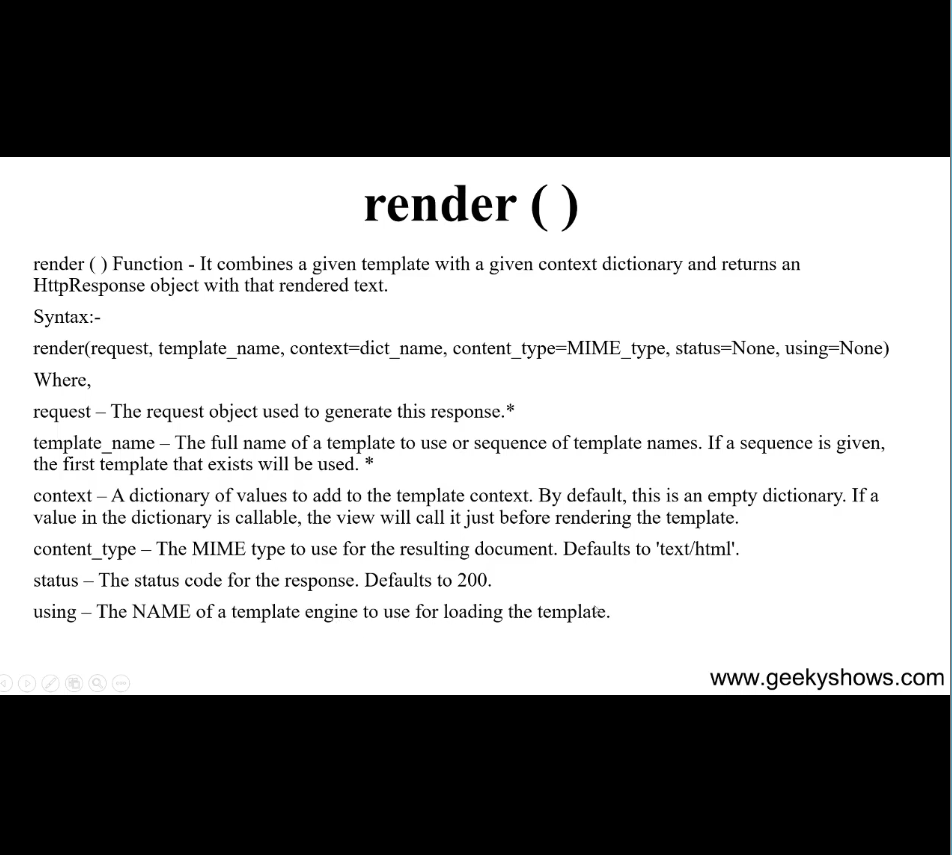
Understand the Flow What we Have to do:

1. We will create a project. (django-admin startproject projectName)
2. Then we will create an Application.(manage.py startapp applicationName)
3. Then we have to install that Application in settings.py file in project folder exact like course name Application File is installed in given below reference image.
4. 
5. Then we will generate views Function views.py file in your Application Folder.
6. Then we write URLs for these Function in urls.py file present in project called Outer URLs and another in application urls.py called inner urls file in its folder that application you generated at second step.

How to create templates and it’s structure



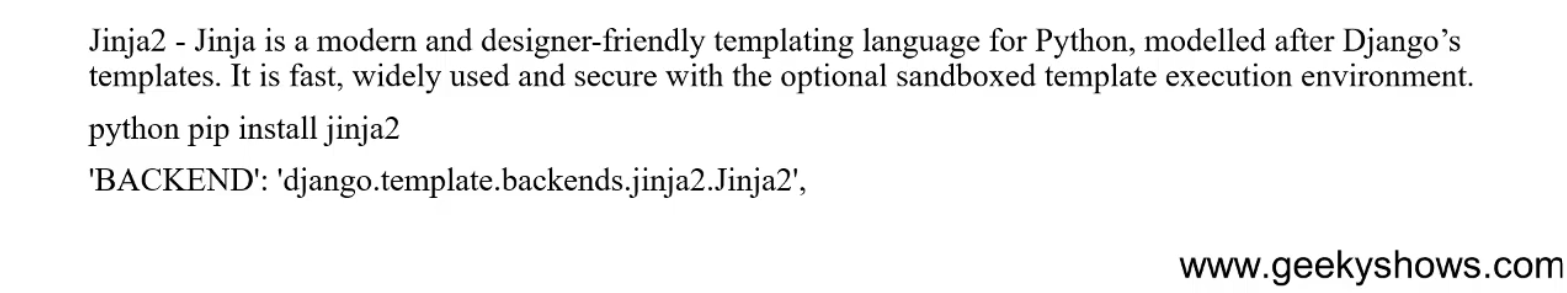
Understand the Render Function



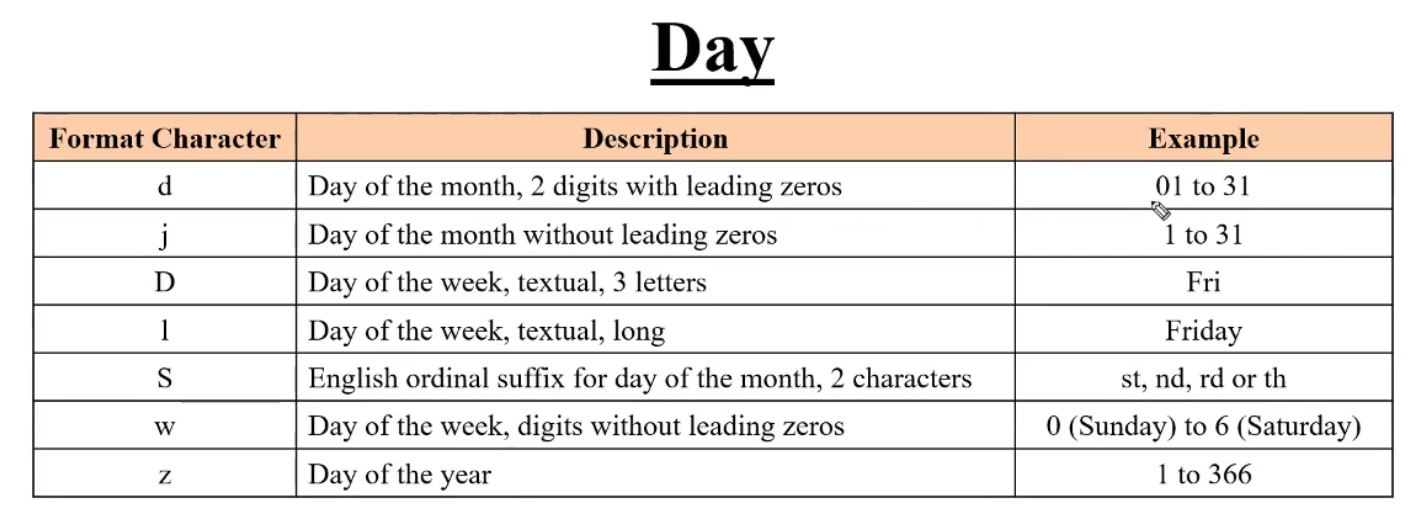
Dynamic Template Files

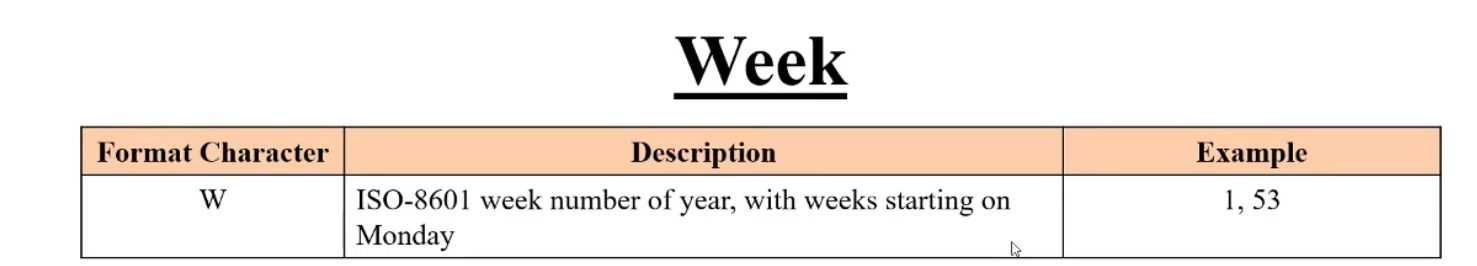


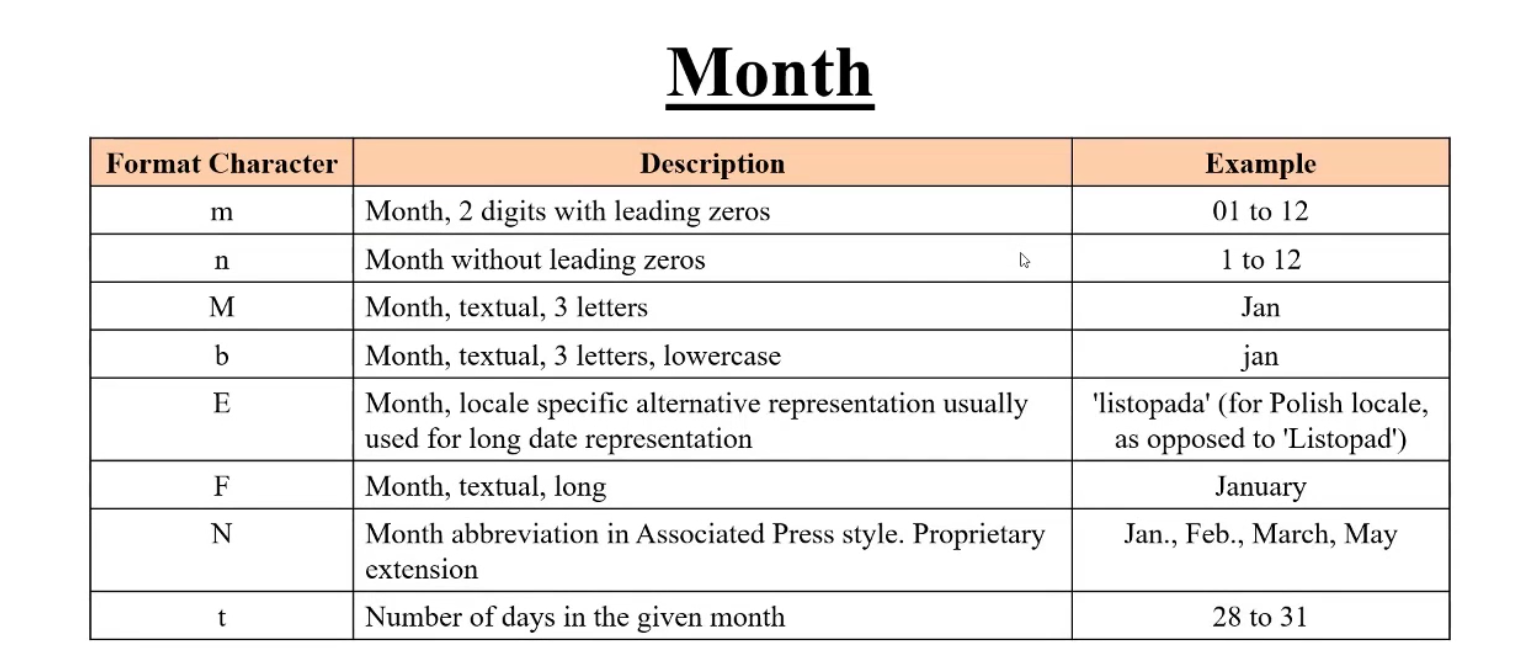
Jinja-2 (A new advance templating language of python)

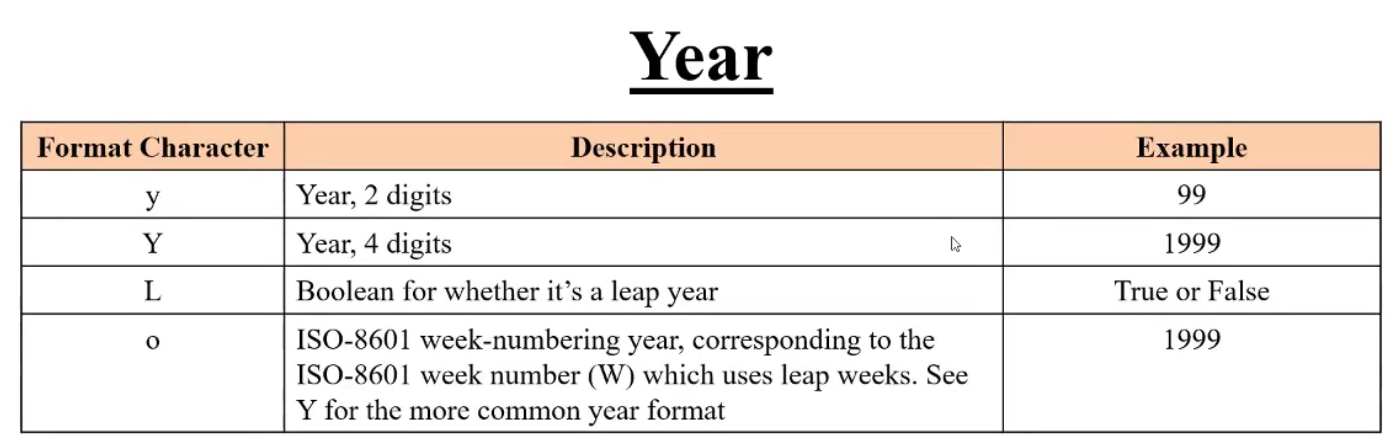


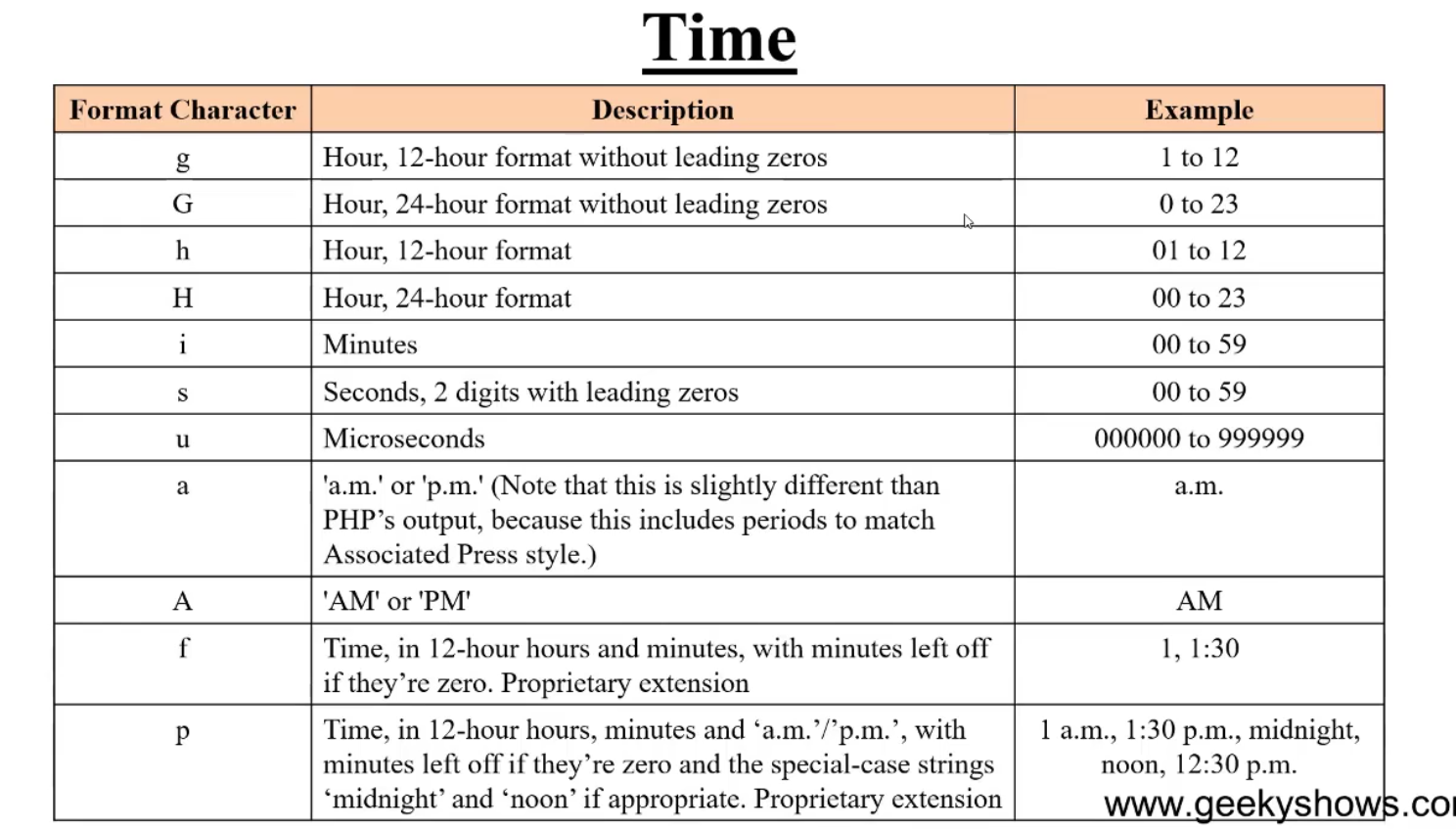
Date and Day Filters in Django Template Language

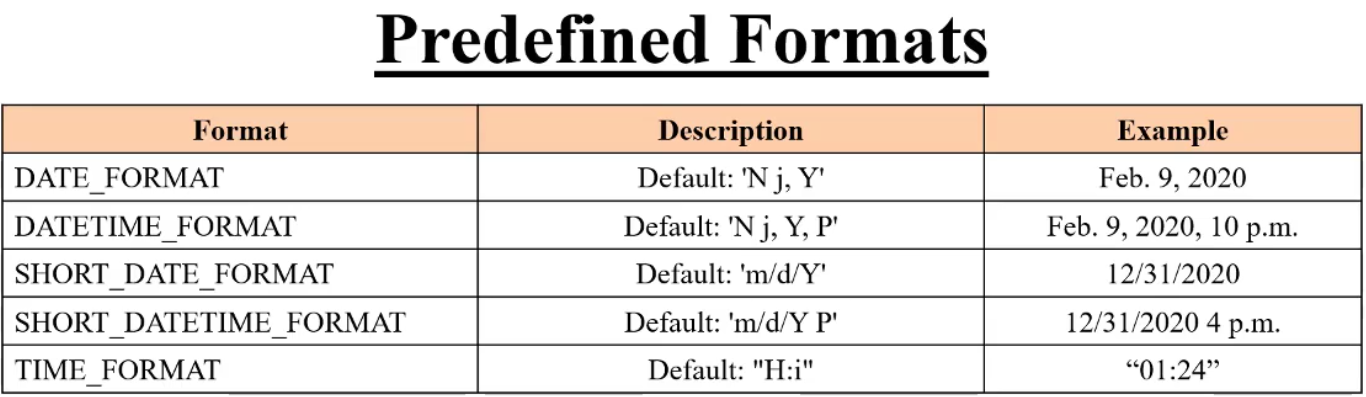


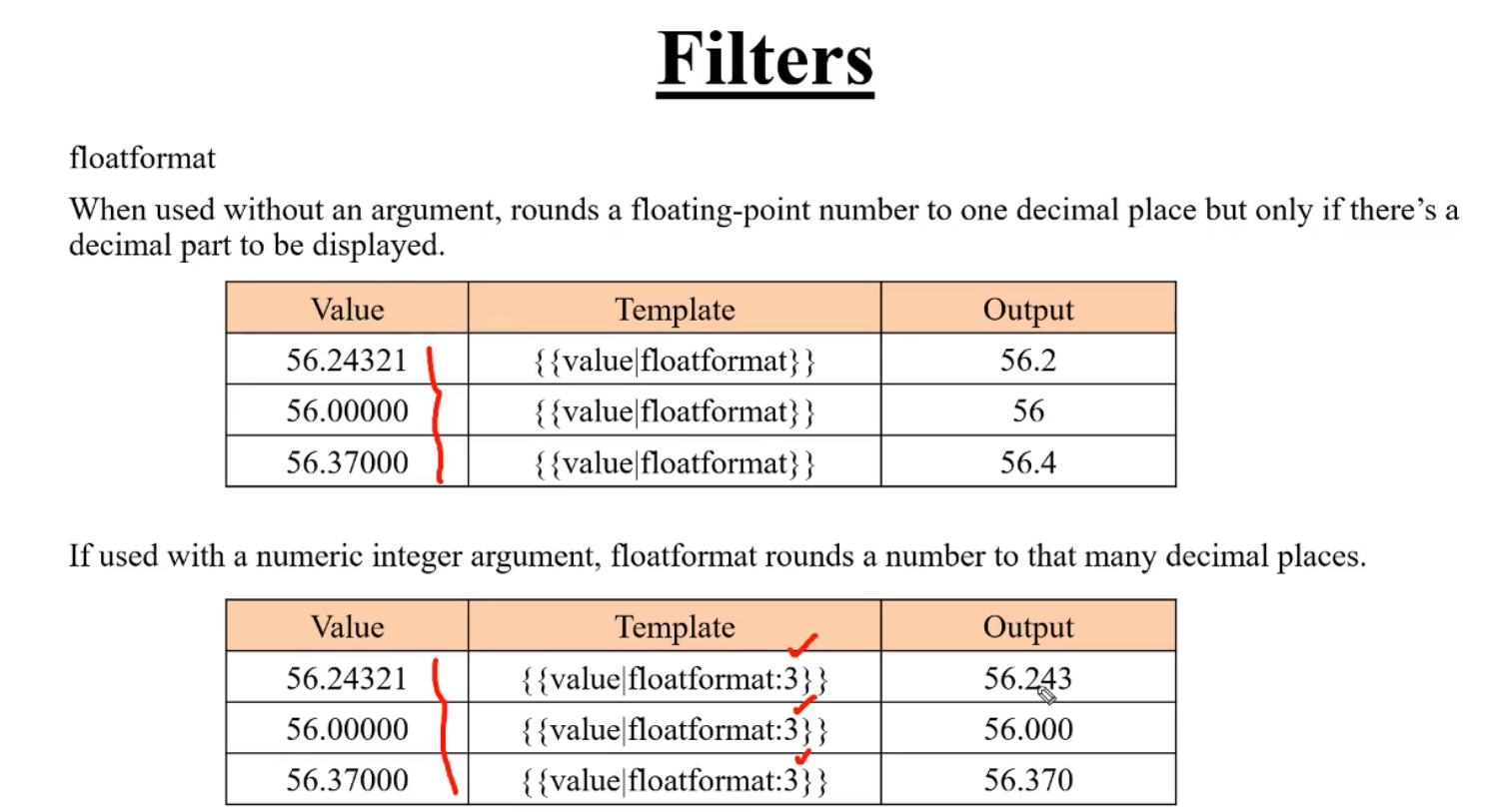


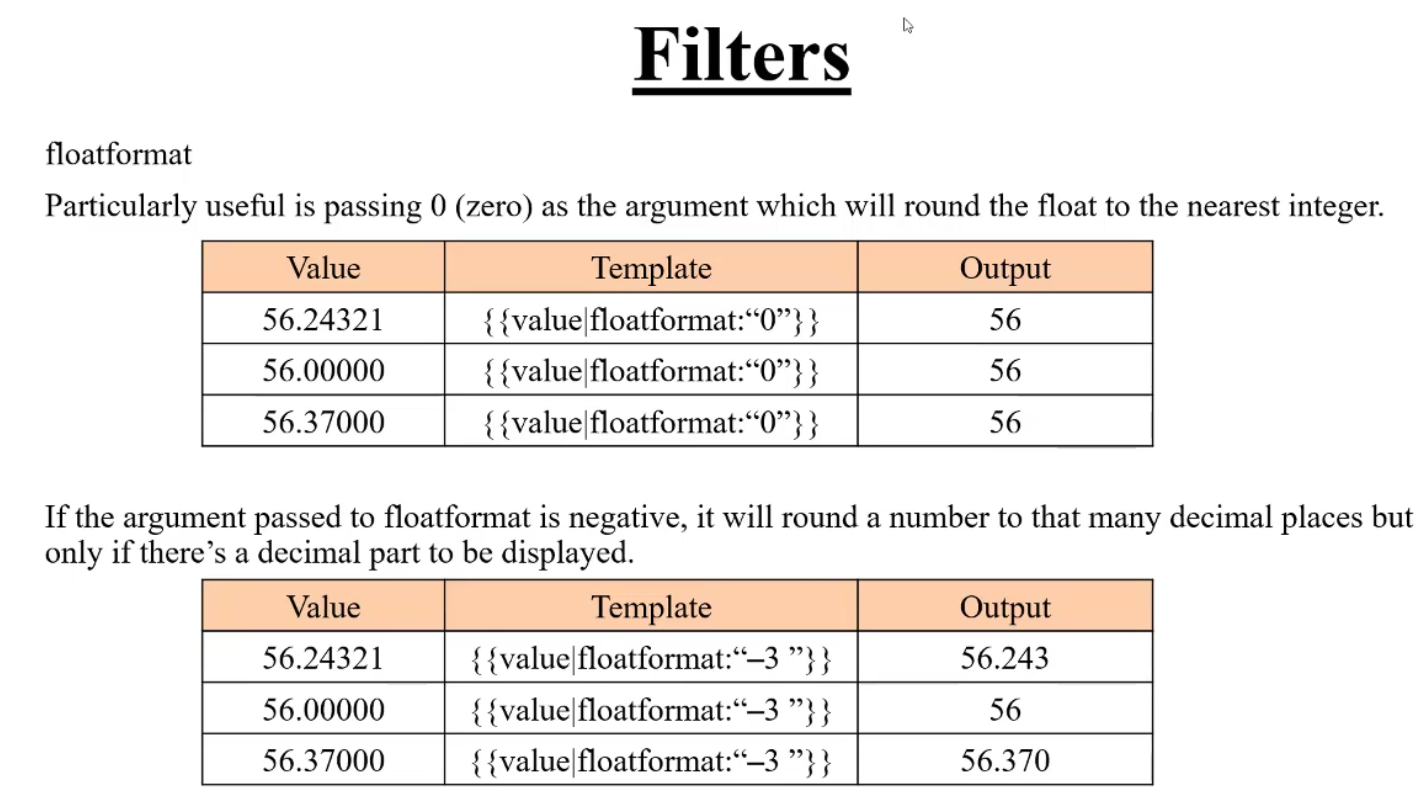












Pre defined For Loop

