Mastering Django: Projects for Beginners

A Beginner's Journey to Django Mastery: Projects and Beyond

Aditya Dhandi

Also by Aditya Dhandi

- Python Course - Learn Python From Scratch : 

Index

Chapter 1: Introduction

Welcome to the immersive world of “Mastering Django: Projects for Beginners”—a book that places projects at the core of your journey into web development using the powerful Django web framework.

This book is meticulously crafted to be hands-on, ensuring your comprehension through an abundance of examples and exercises in every chapter. Whether you're delving into the content sequentially or employing it as a targeted reference, consider this book your steadfast companion on your Django expedition.

Whether you're on the brink of creating a personal project, launching a startup, or simply navigating the expansive realm of web development, Django stands as your ideal companion. With “Projects for Beginners” serving as your navigator, you'll swiftly evolve into a developer capable of crafting impactful web applications that leave a lasting imprint on the digital landscape.

Brace yourself to convert your aspirations in web development into tangible achievements as we commence our exciting Django adventure together!

Just as our approach mirrors the successful structure of “Projects for Beginners”, where you construct progressively intricate web applications, from a humble "Hello, World" app to a sophisticated Job Listing application with advanced features—forms, user accounts, custom user models, email integration, foreign keys, authorization, permissions, and beyond—you can expect a similar journey in “Mastering Django: Projects for Beginners”.

By the final pages of this book, you'll emerge with the confidence to create your own Django projects from scratch, armed with the latest best practices in web development. Django, a free, open-source web framework written in Python and embraced by millions of developers worldwide, seamlessly caters to both beginners and advanced programmers.

While robust enough for the largest websites globally—Instagram, Pinterest, Bitbucket, Disqus—Django remains flexible, making it an excellent choice for early-stage startups and personal projects.

This book is a living document, regularly updated to align with the latest versions of Django(4.0) and Python (3.11x). We adopt Pipenv, the officially recommended package manager by Python.org, for handling Python packages and virtual environments.

Throughout our exploration, we adhere to modern best practices from the Django, Python, and web development communities, placing a strong emphasis on thorough testing—a fundamental aspect of building robust and scalable applications. So, let's embark on this journey, leveraging the collective wisdom and experiences of the web development community to master the art of Django.

What You'll Learn

In the upcoming sections, I'll guide you through the Django framework in a systematic manner. We'll initiate by configuring your development environment and initiating your inaugural Django project. Following this, we'll delve into subjects such as models and databases, views and templates, and the fundamental principles required to construct a web application from the ground up.

We'll explore aspects like forms, authentication, and user management, ensuring the functionality of your application while prioritizing security and user-friendliness. As you advance, you'll confront more intricate subjects such as working with static and media files, employing class-based views, navigating the Django ORM, conducting tests, debugging, and executing deployment.

I'll provide you with the tools and insights essential for crafting your unique application, and we'll also investigate supplementary functionalities, including real-time capabilities.

Upon completing this resource, you'll not only possess a robust understanding of Django but also the assurance to initiate your web development endeavors. Armed with adept practices and optimization techniques, you'll be prepared to fabricate efficient, scalable, and secure web applications.

What is Django

Django is a high-level, open-source web framework for building web applications using the Python programming language.

It follows the Model-View-Controller (MVC) architectural pattern, which is often referred to as Model-View-Template (MVT) in Django's context.

Django's primary goal is to simplify and speed up the process of developing web applications by providing a robust set of tools and conventions for common web development tasks.

Here are some key features and components of Django:

- **ORM (Object-Relational Mapping):** Django includes a powerful ORM that allows developers to define their data models using Python classes. These models are then translated into database tables, and developers can interact with the database using Python code, rather than writing SQL queries directly.

- **Admin Interface:** Django comes with an automatic admin interface, which can be customized to manage your application's data models. This is especially helpful for content management and administrative tasks.

- **URL Routing:** Django uses a URL dispatcher to map URLs to views. This allows for clean and flexible URL routing, making it easy to define how different parts of your application respond to different URL patterns.

- **Template Engine:** Django has a template engine that allows developers to define the structure and layout of HTML pages in a clean and efficient way, separating the presentation layer from the business logic.

- **Authentication and Authorization:** Django provides built-in tools for user authentication and authorization, making it straightforward to add user accounts and manage permissions.

- **Security:** Django includes various built-in security features to help developers protect their applications against common web vulnerabilities, such as cross-site scripting (XSS) and SQL injection.

- **Middleware:** Django allows you to define middleware components that can process requests and responses globally, enabling tasks like authentication, logging, and more.

- **Forms and Validation:** Django provides a forms framework that simplifies the creation and handling of HTML forms, including form validation.

- **Database Support:** Django supports multiple databases, including PostgreSQL, MySQL, SQLite, and Oracle, allowing developers to choose the database that best suits their needs.

- **REST Framework:** While not part of Django core, the Django REST framework is a popular third-party package for building robust RESTful APIs using Django.

- **Community and Ecosystem:** Django has a large and active community, which means there are many reusable packages and extensions available to help streamline development.

Django's "batteries-included" philosophy means it comes with many built-in features, which can accelerate development while maintaining best practices. It's an excellent choice for building a wide range of web applications, from simple websites to complex, data-driven platforms.

Why Django

Why choose Django as your trusty steed on the vast and ever-evolving journey of web development? In a digital world filled with countless frameworks and tools, Django shines as a beacon of innovation, practicality, and sheer magic. Here's why Django is the sorcerer's stone of web development:

- **Pythonic Elegance:** Django is built on Python, a language known for its readability and simplicity. This means you'll spend less time deciphering cryptic code and more time crafting your web applications. With Django, you're working in a programming language that feels like poetry.

- **Batteries Included:** Django doesn't just offer a web framework; it comes fully equipped with a treasure trove of tools and utilities. From an admin panel for data management to authentication and security features, Django provides everything you need to create robust web applications without the hassle of piecing together numerous third-party libraries.

- **Rapid Development:** Django follows the "Don't Repeat Yourself" (DRY) principle. It automates many common development tasks, allowing you to focus on what makes your application unique. This accelerated development process means you can bring your ideas to life faster than ever before.

- **Scalability and Flexibility:** Django is not just for small projects. It scales seamlessly, making it suitable for both startups and large enterprises. Its modular architecture allows you to use only the components you need, providing flexibility to tailor your development stack.

- **Community and Documentation:** Django boasts a vibrant and supportive community of developers, readily available to answer questions and provide guidance. Its extensive documentation serves as a treasure map, guiding you through every corner of the framework.

- **Security Enchanted:** Security is a paramount concern in the digital realm, and Django takes it seriously. The framework comes equipped with built-in defenses against common web vulnerabilities, such as SQL injection and cross-site scripting (XSS). This means you can focus on building your application while Django guards it against malicious sorcery.

- **Battle-Tested and Trusted:** Django has been used to power countless websites, from small personal blogs to major platforms like **Instagram** and **Pinterest**. Its track record speaks to its reliability and scalability.

- **Versatile Ecosystem:** Django's ecosystem extends beyond web development. It offers tools and libraries for tasks like creating RESTful APIs with the Django REST framework, building real-time applications with Django Channels, and integrating with other Python packages for data analysis, machine learning, and more.

- **Open Source Magic:** Django is open source, meaning it's free to use and has a community that continually contributes to its improvement. This open nature encourages innovation and ensures that Django remains relevant and up-to-date.

- **Future-Proof:** Django evolves with the ever-changing landscape of web development. Its maintainers and community work tirelessly to keep it in sync with modern practices and technologies, making it a reliable choice for the long haul.

In summary, Django isn't just a web framework; it's an enchanting toolkit for web sorcerers. With its Pythonic elegance, comprehensive features, and strong community support, Django empowers developers to create web applications that are not only functional but truly magical. So, embrace the Django magic, and let your web development journey begin!

Django Vs Flask

Django and Flask are both popular Python web frameworks, but they have different philosophies, use cases, and levels of complexity. The choice between Django and Flask depends on your project requirements and your development preferences. Here's a comparison of Django and Flask:

Django:

Philosophy: Django follows the "batteries-included" philosophy, which means it provides a comprehensive set of tools and features for web development out of the box. It includes an ORM, an admin interface, authentication, and more.

Complexity: Django is a high-level framework that enforces a specific project structure and follows the Model-View-Controller (MVC) architectural pattern, which is often referred to as Model-View-Template (MVT) in Django's context. This makes it suitable for complex, large-scale web applications where a lot of built-in functionality is required.

Built-in Features: Django includes many built-in features like an admin interface, user authentication, database migrations, and an ORM. This can save time and effort for developers.

Learning Curve: Due to its comprehensive nature, Django can have a steeper learning curve for beginners. However, it provides clear guidelines and conventions for development, which can be beneficial in the long run.

Community and Ecosystem: Django has a large and active community, along with a rich ecosystem of third-party packages and extensions.

Use Cases: Django is often chosen for large-scale, data-driven applications, content management systems (CMS), and projects that require a lot of built-in functionality.

Here are some popular websites and platforms that use Django:

- Instagram
- Pinterest
- The Washington Post
- Eventbrite
- Disqus
- NASA
- Bitbucket
- Mozilla Support

- The Onion
- The Guardian
- Prezi
- Mahalo

Flask:

Philosophy: Flask follows the "micro" framework philosophy, which means it provides the basic tools and leaves many decisions to the developer. The flask is minimalistic and lightweight.

Complexity: Flask is simpler and more flexible compared to Django. It doesn't impose a specific project structure or architecture, allowing developers to choose their preferred components and libraries.

Extensibility: Flask is highly extensible and allows developers to add only the components they need. This results in more control and flexibility but also requires more decisions on the developer's part.

Learning Curve: Flask has a lower learning curve, making it a good choice for beginners or projects that need to be developed quickly.

Community and Ecosystem: Flask has a vibrant community and a range of extensions, but its ecosystem is not as extensive as Django's.

Use Cases: Flask is often chosen for smaller to medium-sized projects, RESTful APIs, microservices, and projects where developers want more control over the components they use.

In summary, Django is a full-featured framework with a lot of built-in functionality, making it suitable for complex and feature-rich applications. Flask, on the other hand, is lightweight and flexible, giving developers more control over the components they use, making it a great choice for smaller projects or when simplicity and customizability are priorities. The choice between the two depends on the specific needs of your project and your development preferences.

Why this Book

"**Mastering Django: Projects for Beginners "** serves as your indispensable companion for honing the skills of web development using Django. Whether you're new to programming or making a transition from another framework, this book is tailored to meet your educational requirements.

Key Features:

Structured Learning Path: Follow a meticulously designed curriculum that seamlessly guides you from fundamental concepts to advanced Django topics, ensuring a well-paced learning experience.

Hands-On Examples: Immerse yourself in real-world examples and practical exercises that solidify your grasp of Django fundamentals. These hands-on activities make intricate concepts understandable and applicable.

Comprehensive Coverage: Delve into the complete Django ecosystem, covering the spectrum from crafting dynamic web applications to understanding databases, authentication, and deployment. Gain a comprehensive understanding of full-stack development.

Problem-Solving Approach: Develop the skills to address common challenges encountered by developers. Detailed explanations and solutions empower you to navigate obstacles independently.

Embark on your Django development journey confidently with "**Django Fundamentals."** This book provides you with the skills and knowledge necessary to construct robust web applications. Unleash the potential of Django and commence the creation of dynamic, scalable, and secure websites today.

Book layout

In this book, you'll encounter various code examples, which we'll identify using the following conventions:

```
# This is Python code
print("Hello, World")
```

To keep things concise, we'll represent unchanged, pre-existing code with dots (...) when, for instance, we're making updates within a function:

```
def this_is_my_function:
    ...
    print("Completed")
```

Throughout the book, we'll also use the command line console extensively. Commands will be preceded by a **>** symbol, following the traditional Unix style:

command prompt

```
> echo "Hello World"
```

The resulting output of a command will be presented without the **>** symbol:

command prompt

```
"Hello World"
```

To simplify comprehension, we'll often display both the command and its output together:

command prompt

```
> echo "Hello World"
Hello World
```

For the complete source code of all examples, please refer to the official GitHub repository.

A Brief Overview of Python

Before we dive headfirst into Django's enchanting world, it's essential to have a basic understanding of the magic wand we'll be wielding—Python. If you're already familiar with Python, consider this a quick refresher; If you're new to it, please have some understanding of Python.

What Is Python?

Python is a versatile, high-level programming language celebrated for its simplicity and legibility. Think of Python as a versatile spellbook, packed with readable and expressive spells (code) that can accomplish a wide range of tasks. It's a language that emphasizes clean and concise code, making it an ideal choice for beginners and seasoned developers alike.

Python's Philosophy

Python follows a set of guiding principles known as "The Zen of Python" or "PEP 20." some of these guiding principles include:

- Readability counts: Code should be easy to read and understand.
- Simple is better than complex: Python encourages simplicity in code design.
- There should be one-- and preferably only one --obvious way to do it: Python promotes a single, clear way of accomplishing tasks.

Python's Strengths

Python's versatility extends to various domains:

- **Web Development:** Django and Flask are popular Python frameworks for building web applications.
- **Data Science:** Python is a go-to choice for data analysis, machine learning, and scientific computing, with libraries like NumPy, Pandas, and sci-kit-learn.
- **Automation:** Python's simplicity makes it perfect for scripting tasks, automating repetitive actions, and working with APIs.
- **Game Development:** Pygame allows you to create 2D games with Python.
- **Scripting:** It's often used for writing scripts to automate tasks on servers and in system administration.

Python Syntax

Python uses a clean and minimalistic syntax. Here's an example:

```
# This is a Python comment

def greet(name):
    message = "Hello, " + name + "!"
    print(message)

greet("Alice")
```

In Python, indentation (whitespace) is significant, and code blocks are defined by colons and indentation levels.

Data Types

Python supports various data types, including

Integers (int)
Floating-point numbers (float)
Strings (str)
Lists (list)
Dictionaries (dict)
Booleans (bool)
Tuples (tuple)
Sets (set)

Control Structures

Python provides control structures like loops and conditionals. Here's an example:

```
for i in range(5):
    if i % 2 == 0:
        print(i,"is even")
    else:
        print(i,"is odd")
```

Functions

You can define functions in Python to encapsulate blocks of code for reuse:

```
def add(x, y):
    return x + y

result = add(3, 5)
```

Libraries and Modules

Python's power extends through its vast library ecosystem. You can import and use modules such as math, random, and others to extend Python's capabilities.

Your Python Toolkit

As you embark on your Django journey, Python will be your trusty wand. You'll cast spells (write code) with it to create the magic of web applications. Don't worry if you're new to Python; we'll guide you through the incantations and reveal the secrets of Django along the way. So, let's step into Python's mystical world together.

Python Resources

If you are new to Python, you can learn Python from the following books:

- Python Crash Course
- Think Python
- The Hitchhiker's Guide to Python

Chapter 2: Setting up a development environment

This chapter guides on configuring your computer effectively for working on Django projects. We begin with an overview of the command line and utilize it to install the latest versions of both Django (4.0), and Python (3.11). Subsequently, we delve into topics such as virtual environments, git, and working with a text editor.
By the conclusion of this chapter, you will be well-prepared to create and modify Django projects with ease.

The Command Line

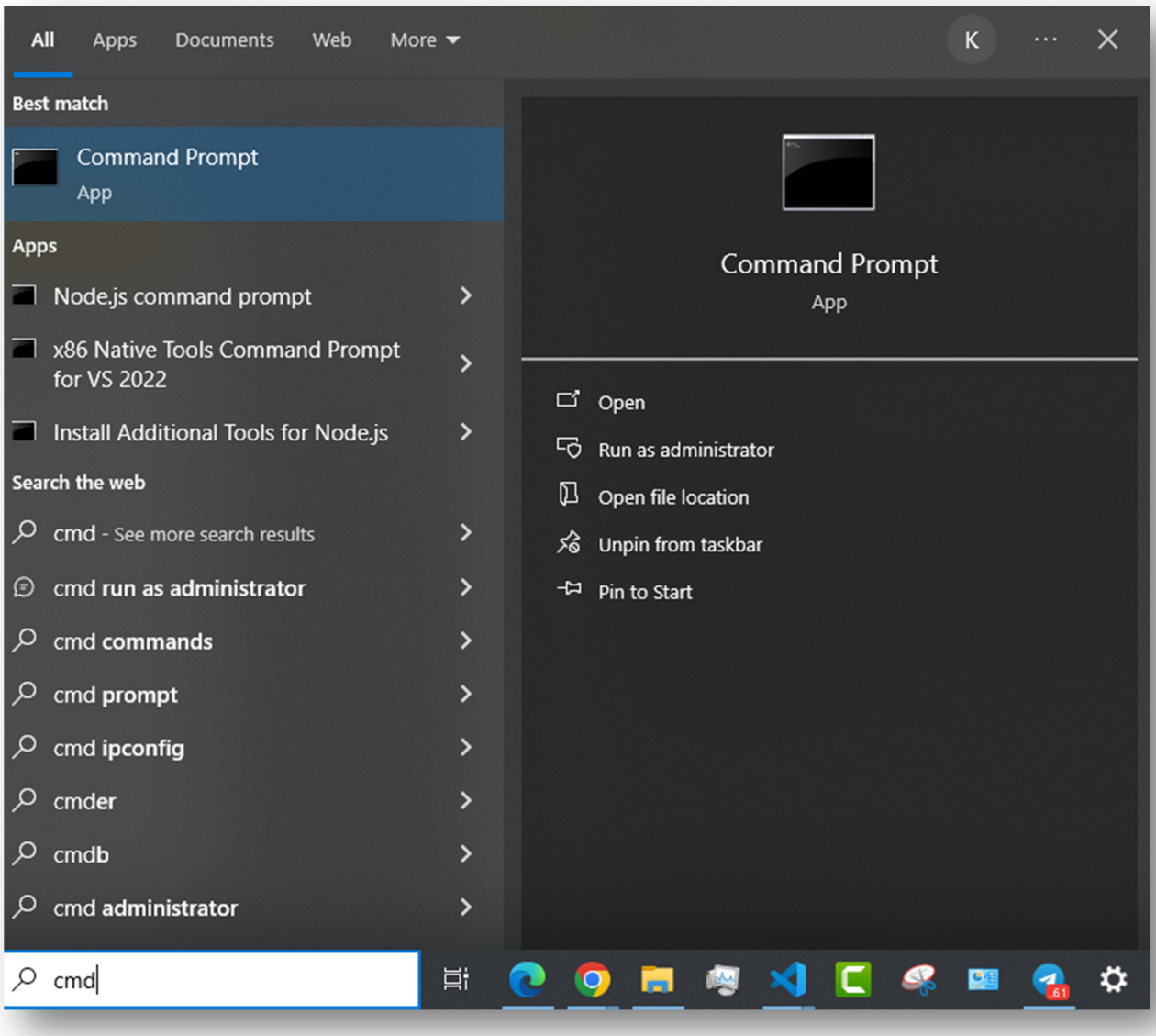
The command line is a potent, text-based interface to your computer. As developers, we will utilize it extensively throughout this guide for installing and configuring each Django project.

On a Mac, you can access the command line via a program called Terminal, located at **/Applications/Utilities.**

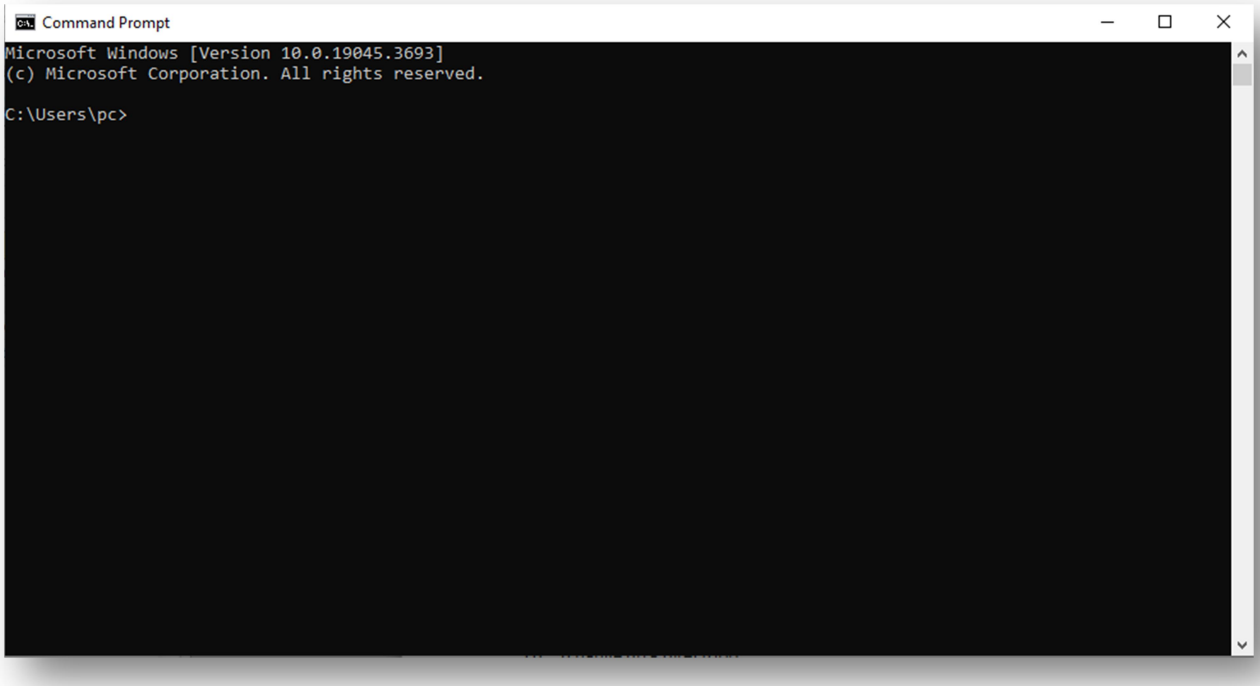
To find it, open a new Finder window, navigate to the Applications folder, scroll down to open the Utilities folder, and double-click the Terminal application.

For Windows users, there is an integrated command line program called Command Prompt. From here on, when we mention the "command line," we mean opening a new console on your computer using Terminal.

To get the command line, go to the search box of your system and type **cmd**. You will see something like this:



Just click on Command Prompt to open the command line.



While there are numerous commands available, we primarily use five commands in Django development:

- **CD** (change down a directory)
- **CD ..** (change up a directory)
- **DIR** (list files in your current directory)
- **MKDIR** (make directory)
- **ECHO>** (create a new file)

Open your command line and try them out. The "**>**" symbol represents our command line prompt for Windows and $ for Mac, and all commands in this guide should be typed after the "**>**" prompt.

For example, assuming you are on a window, let's change into our Desktop directory:

command prompt

```
> cd Desktop
```

Now, let's create a new directory folder using 'mkdir,' navigate into it, and add a new file named 'index.html,':

command prompt

```
> mkdir django_apps
> cd django_apps
> echo> index.html
```

Next, use "dir" to list all the current files in our directory. You will see that there is just the newly created "index.html":

command prompt

```
> dir
12/12/2023  07:46 AM    <DIR>          .
12/12/2023  07:46 AM    <DIR>          ..
12/12/2023  07:46 AM                13 index.html
               1 File(s)             13 bytes
               2 Dir(s)   5,406,769,152 bytes free
```

Advanced developers can efficiently navigate their computers using the keyboard and command line. With practice, using the command line approach is often faster than using a mouse.

While this book provides step-by-step instructions, you do not need to be an expert on the command line. However, developing command-line skills is beneficial for any professional software developer over time.

Two recommended free resources for further study are the Command Line Crash Course and the Codecademy Course on the Command Line.

Install Python 3 on Mac OS X

While Python 2 comes pre-installed on Mac computers by default, Python 3 is not included. You can verify this by entering the following command in your command line console and pressing Enter:

mac command prompt

```
$ python --version
Python 2.7.15
```

To check if Python 3 is already installed, you can run the same command using python3 instead of python:

mac command prompt

```
$ python3 --version
```

If your computer displays a version number like 3.7.x (any version of 3.7 or higher), then Python 3 is already installed.
However, if you encounter an error, you'll need to install Python 3 manually. Here are the steps:

First, install Apple's Xcode package:

mac command prompt

```
$ xcode-select --install
```

Follow the on-screen instructions to complete the installation. Xcode is a large program, so it may take some time.

Next, install the package manager Homebrew using the following command:

mac command prompt

```
$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Confirm that Homebrew is correctly installed by running:

mac command prompt

```
$ brew doctor
```

If you receive a message indicating that your system is ready to brew, you're good to go.
Now, let's install the latest version of Python 3:

mac command prompt

```
$ brew install python3
```

Confirm the Python 3 installation by running:

mac command prompt

```
$ python3 --version
Python 3.11.0
```

To launch a Python 3 interactive shell, which allows you to run Python commands directly, simply type the following in your command line:

mac command prompt

```
$ python3
```

You should see a prompt like:

mac command prompt

```
Python 3.X.X (default, Month Day Year, HH:MM:SS)
[GCC X.X.X] on darwin
Type "help", "copyright", "credits" or "license" for more information.
```

To exit the Python 3 interactive shell at any time, press Control+d (the “Control” and “d” keys simultaneously).
Remember that you can still use Python 2 on your system if needed.

Install Python 3 on Windows

To check if Python is already installed on your Windows system and, if not, to install it, follow these steps:

Step 1: Check if Python is already installed

Press Win + R on your keyboard to open the Run dialog.
Type cmd and press Enter to open the Command Prompt.
In the Command Prompt, type python --version or python -V and press Enter.

If Python is already installed, you'll see the installed version number (e.g., Python 3.8.3).
If Python is not installed, you'll likely see an error message.

Step 2: Download Python (if not already installed)

Open your web browser and go to the official Python website at https://www.python.org/downloads/windows/.

On the Python download page, you'll see the latest version of Python for Windows. Click on the "Download Python X.X.X" button, where "X.X.X" is the version number (e.g., Python 3.9.6).

Step 3: Run the Python Installer

Once the installer is downloaded, locate the downloaded file (usually in your Downloads folder) and double-click it to run the installer.

In the Python installer window, make sure to check the box that says "Add Python X.X to PATH." This option is essential for easily running Python from the Command Prompt.

Click the "Customize installation" button if you want to customize the installation further. Otherwise, you can proceed with the default settings.

Click the "Install Now" button to start the installation process.

Step 4: Wait for Installation to Complete

The installer will now install Python on your system. This may take a few moments.

Step 5: Verify Python Installation

After the installation is complete, go back to the Command Prompt by pressing Win + R, typing cmd, and pressing Enter.

Type python --version or python -V again and press Enter.

If Python was installed successfully, you should see the version number you selected during installation.

Step 6: Test Python

You can also run a simple Python command to test if it's working. In the Command Prompt, type python and press Enter. You should see a Python shell prompt (usually >>>).

You can type print("Hello, World!") and press Enter. You should see the output "Hello, World!" displayed on the screen.

That's it! You've successfully installed Python on your Windows system and verified that it's working. You can now start using Python for programming and scripting.

Install Python 3 on Linux

To check if Python is already installed on your Linux system and, if not, to To check if Python is already installed on your Linux system and, if not, to install it, follow these steps::

Step 1: Check if Python is already installed

Open a terminal. You can usually do this by pressing Ctrl + Alt + T or by searching for "Terminal" in your desktop environment's applications menu.

In the terminal, type the following command and press Enter:

terminal

```
python3 --version
```

If Python is already installed, you'll see the installed version number (e.g., Python 3.8.3).
If Python is not installed, you'll likely see an error message.

Step 2: Install Python (if not already installed)

Python is often pre-installed on many Linux distributions. However, if it's not installed, you can install it using your system's package manager.

Ubuntu/Debian:

If Python is not already installed, open a terminal and run the following command to install Python 3:

terminal

```
sudo apt-get install python3
```

Fedora:

If Python is not already installed, open a terminal and run the following command to install Python 3:

terminal

```
sudo dnf install python3
```

CentOS/RHEL:

Python may not be pre-installed on some CentOS/RHEL systems. To install Python 3, open a terminal and run the following command:

terminal

```
sudo yum install python3
```

Step 3: Verify Python Installation

After installing Python, you can verify the installation by running the following command in the terminal:

terminal

```
python3 --version
```

You should see the version number you installed.

Step 4: Test Python

You can also run a simple Python command to test if it's working. In the terminal, type:

terminal

```
python3
```

You should see a Python shell prompt (usually >>>).

You can type print("Hello, World!") and press Enter. You should see the output "Hello, World!" displayed on the screen.

That's it! You've successfully installed Python on your Linux system and verified that it's working. If you face any issues, please refer to the following blog:

If you face any issue, please refer to the following blog:
https://solarianprogrammer.com/2017/06/30/building-python-ubuntu-wsl-debian/

You can now start using Python for programming and scripting.

Virtual environments

Virtual environments are a fundamental and indispensable aspect of modern programming, particularly in the context of Python development.

These environments are essentially isolated containers that house all the necessary software dependencies required for a specific project.

Their significance lies in addressing the challenges posed by managing multiple projects with varying software requirements on a single computer.

The primary reason for using virtual environments, as exemplified in Python programming, is to avoid conflicts between different project dependencies.

In the absence of virtual environments, Python and related packages are typically installed globally on the system.

This poses a significant issue when you need to work on multiple projects simultaneously, each having its own set of software requirements.

For instance, imagine Project A relies on Django version 2.1, while Project B, developed a year earlier, is still dependent on Django 1.10. Without virtual environments, managing such scenarios can become exceedingly complex, but with their implementation, these issues are effortlessly resolved.

One of the key advantages of virtual environments is their ability to keep project dependencies neatly isolated.

Each virtual environment acts as a self-contained ecosystem, allowing you to install and manage specific versions of libraries, frameworks, and packages without affecting the global system environment.

This isolation ensures that changes made to one project's dependencies do not inadvertently impact another project.

It provides a clean slate for each project, eliminating version conflicts and making it easier to maintain and update software components without conflicts.

While, there are several tools available for creating virtual environments in Python, one prominent and officially recommended option is Pipenv.

Introduced by renowned Python developer Kenneth Reitz in 2017, Pipenv has gained widespread popularity due to its simplicity and robustness.

Pipenv operates in a manner akin to npm and yarn in the Node.js ecosystem. It uses a Pipfile to define project dependencies and a Pipfile.lock to guarantee deterministic builds.

Determinism, in this context, means that every time you create a new virtual environment and install dependencies using Pipenv, you will obtain precisely the same configuration.

This predictability is crucial for ensuring that your project behaves consistently across different environments, and when collaborating with other developers.

Deterministic builds minimize the risk of unexpected issues arising from variations in dependency versions, thereby enhancing project stability.

In practice, creating a new virtual environment for each Python project using Pipenv has become the standard practice.

This approach ensures that projects remain self-contained, simplifying development, testing, and deployment processes.

With the help of Pipenv, Python developers can effortlessly manage project-specific dependencies, facilitating smoother project development and collaboration.

To get started with Pipenv, you can use the package manager pip, which is often installed alongside Python 3. The following command demonstrates how to install Pipenv:

command prompt

```
> pip install pipenv
```

In conclusion, virtual environments are an essential tool in modern software development, offering isolation and determinism that simplifies project management and ensures reliable and consistent outcomes.

Python's Pipenv, in particular, has become a popular choice for managing virtual environments, making it easier than ever to work on multiple projects with distinct dependencies while maintaining a clean and predictable development environment.

Install Django

In this section, we will walk you through a series of commands used to set up a new Django project from scratch. These commands will help you create a new project directory, install Django, and start a development server.

Change Directory to Desktop:

command prompt

```
> cd Desktop
```

This command changes your current working directory to the Desktop. All the subsequent commands will be executed in this location.

Create a Project Directory:

The next step is to create a directory where you will store all your Django projects. In this example, we will create a directory called django_apps on your desktop.

You can replace the path with the location where you want to store your projects. Open your command prompt (or terminal on macOS/Linux) and run the following commands:

command prompt

```
> mkdir django_projects
> cd django_projects
```

Here, we use the mkdir command to create a new directory called django_apps, and then we use cd to change our current directory to django_ apps.

command prompt

```
>mkdir myfirstproject
```

Here, we're creating a new directory called myfirstproject within your Desktop directory. This is where your Django project will reside.

Change Directory to your Project Directory:

command prompt

```
>cd myfirstproject
```

You navigate into the myfirstproject directory that you just created. This is where you'll be working on your Django project.

Activate a Virtual Environment:

command prompt

```
>pipenv shell
```

This command activates a virtual environment using Pipenv.

A virtual environment is an isolated environment for your project, ensuring that your project's dependencies don't interfere with other Python projects on your system.

You'll notice that your command prompts changes to include the virtual environment name, in this case, (myfirstproject).

Install Django:

command prompt

```
> pip install django
```

With the virtual environment activated, you install Django using the pip package manager. This step ensures that your project has Django available as a dependency.

Start a New Django Project:

command prompt

```
> django-admin startproject myfirstproject .
```

Here, you use the django-admin command to create a new Django project named myfirstproject. The . at the end specifies that the project should be created in the current directory.

This command sets up the initial structure for your Django project.

Run the Development Server:

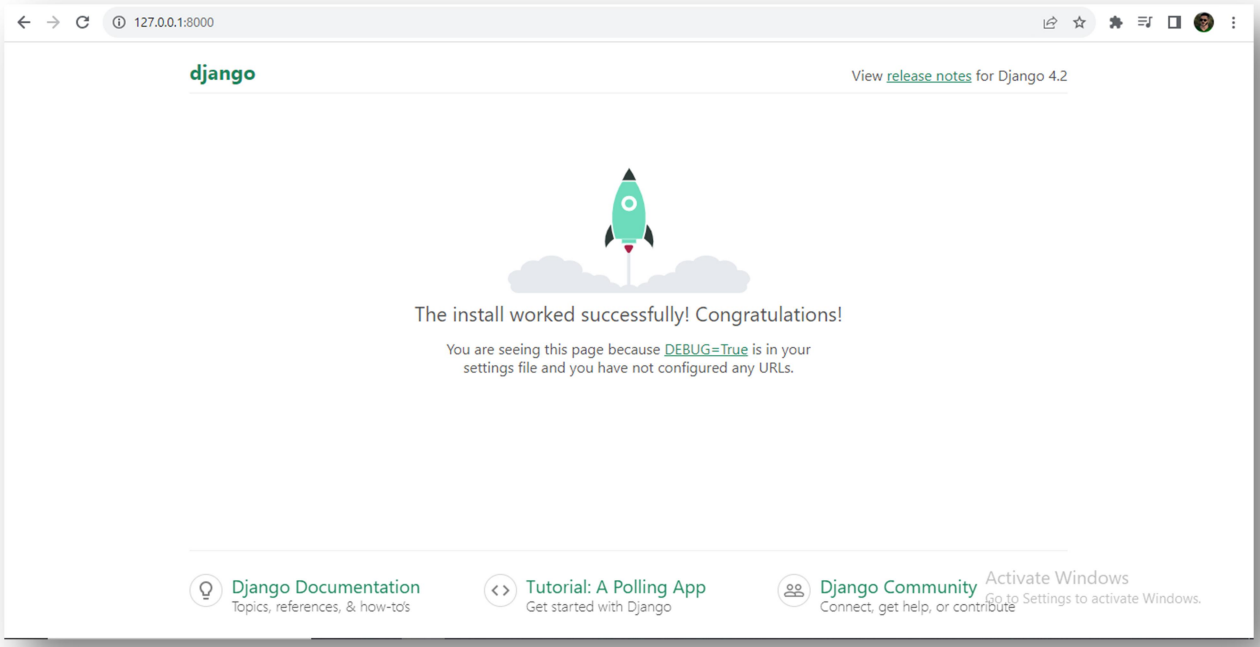
command prompt

```
> python manage.py runserver
```

Finally, you use the manage.py script provided by Django to start the development server. This server allows you to preview your Django project locally.

The server will be accessible http://127.0.0.1:8000/.

You can open a web browser and visit the provided URL to see the default Django welcome page, indicating that your project is up and running.



These commands are the initial steps to create a Django project and start developing your web application.

As you continue your journey with Django, you'll use additional commands to create apps, define models, and build the functionality of your web application.

Understanding the Django Project Structure

In this section, we'll dive deep into the Django project structure, unveiling the various files and directories that make up a Django project.

Understanding the project structure is crucial for any Django developer, as it forms the foundation of your web application.

Let's embark on this journey to explore the inner workings of a Django project.

Overview

A Django project is organized into a well-defined directory structure that follows the "Django way" of development.

This structure encourages modularity, clean code, and separation of concerns.

Before we delve into the details, let's take a look at the high-level overview of a typical Django project directory:

```
myproject/
    ├── myproject/
    │   ├── __init__.py
    │   ├── settings.py
    │   ├── urls.py
    │   └── wsgi.py
    ├── manage.py
    ├── app1/
    ├── app2/
    └── ...
```

Let's break down each of these components step by step.

The Project Directory

myproject/
The outermost directory, often named after your project, contains all the project-related files and directories.

The project name is user-defined and may vary from one project to another. It's good practice to choose a name that reflects the purpose of your project.

myproject/__init__.py
This empty file is required to tell Python that this directory should be treated as a Python package. It's often empty but can contain package-level initialization code if necessary.

myproject/settings.py
The heart of your Django project, this file holds all the project settings. You'll configure your database, installed apps, middleware, authentication settings, and much more in this file.

myproject/urls.py
This file defines the URL patterns for your project and serves as a roadmap for how incoming requests are routed to the appropriate view functions within your apps.

myproject/wsgi.py
This file is used for deploying your Django application on a production server. It serves as an entry point for the WSGI (Web Server Gateway Interface) server to communicate with your Django app.

The manage.py Script

The manage.py script is a powerful tool that simplifies various tasks related to your Django project.

You can use it to create database tables, run development servers, create superusers, and much more. The manage.py script is a Python script that interacts with your project's settings.

The Apps

Django encourages a modular approach to development. Each logical component of your project is encapsulated within a Django app.

These apps can be reused across projects, fostering code reusability.

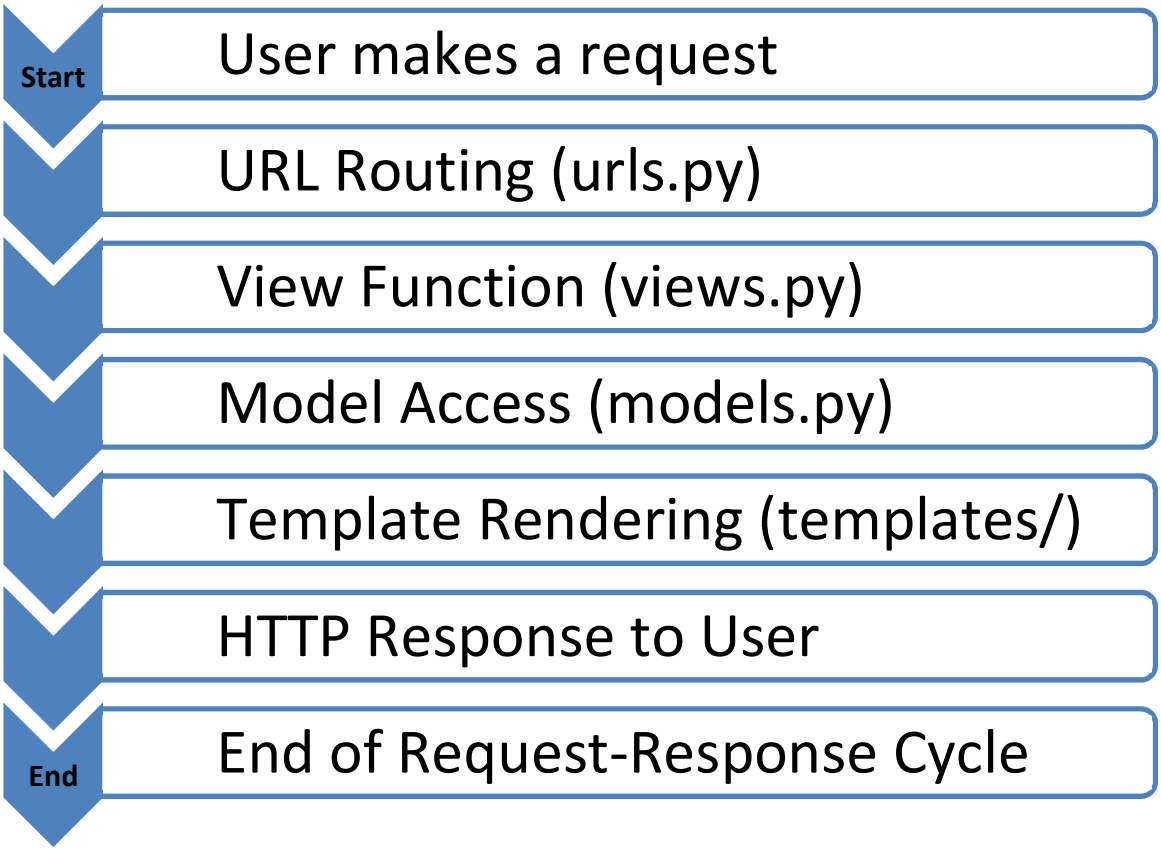
Examples of apps include user authentication, blog functionality, or an e-commerce system. You can create your apps using the Python manage.py startapp app_name command.

Understanding the Django project structure is the first step in becoming a proficient Django developer.

In the next section, we'll delve into configuring your project settings and kickstart your Django journey!

FlowChart

Designing a comprehensive flowchart for the entire Django framework can pose a challenge due to its modular and adaptable characteristics. Presented here is a simplified flowchart that delineates the fundamental request-response cycle within a Django application. It is essential to note that this representation provides a high-level overview, while the actual flow includes additional intricacies and interactions.



User Initiates Request: The process commences when a user triggers an HTTP request within the Django application, typically through actions like clicking a link, submitting a form, or engaging in other interactive elements.

URL Routing (urls.py): Django utilizes the defined URL patterns in the urls.py file to ascertain which view function should handle the incoming request.

View Processing (views.py): The view function takes charge of processing the request, engaging with models if necessary, and determining the data to be presented. Subsequently, it produces an HTTP response.

Model Interaction (models.py): If database interaction is required, the view communicates with model classes specified in models.py. This involves tasks such as creating, retrieving, updating, or deleting data in the database.

Template Rendering (templates/): To generate dynamic content, the view employs template rendering. Templates, comprised of HTML files with embedded Django template language syntax, enable the incorporation of dynamic data.

User Receives HTTP Response: The rendered template becomes part of an HTTP response, which is dispatched back to the user's browser. The response may contain HTML, JSON, or other content types based on the nature of the request.

Conclusion of Request-Response Cycle: The cycle concludes, and the user observes the outcome of their request in their browser.

It's important to acknowledge that this depiction is a simplified overview, and real-world scenarios may involve additional components, middleware, and configurations in the overall process.

Here is an example:

Django, a Python web framework known for fostering rapid development and adhering to a clean, pragmatic design philosophy, follows the Model-View-Template (MVT) architectural pattern, a variation of the traditional Model-View-Controller (MVC) pattern. Here's a simplified breakdown of Django's functioning:

Models:

Within Django, a model takes the form of a Python class representing a database table, with each class attribute corresponding to a field in the respective table.

Models serve to define the structure of data and provide an abstraction layer for interacting with the database.

Django employs an Object-Relational Mapping (ORM) system, enabling interaction with the database through Python code rather than direct SQL queries.

Views:

Views in Django play a crucial role in handling user requests and delivering corresponding responses. They take input from users in the form of HTTP requests, interact with required data (often through models), and subsequently generate responses, which may manifest as HTML pages, JSON, or other formats. Views encapsulate the application's business logic and manage the flow of data within the framework.

Templates:

Templates in Django manage the presentation logic by dynamically generating HTML based on data supplied by views. Utilizing a distinct syntax, templates enable the embedding of Python-like code directly within HTML for seamless integration of dynamic content.

URLs and Routing:

Within Django, a URL routing mechanism is employed to link URLs to specific views.

In the urls.py file, you establish patterns that align with particular URLs and link them to their corresponding views.

Settings and Configuration:

Django employs a settings module to store configuration parameters for your project, encompassing aspects like database settings, middleware, and more.

These configurations are housed in the settings.py file within your project.

Middleware:

Middleware components serve to globally process requests and responses, either before they reach the view or after they leave it.

Instances of middleware include authentication middleware, security middleware, and others.

Django furnishes a toolkit and established conventions that simplify the construction of web applications, fostering code reusability, modularity, and maintainability. Adhering to these conventions and patterns allows developers to concentrate on feature development, rather than grappling with low-level intricacies.

Install Git

In the world of web development, managing your project's codebase is essential.

You need a system that allows you to keep track of changes, collaborate with other developers seamlessly, and ensure that your Django web applications are always in a reliable state.

This is where Git comes into play.

What is Git?

Git is a distributed version control system that helps you track changes in your codebase. It was created by Linus Torvalds in 2005 and has since become an indispensable tool for software development. Git is particularly valuable in the Django ecosystem for several reasons:

- **Version Control:** Git allows you to track every change made to your project, making it easy to roll back to previous states or identify when and why a particular change was made.

- **Collaboration:** When working on a Django project with a team, Git enables smooth collaboration. Multiple developers can work on the same codebase simultaneously without interfering with each other's work.

- **Branching and Merging:** Git's branching and merging capabilities are powerful tools for managing different features or bug fixes in isolation. You can develop new features in separate branches and then merge them into the main codebase when they are ready.

- **Backup and Recovery**: Git provides a secure and efficient way to back up your project. If something goes wrong, you can always revert to a known working state.

- **Open Source:** Git is open source and widely adopted, which means a vast community of developers continually improves and supports it.

Installing Git

Before you can start using Git with your Django project, you'll need to install it on your development machine. Here's how to do it:

On Windows:

Visit the official Git website at https://git-scm.com/.

Download the Windows installer and run it.

Follow the installation wizard's instructions, accepting the default settings unless you have a specific reason to change them.

After installation, open the Git Bash terminal that comes with Git, or use your preferred terminal if you're more comfortable with that.

On macOS:
If you have Homebrew installed, you can install Git by running the following command in your terminal:

mac command prompt

```
brew install git
```

If you don't use Homebrew, you can download the macOS Git installer from https://git-scm.com/download/mac and follow the installation instructions.

Once installed, open your terminal, and Git should be available for use.

On Linux (Ubuntu/Debian):
Open a terminal.

Run the following command to install Git:

Terminal

```
sudo apt-get update
sudo apt-get install git
```

After installation, Git will be ready to use in your terminal.

Verifying the Installation
To verify that Git has been installed successfully, open your terminal and run the following command:

Terminal

```
$ git --version
```

If Git is installed correctly, you should see the installed version displayed in the terminal.

Congratulations! You now have Git installed on your system and are ready to start using it to manage your Django projects more effectively. In the following chapters, we'll explore how to initialize a Git repository, make commits, create branches, and collaborate with others using Git.

Text Editors

Text Editors and Setting Up Visual Studio Code for Django.
A text editor is a software application that allows you to create, edit, and manage the code that makes up your Django projects. Choosing the right text editor is crucial as it significantly impacts productivity and coding experience. In this chapter, we'll explore what text editors are, recommend a few popular ones, and guide you through the process of downloading and installing Visual Studio Code (VS Code), a widely used and versatile text editor for Django development.

What Are Text Editors?
Text editors are software tools designed specifically for working with plain text files, including code files written in programming languages like Python, HTML, CSS, and JavaScript. They provide an environment for developers to write, edit, and organize code efficiently. Unlike full-fledged integrated development environments (IDEs), text editors are lightweight and flexible, making them an excellent choice for web development with Django.

Why Do You Need a Text Editor?
Here are some reasons why you need a text editor for Django development:

- **Code Writing:** Text editors are optimized for writing code, offering features like syntax highlighting, auto-indentation, and code completion that make coding faster and more accurate.

- **Customization:** Text editors are highly customizable, allowing you to install extensions and configure settings to suit your development workflow and coding style.

- **Lightweight:** Text editors are generally less resource-intensive than IDEs, making them more responsive, especially on older computers.

- **Versatility:** Text editors are not limited to Django development; you can use them for various programming tasks and languages.

Now, let's dive into one of the most popular text editors for Django development: Visual Studio Code (VS Code).

Visual Studio Code (VS Code)
Visual Studio Code, often referred to as VS Code, is a free and open-source text editor developed by Microsoft. It has gained immense popularity among developers due to its extensibility, powerful features, and a large community of users and extensions.

Downloading and Installing VS Code
Here's how you can download and install VS Code:

Visit the Official Website: Open your web browser and go to the official Visual Studio Code website at https://code.visualstudio.com/.

Download for Your Platform: VS Code is available for Windows, macOS, and Linux. Click on the download link corresponding to your operating system.

Install VS Code:

Windows: Once the installer is downloaded, run it and follow the installation wizard's instructions.

macOS: Locate the downloaded file in your Downloads folder, double-click it, and follow the installation instructions.

Linux: Depending on your distribution, you may need to follow specific installation instructions.

Refer to the VS Code documentation for details.

Launch VS Code: After installation, you can launch VS Code from your system's application launcher or by running the code command in your terminal (Linux).

Basic Configuration: Upon first launch, you can customize VS Code to your liking by installing extensions and adjusting settings. VS Code will guide you through this process.

Recommended VS Code Extensions for Django Development

To enhance your Django development experience with VS Code, consider installing the following extensions:

- **Python:** Provides Python language support, including syntax highlighting, debugging, and code completion.
- **Django:** Offers Django-specific features like template language support and code snippets.
- **Pylance:** Enhances Python support in VS Code, providing advanced features like type checking and code analysis.
- **GitLens:** Improves Git integration within VS Code, helping you manage version control effectively.

With your VS Code installation and extensions set up, you're well-prepared to embark on your Django development journey.
In the upcoming chapters, we'll delve deeper into using VS Code for Django projects and explore various aspects of web development with this powerful framework.

Occasionally, you may observe incorrect Python code indentation due to width constraints on this page. To rectify this issue, kindly copy the code and paste it into a text editor such as VS Code. Doing so should result in the correct indentation.

Chapter 3: Views and Templates

Creating a New Django Project

In this chapter, we will go through the process of setting up our development environment and creating our first Django project.
By the end of this chapter, we will have a basic understanding of how to create a Django project and run it locally.

Step 1: Creating a Project Directory

The first step is to establish a directory where we can store all our Django projects. In this example, we will create a directory named "django_projects" on our desktop.

You can replace the path with your preferred location for storing your projects.
Now, open the command prompt (or terminal on macOS/Linux) and execute the following commands:

command prompt

```
> cd Desktop
> mkdir django_projects
> cd django_projects
```

Here, we use the mkdir command to create a new directory called "django_projects," and then we use cd to change our current directory to "django_projects."

Step 2: Creating a Django Application Directory

Within the "django_projects" directory, we need to establish a directory for our specific Django project. Let's name it "newspaper_app."

Execute the following commands:

command prompt

```
> mkdir newspaper_app
> cd newspaper_app
```

Once again, we use the mkdir command to create a new directory called "newspaper_app," and then we use cd to change our current directory to "newspaper_app."

Step 3: Setting Up a Virtual Environment

Now, it's time to set up a virtual environment for our Django project.
A virtual environment is an isolated environment that allows us to install packages and dependencies separately for each project, ensuring that project-specific dependencies don't interfere with each other. We will use pipenv, a popular tool for managing virtual environments and dependencies.

Run the following command:

command prompt

```
> pipenv shell
```

This command creates and activates a virtual environment named 'newspaper' for our project. You can replace 'newspaper' with your preferred name. Once activated, any packages installed will be isolated within this environment.

Step 4: Installing Django

Now that we have our virtual environment set up, we can proceed to install Django.
Run the following command:

command prompt

```
> pip install django
```

This command installs Django within our virtual environment, making it available for our project.

Step 5: Creating a Django Project

With Django installed, we can now create a new Django project.
Run the following command:

command prompt

```
> django-admin startproject newspaper_project .
```

Here, we use django-admin to initiate a new Django project named "newspaper_project." The "." at the end specifies that the project should be created in the current directory. This command generates the basic structure and files needed for a Django project.

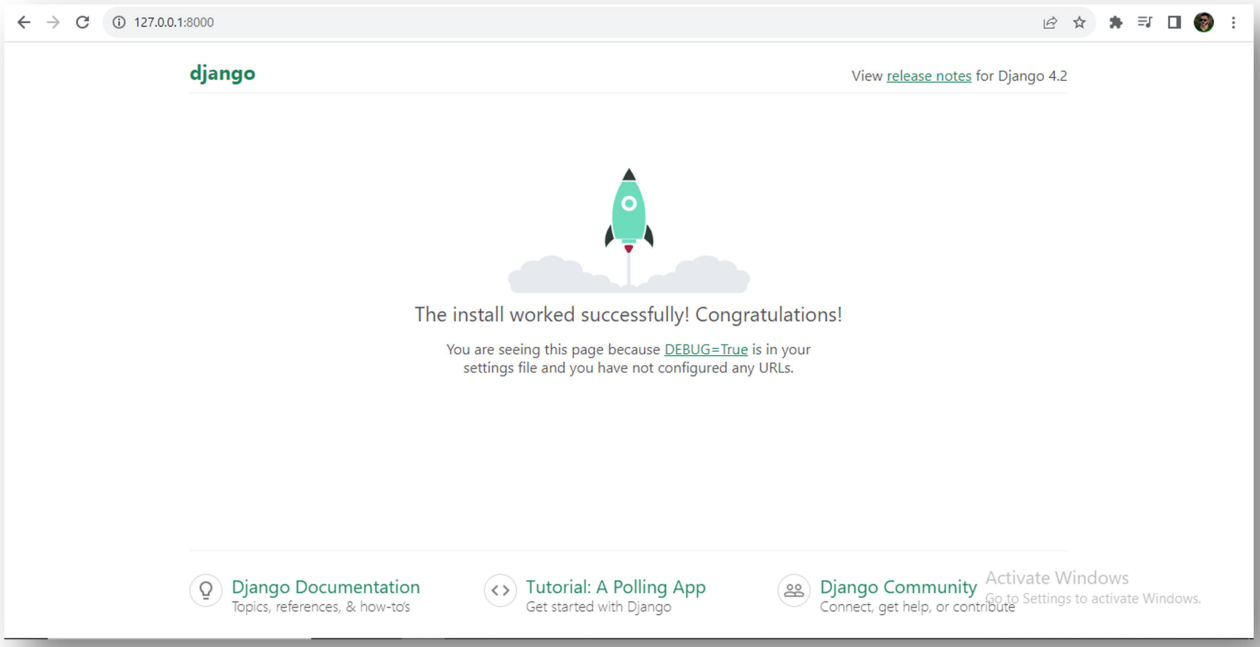
Step 6: Running the Development Server

Finally, let's start the development server to see our Django project in action.
If the server is already running, stop it using Control+C. Then, enter the command "python manage.py runserver" in the command line:

command prompt

```
> python manage.py runserver
```

This command launches the Django development server, and we should see output indicating that the server is running.
We can now open a web browser and visit http://127.0.0.1:8000/ to view our new Django project's welcome page.



Congratulations! we've successfully set up our Django development environment, created a new project, and started the development server.

Creating a New App - The News App

Creating an app in a Django project is a fundamental step in structuring and organizing our web application.

Django, a high-level Python web framework, is renowned for its adherence to the "Don't Repeat Yourself" (DRY) principle and employs the "Model-View-Controller" (MVC) architectural pattern, known as "Model-View-Template" (MVT) in Django.

Benefits of Creating Apps:

- **Modularity and Reusability:** Django apps are modular components that encapsulate specific functionality within our web application. This modular approach simplifies code organization and encourages reusability across different projects.
- **Separation of Concerns:** Apps enable the separation of different aspects of our application, such as user authentication, blog posts, or e-commerce features. This separation enhances code clarity and organization.

- **Code Organization:** Django enforces a structured directory layout within each app, aiding developers in locating and working with different parts of the application. This structure includes directories for models, views, templates, static files, and more.
- **Collaboration:** When collaborating in a team, dividing our project into apps promotes teamwork by allowing team members to focus on different app components, minimizing conflicts in a shared codebase.
- **Scalability:** Django apps are designed for scalability. As our project grows, we can effortlessly introduce new apps to manage additional features without disrupting the existing codebase.
- **Reusable Apps:** Django's ecosystem offers a wide array of third-party apps and packages that can be seamlessly integrated to add common functionalities, saving valuable development time.
- **Testing and Maintenance:** Apps can be individually tested, simplifying issue identification and resolution. This modular approach also streamlines maintenance efforts, as specific apps can be updated or replaced without affecting the entire project.
- **Encapsulation and Isolation:** Apps provide a level of encapsulation and isolation, allowing for the independent management of dependencies, templates, and static files for each app.

Exiting the Running Server:
To exit the currently running server, follow these steps:
Open the command window.
Press **Control + C**.

Creating a new Django app named 'news':

To create a new Django app named 'news,' execute the following command:

command prompt

```
> python manage.py startapp news.
```

In Django, an app is a modular component that encapsulates a specific functionality within the application. In this case, we are creating the 'news' app to manage news-related content in our newspaper application.

Configuring installed apps

To configure our Django project's installed apps, we need to work with the INSTALLED_APPS list located in the settings.py file. This list determines which applications are active within our project.

Occasionally, you may observe incorrect Python code indentation due to width constraints on this page. To rectify this issue, kindly copy the code and paste it into a text editor such as VS Code. Doing so should result in the correct indentation.

settings.py

```
INSTALLED_APPS = [
```

```
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'news',  # Add the 'news' app here.
]
```

This list includes core Django functionality with prefixes like django.contrib, offering features such as an admin interface, authentication, content types, sessions, and message handling. We can include or exclude these apps based on our project's needs.

Additionally, we can include custom apps like 'news' in the INSTALLED_APPS list. This indicates that we've either created a 'news' app ourselves or included one created by someone else, and we want to integrate it into our project.

Adding an app to INSTALLED_APPS, allows Django to recognize and use it within our project. This step is crucial for organizing our project's functionality into manageable and scalable components, known as apps.

The 'news' app likely contains models, views, and templates designed for handling news articles or related content. Including it in INSTALLED_APPS ensures Django can recognize and utilize these components.

Before proceeding, let's open Visual Studio Code (VS Code) and select the appropriate interpreter. It is crucial to choose the correct interpreter as, since we have already downloaded Django and other dependencies in the virtual environment. Therefore, we need to instruct VS Code on where to locate and interpret the code.

So we have to point the interpreter to our virtual environment.

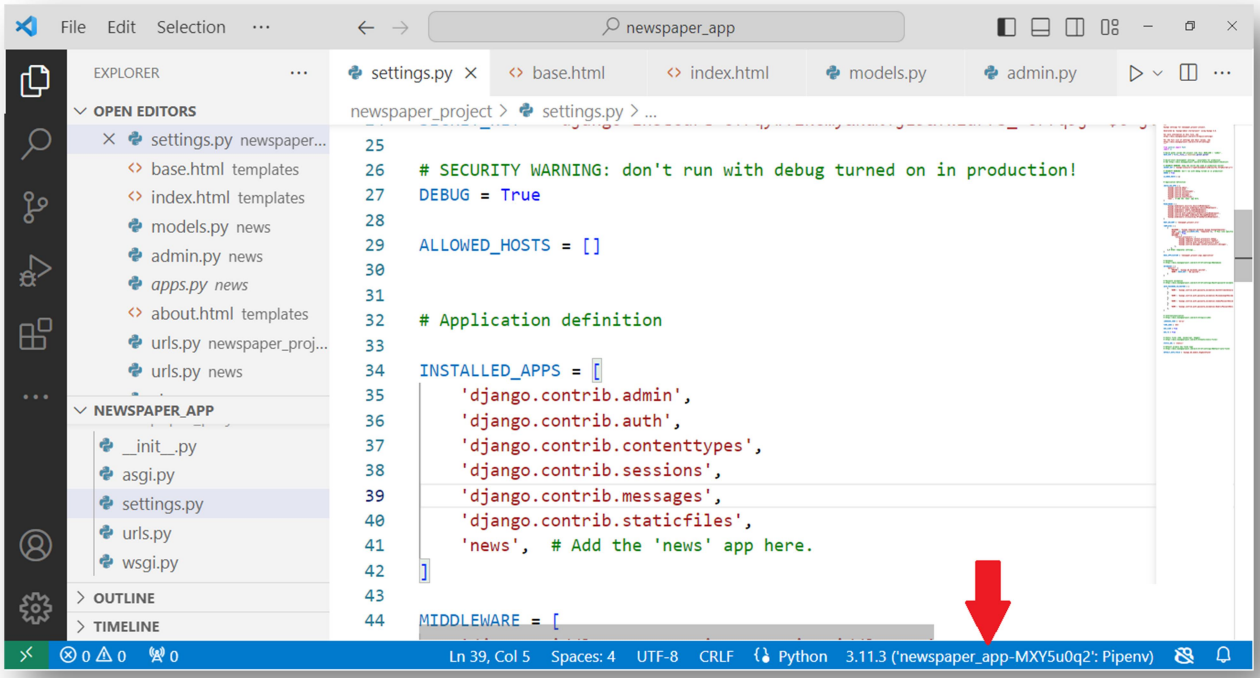
Select Python Interpreter:

- Open the Command Palette by pressing Ctrl+Shift+P.
- Type and select "Python: Select Interpreter."
- A list of detected Python interpreters will be displayed. Choose the one we want to use for our project.

Manually Specify Interpreter Path (Optional):

- If the interpreter we want to use is not automatically detected, we can manually specify the interpreter path.
- Open the Command Palette (Ctrl+Shift+P) and type "Python: Select Interpreter."

- Choose "Enter interpreter path", and provide the path to our Python interpreter executable.



After completing these steps, if the server is already running, stop it using Control+C. Then, enter the command "python manage.py runserver" in the command line:

command prompt

```
> python manage.py runserver
```

This step reloads the server with the news app and URL configurations, enabling us to define views and templates for the 'news' app, and access it through the development server.

Views

Creating Our First View

To create our first view, follow these steps:

Go to the 'news/views.py' file.

Paste the following code:

news/*views.py*

```
# We're creating a view function, which is like a handler for web requests.
# First, we import some tools from Django that we need for our view.

from django.http import HttpResponse  # A class for creating HTTP responses

# Now, let's define our view function. It's called newsView, and it takes a
'request' as its parameter.
```

```
def newsView(request):

    # Inside the function, we're keeping things simple. We're just returning an
HTTP response.

    # In this case, it's a response saying 'Hello, World!'

    return HttpResponse('Hello, World!')
```

The HttpResponse class is what we're using to send a response back to whoever made a request to our web application.Our newsView function is quite straightforward. When someone accesses a specific URL in our web app, this function gets called.

It doesn't do anything fancy; it just responds with 'Hello, World!'. Views are fundamental to web applications, as they handle user requests, execute logic, interact with the database, and generate responses. In more complex applications, this function might interact with a database, render dynamic web pages, or do a lot of other cool things.

But for now, it's keeping it simple and just saying hi!

Urls

Setting Up URL Routing in Django for our Newspaper App

In this section, we'll delve into Django's URL routing mechanism, which enables us to connect URLs to view functions.

Creating an 'urls.py' File for the 'news' App:

To create an empty 'urls.py' file within the 'news' app directory, use the following command:

command prompt

```
>echo >news/urls.py
```

The 'urls.py' file for the 'news' app defines URL patterns and routing. URL mappings link views to specific URLs, which dictate the displayed content.

In Django, URL routing directs incoming requests to view functions, enabling the creation of dynamic and interactive web applications.

news/urls.py

```
# We start by importing the necessary module for handling URL patterns in Django.
```

```python
from django.urls import path

# Next, we import the 'newsView' function from the 'views' module of the current Django app (denoted by the dot).

from .views import newsView

# Now, we define a variable called 'urlpatterns'. This variable will hold a list of URL patterns for our Django app.

urlpatterns = [

    # Inside the list, we use the 'path' function to define a URL pattern.
    # The first argument ('') represents the URL path. An empty string ('') means this pattern matches the root URL.
    # The second argument ('newsView') is the view function that will be called when this URL pattern is matched.
    # The third argument ('home') is a unique name given to this URL pattern, which can be used to reference it later in the code.

    path('', newsView, name='home')

]
```

In simpler terms:

Importing Modules:
We're importing the necessary tools from Django to handle URL routing (path) and importing our newsView function.

URL Patterns List:
We create a list called urlpatterns to store our URL patterns.

Defining a URL Pattern:
We use the path function to define a URL pattern.

- The first argument '' means this pattern matches the root URL (e.g., http://example.com/).
- The second argument newsView is the function that will be executed when this URL is visited.
- The third argument name='home' is a name we give to this pattern, which helps us reference it later in the code.

This code essentially says, "Hey Django, when someone visits the root URL of our app, call the newsView function, and we'll call this pattern 'home'."

urls.py in the newspaper_app

newspaper_*projects/urls.py*

```
# This line imports the admin module from the django.contrib package.
from django.contrib import admin

# This line imports the path and include functions from the django.urls package.
from django.urls import path, include

# This is a list called urlpatterns, which is used to define the URL patterns for
our Django project.
urlpatterns = [
    # This line maps the URL 'admin/' to the Django admin site. When we go to
'/admin/', we access the admin interface.
    path('admin/', admin.site.urls),

    # This line maps the root URL ('') to the 'news.urls' module.
    # It means that any request to the root URL of our project will be handled by
the URLs defined in the 'news.urls' module.
    # This is a way to organize and modularize our URL patterns.
    path('', include('news.urls')),
]
```

So, in summary, this code sets up the URL patterns for a Django project. It includes the default admin interface at '/admin/' and delegates any other URL patterns to be handled by the URLs defined in the 'news.urls' module. This modular approach makes it easier to manage and organize our project's URL structure.

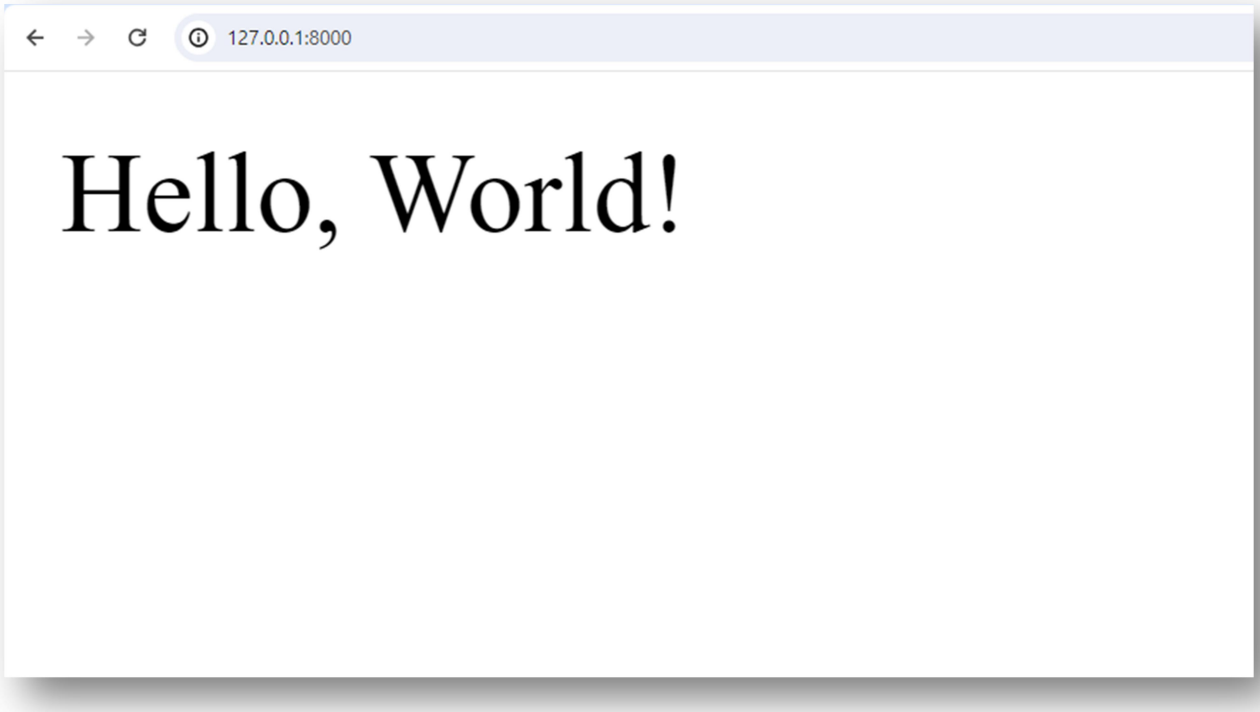
URL routing in Django is essential for creating well-organized, maintainable, and scalable web applications. It segregates the logic of handling different URLs into distinct views, enhancing code modularity. By associating URLs with views, we can ensure that each page of our website is managed by the appropriate function, facilitating the creation of complex web applications.

Furthermore, it permits the reuse of apps across different projects, as their URL patterns can be included wherever needed.After completing these steps, if the server is already running, stop it using Control+C. Then, enter the command "python manage.py runserver" in the command line:

command prompt

```
> python manage.py runserver
```

This command starts the Django development server.



Templates

Creating a 'template' Directory:

To begin, let's create a 'template' directory in the project's root directory. This directory will serve as the storage space for HTML templates, which are essential for rendering web pages in our Django application.

To create this directory, use the following command:

command prompt

```
>mkdir template
```

Creating an 'index.html' File:

Now, let's create an empty 'index.html' file within the 'template' directory.HTML templates define the structure and layout of web pages in a Django application. This file will be populated with HTML code later.

Execute the following command to create the 'index.html' file:

command prompt

```
>echo> template/index.html
```

Configuring Templates:

Moving forward, let's delve into the TEMPLATES setting in Django.This setting plays a crucial role in determining how Django handles the rendering of HTML templates for our views. It's essential for building the user interface of our web application.

Now, let's define the path to our templates file in settings.py.

settings.py

```
from pathlib import Path
import os
```

import os: Import the Python os module at the beginning of your settings.py file. The os module is essential for handling operating system-related tasks. Placing the import statement at the top of the file ensures that all subsequent code can make use of the os module.

Now, let's update the TEMPLATES section in settings.py.
Here's the relevant portion of the code:

settings.py

```
TEMPLATES = [
    {

        # Other templates settings...
```

```
        'DIRS': [os.path.join(BASE_DIR, 'templates')],   # This line specifies
the template directory path.

        # Other templates settings...
  },
]
```

In this code snippet, we are primarily focusing on the 'DIRS' key within the template settings dictionary. This key points to a list of directories where Django should search for template files.Storing templates in a dedicated 'templates' directory helps maintain project organization and aligns with best practices for Django development.

Properly configuring INSTALLED_APPS and templates is a foundational step in setting up a Django project, enabling the development of robust and maintainable web applications.

Creating a Basic View:
In Django, views are responsible for handling incoming requests, processing data if necessary, and returning an appropriate response.

Let's begin by creating a simple view that displays the homepage of our website.To create views in Django, define Python classes. In our views.py file, we should have a structure similar to the following:

news/views.py

```
# We're importing a class named TemplateView from the django.views.generic
module.
from django.views.generic import TemplateView

# We're defining a new class called HomePageView, and it's inheriting from
TemplateView.
class HomePageView(TemplateView):
    # Here, we're setting a property of the class called template_name.
    # This property tells Django which HTML template to use when rendering this
view.
    # In this case, it's set to 'index.html', so Django will look for a file
named 'index.html'.
    template_name = 'index.html'
```

In simpler terms:

- We import a special class called TemplateView from Django that helps in rendering HTML templates.