2CSDE86 Application Development Frameworks (ADF)

Lecture-1

7th CSE

Daiwat Vyas & Ajaykumar Patel

Disclaimer

- The content in the ppt is taken from various online sources and reference books and after referring it was considered for including it in the ppt slides.

Course Policy

- ADF Course Policy Odd 22-23

Introduction to Course- ADF

- 3 credit course : [2 0 2 3] ([L T P C])

- **Importance of Course:**

- Django is a Python based web framework that enables rapid development of secure and maintainable websites.
- Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel.
- It allows developers to use modules for faster development. As a developer, you can make use of these modules to create apps, websites from an existing source. It speeds up the development process greatly, as you do not have to code everything from scratch.

Introduction to Course- ADF (contd…)

- **Importance of Course:**

- Basically Python is a language that can be used for developing anything and everything you want.

- Django is new web development framework in python which support MVT (Model-View-Template) architecture.

- Now a days, most of the organization wants to develop data science and analytics to enhance decision making.

- Huge community support and large amount of API is available.

Introduction to Course- ADF (contd…)

- **Pre-requisite:**

- Awareness of basics of python

- Eagerness and readiness to do programming

Web Development Framework

- What is a framework?

- What is a web development framework?

Web Development Framework (contd…)

- What is a framework?

- What is a web development framework?
- A framework, or software framework, is a platform for developing software applications. For example, a framework may include predefined classes and functions that can be used to process input, manage hardware devices, and interact with system software.

- A web development framework is a set of resources and tools for software developers to build and manage web applications, web services and websites.

- Developers can use the framework to define the 'out-of-the-box' content management capabilities, user authentication features, and administrative tools.

Web Development Framework (contd…)

- Why framework needed in web development?

Web Development Framework (contd…)

- Why framework needed in web development?

- The purpose of framework is to allow designers and developers to focus on building an unique feature for their web based projects rather than re-inventing by coding.

- Framework is specially created to help you boost the performance and efficiency of your web app development task.

- Easy de-bugging, improved code efficiency, Easy code reusability, etc.

Web Development Framework (contd…)

- How have Web Development Frameworks Eased the App Development Process?

- **Higher Performance**

- Web frameworks are specifically designed with pre-built features and functionalities to deliver extremely high efficiency and performance while building apps.

- The loading capacity of the websites designed with appropriate framework enhances significantly.

Web Development Framework (contd…)

- How have Web Development Frameworks Eased the App Development Process?

- **Reduction in Errors**

- Most of the framework methodologies incorporate the finest software engineering practices.

- Numerous web frameworks have a pre-built or externally integrated testing mechanism that tests the code there and then, reducing the number of errors in the final code.

Web Development Framework (contd…)

- How have Web Development Frameworks Eased the App Development Process?

- **Faster Development**

- Most web frameworks come with pre-written templates and objects that could be leveraged to complete redundant programming tasks.

- These tools save the time of developers and let them focus on the core programming part producing swift and more productive outcomes.

Web Development Framework (contd…)

- How have Web Development Frameworks Eased the App Development Process?

- **Better Reliability and Security**

- Frameworks include hundreds of ready-made components created and regularly updated by a community of developers.

- This enormous backing by the programming community ensures your project won’t get stuck in between, and the best possible solution for the business challenges can be developed.

Web Development Frameworks (contd…)

- Ruby on Rails
- Django
- Angular
- Laravel
- Express
- Etc…

Web Development Frameworks (contd…)

- **Post Lecture task**

- **Comparative study of Web Development Frameworks available in market.**

Web Development Framework- Django

- **What is Django?**

- Django is an open-source framework for backend web applications based on Python — one of the top web development languages.

- Its main goals are providing simplicity, flexibility, reliability, and scalability to developers.

- Django is the framework for web applications, as it allows developers to use modules for faster development. As a developer, you can make use of these modules to create apps, websites from an existing source. It speeds up the development process greatly, as you do not have to code everything from scratch.

Web Development Framework- Django

- **What is Django?**

- Django is a prominent Python framework that has an extremely high demand among developers and businesses.

- Django uses the concept of code reusability and follows the MVT (Model-View-Template) Architecture that enables it to develop applications at a faster pace.

- Django is one of the finest web frameworks that can be leveraged to create both the frontend and backend of the application.

Web Development Framework- Django (contd…)

- **Why Django?**

- Some of the key features of Django are messaging, in-built validation, and authentication.

- This framework is suitable to design complex and large business projects.
- Faster development.
- Availability of inbuilt authentication tool.
- Huge community support.
- Well-organized documentation.
- It can create secure and robust applications.

Web Development Framework- Django (contd…)

- **Why Django?**

- **It’s fast and simple**

- One of Django’s main goals is to simplify work for developers. To do that, the Django framework uses:

  - The principles of rapid development, which means developers can do more than one iteration at a time without starting the whole schedule from scratch;
  - DRY philosophy — Don’t Repeat Yourself — which means developers can reuse existing code and focus on the unique one.

- As a result, it takes a lot less time to get the project to market.

Web Development Framework- Django (contd…)

- **Why Django?**

- **It’s secure**
- Security is also a high priority for Django. It has one of the best out-of-the-box security systems out there, and it helps developers avoid common security issues, including:

  - clickjacking,
  - cross-site scripting
  - SQL injection.

- Django promptly releases new security patches. It’s usually the first one to respond to vulnerabilities and alert other frameworks.

Web Development Framework- Django (contd…)

- **Why Django?**

- **Clickjacking:** an attack that fools users into thinking they are clicking on one thing when they are actually clicking on another. Its other name, user interface (UI) redressing, better describes what is going on.

- Eg. The user visits the page and clicks the “Book My Free Trip” button. In reality the user is clicking on the invisible iframe, and has clicked the “Confirm Transfer” button. Funds are transferred to the attacker’s account.

Web Development Framework- Django (contd…)

- **Why Django?**

- **Cross-Site Scripting:** Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites.

- Eg.: Examples of reflected cross-site scripting attacks include when an attacker stores malicious script in the data sent from a website's search or contact form. A typical example of reflected cross-site scripting is a search form, where visitors sends their search query to the server, and only they see the result.

Web Development Framework- Django (contd…)

- **Why Django?**

- **It suits any web application project**
- With Django, you can tackle projects of any size and capacity, whether it's a simple website or a high-load web application. Why use Django for your project? Because:

  - It's fully loaded with extras and scalable, so you can make applications that handle heavy traffic and large volumes of information;
  - It is cross-platform, meaning that your project can be based on Mac, Linux or PC;
  - It works with most major databases and allows using a database that is more suitable in a particular project, or even multiple databases at the same time.

Web Development Framework- Django (contd…)

- **Why Django?**

- **It’s well-established**
- Django is time- and crowd-tested. It has a big, supportive community accessed through numerous forums, channels, and dedicated websites.

- It’s easy to find help when there’s a problematic function in the code, and to find developers if your company is looking to base the next project on Django.

- Well documented, updated along with the new functions and fixes, so you can easily adapt to changes.

Web Development Framework- Django (contd…)

- **What can be Django be used for?**

- Django was first created to power a web application for a newspaper publisher, the Lawrence Journal-World.

- You can expect it to be amazing at handling projects with volumes of text content, media files, and heavy traffic — or anything else that works like a web-based periodical.

- But the publishing industry is not the framework’s only area of application.

Web Development Framework- Django (contd…)

- **What can be Django be used for?**
- Django is also used to build e-commerce websites and health care and financial applications for transportation and booking, social media sites, and more.

- Here are some of the many project types you can develop using the framework:

  - Financial platforms with features for analyzing and calculating approximate results based on personal data, risk tolerance, the probability of achieving goals.
  - B2B CRM systems for handling communication between businesses
  - Android and iOS mobile apps that support web application
  - Real estate property evaluation systems etc…

Web Development Framework- Django (contd…)

- **What can be Django be used for?**
- Some companies choose to base their projects on more than one framework. Django can also be used to create separate features, such as:

  - An emailing system for sending notifications to users
  - A filtering system with advanced logic and dynamically changing rules
  - Algorithm-based generators
  - Data-analysis tools
  - Interfaces for managing investment funds
  - Admin dashboards
  - Photo-based verification systems etc…

Web Development Framework- Django (contd…)

- **Famous companies that use the Django framework**

Instagram: A popular social network that deals with a great deal of media data and user interactions. Django enables the functionality that makes the web application work seamlessly, add new features, and fix issues in no time.

Spotify: A large media library with huge volume of information that allows users to listen to music free of charge or on an ad-free subscription basis. On the technical side of things, Spotify also uses machine learning, where Python is one of the best choices. The creators chose to combine it with the Django framework.

Web Development Framework- Django (contd…)

- **Famous companies that use the Django framework**

The Washington Post: It's no wonder that The Washington Post would use Django to handle its heavy traffic, since the framework itself was created to power an online newspaper. Several other online periodicals also use Django.

Dropbox: A cloud technology for file storage requires high-performance functionality. Django provides Dropbox with the tools it needs to provide sharing and synchronization, as well as scalability.

Web Development Framework- Django (contd…)

Django is a great addition to projects that need to handle large volumes of content (e.g., media files), user interactions or heavy traffic, or deal with complex functions or technology (e.g., machine learning).

Yet it is simple enough for smaller-scale projects, or if you intend to scale your project to a much higher level.

That’s why Django is used by so many companies that vary in size and goals.

Next Lecture Topics

- MVC architecture

- MVT architecture

2CSDE86 Application Development Frameworks (ADF)

Lecture-2

MVC, MVT

7th CSE

Daiwat Vyas & Ajaykumar Patel

Disclaimer

- Some of the content in the ppt is taken from various online sources and reference books and after referring it was considered for including it in the ppt slides.

Traditional Web Application Development

- **How traditional web application development worked?**

Traditional Web Application Development

- **How traditional web application development worked?**

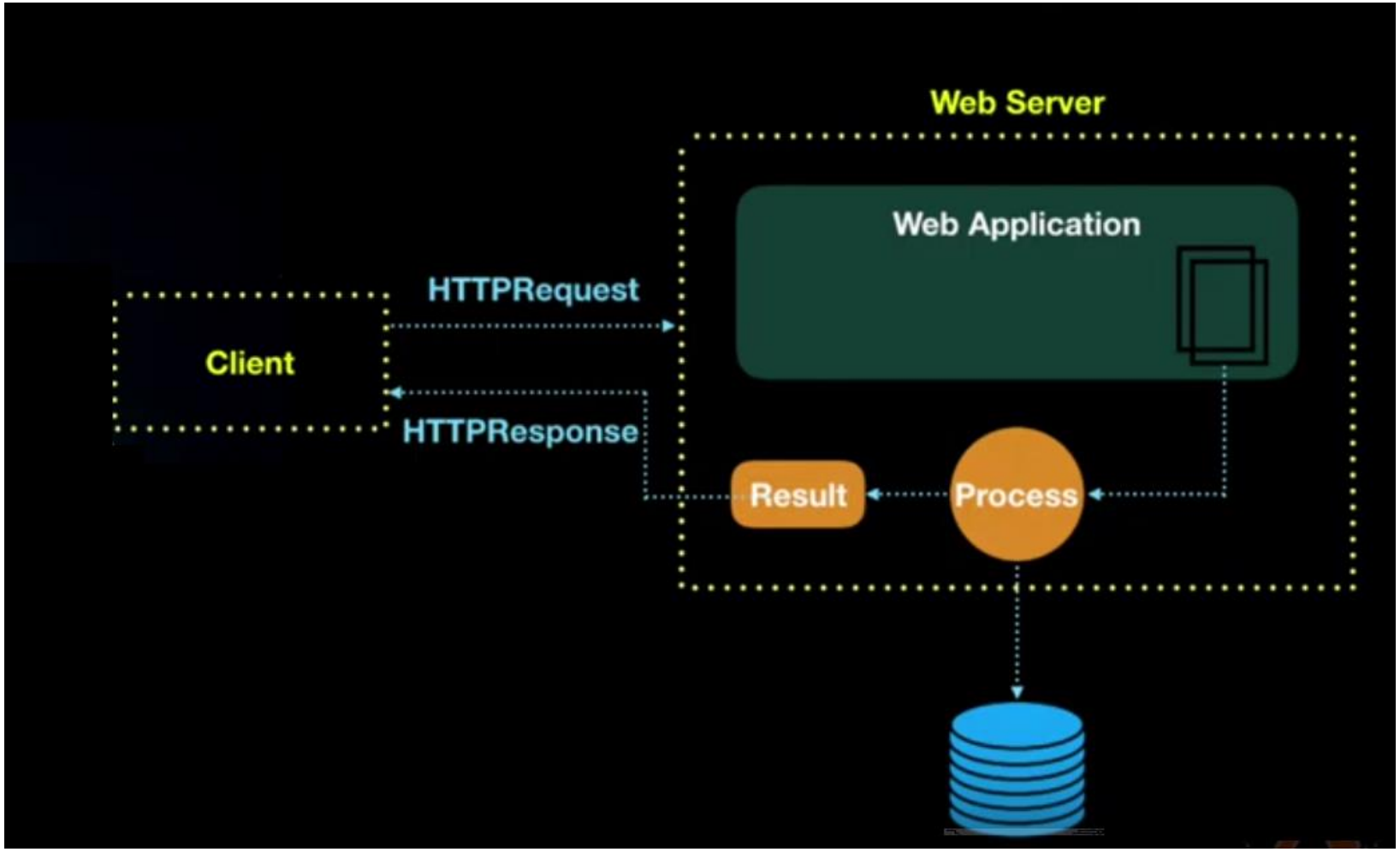


Figure:-1 Traditional Web Development Work-Flow

Traditional Web Application Development

- **How traditional web application development worked?**

- Client sends request to web-server. The request will be given as a HttpRequest to the web-server.

- Server accepts the request and identifies the requester web application and then the requested page will be processed within the web-server. (It can reject the request too).

- If there is any requirement to interact with the database, then the process interacts with the database & generate the result which will be provided as a HttpResponse to the requester client.

Model-**V**iew-**C**ontroller (MVC)

- **History of MVC**

- Tygrve Reenskaug invented MVC. The first reports on MVC were written when he was visiting scientist at Xerox Palo Alto Research Laboratory (PARC) in 1978/79.

- At first, MVC was called “Thing Model View Editor” but rapidly changed it to “ Model View Controller.

- The goal of Tygrve was to solve the problem of users controlling a large and complex data set. The practice of MVC has changed over the years. Since the MVC pattern was invented before web browsers, initially was used as an architectural pattern for graphical user interfaces(GUI).

Model-**V**iew-**C**ontroller (MVC)

- **History of MVC (contd…)**

- Currently MVC is used for designing web applications.

- Some web frameworks that use MVC concept:

- Ruby on Rails
- Laravel
- CherryPy
- Symphony, etc

Model-**V**iew-**C**ontroller (MVC)

- What is MVC?

- MVC is an architectural design pattern that separated an application into three logical components:

  - Model

  - View

  - Component

Model-**V**iew-**C**ontroller (MVC)

- What is MVC? Contd…

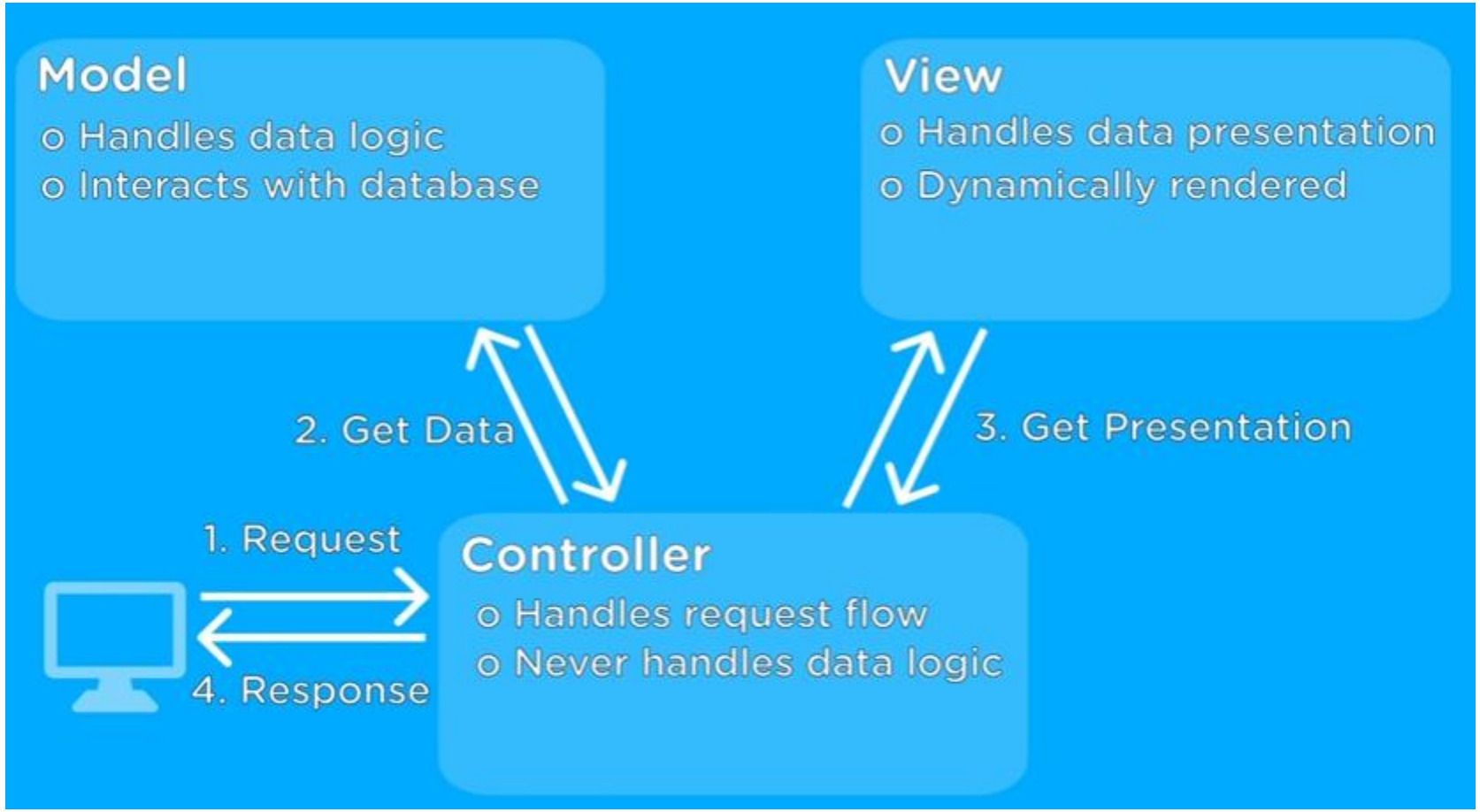


Figure:-2 MVC Architecture Representation

Model-**V**iew-**C**ontroller (MVC)

- What is MVC? Contd…

- What are the key points that you observe by referring the MVC architecture representation in the Figure:-2 in previous slide?

Model-**V**iew-**C**ontroller (MVC)

- What is MVC? Contd…

- What are the key points that you observe by referring the MVC architecture representation in the Figure:-2 in previous slide?

- Model and View never interact with each other directly.

- Any interactions between the Model and View happens through the Controller.

- What can be the advantage of having the Controller between the Model and View?

Model-**V**iew-**C**ontroller (MVC)

- What is MVC? Contd…

- What can be the advantage of having the Controller between the Model and View?

- Presentation of data and logic of data are completely separated.

- Helps in creating complex applications more easily.

Model-View-Controller (MVC)

- What is MVC? Contd…

- Example

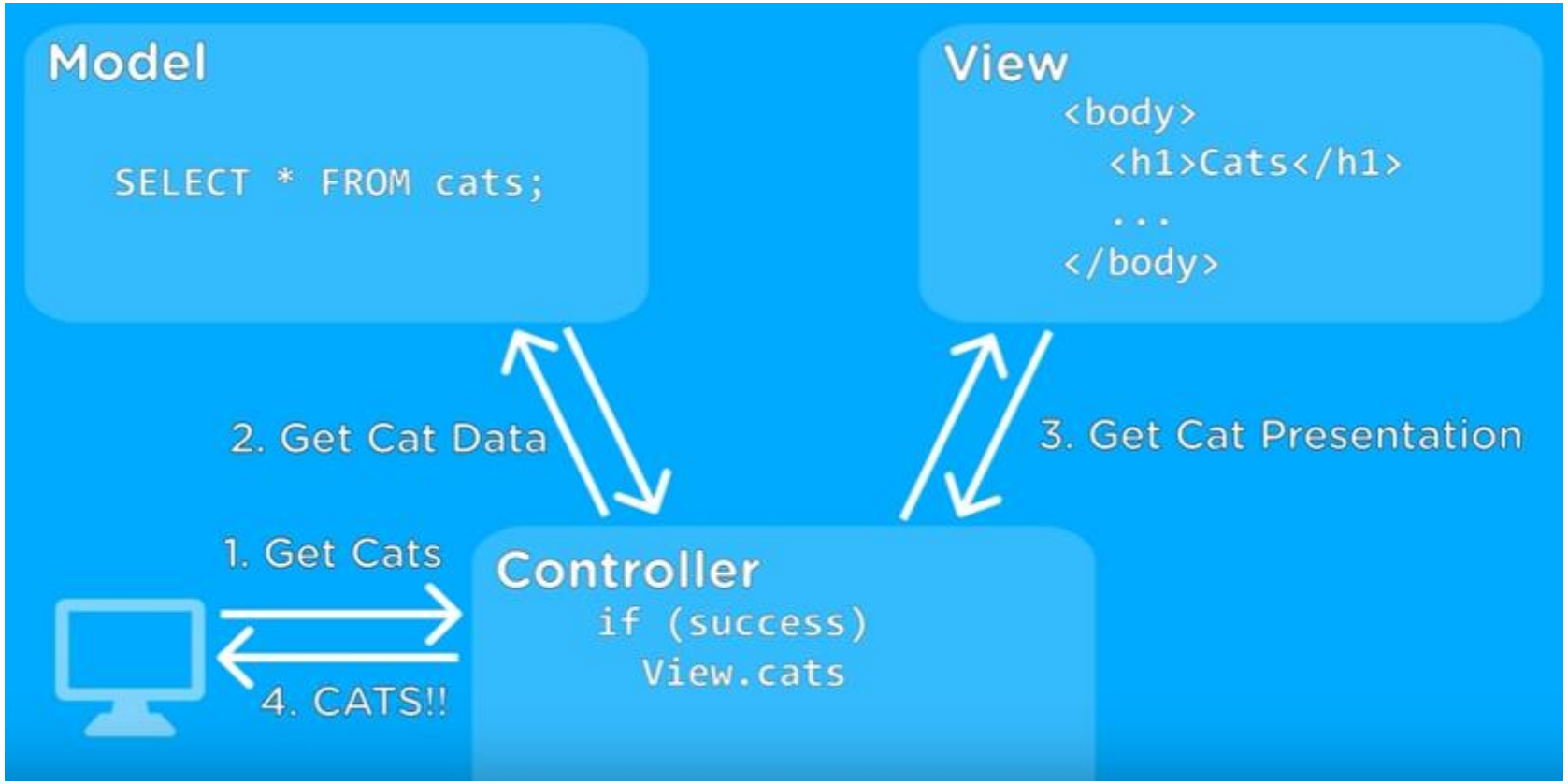


Figure:-3 Example for understanding MVC

Model-View-Controller (MVC)

- What is MVC? Contd…
- In context to previous example, What if the model returns an error?

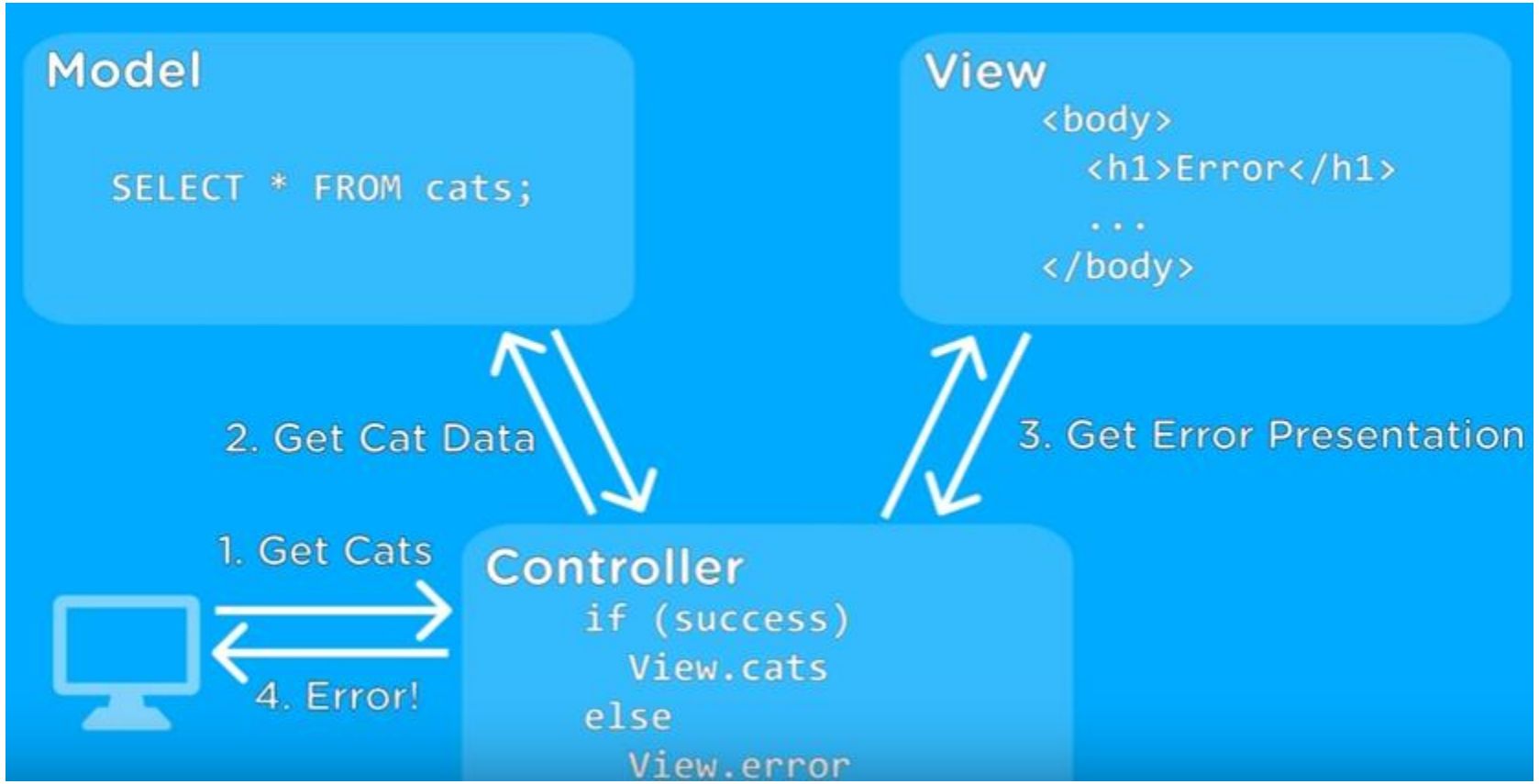


Figure:-4 Example for understanding how error is handled

Model-**V**iew-**C**ontroller (MVC)

- **What is MVC? Contd…**
- **Model:-**

- It is known as the lowest level which means it is responsible for maintaining data.
- Handle data logically so it basically deals with data.
- The model is actually connected to the database so anything you do with data.
- Adding or retrieving data is done in the model component.
- It responds to the controller requests because the controller never talks to the database by itself.
- The model talks to the database back and forth and then it gives the needed data to the controller. **Note: the model never communicates with the view directly.**

Model-**V**iew-**C**ontroller (MVC)

- **What is MVC? Contd…**
- **View:-**

- Data representation is done by the view component.
- It actually generates UI or user interface for the user.
- So at web applications when you think of the view component just think the Html/CSS part.
- Views are created by the data which is collected by the model component
- but **these data aren’t taken directly but through the controller, so the view only speaks to the controller.**

Model-**V**iew-**C**ontroller (MVC)

- **What is MVC? Contd…**
- **Controller:-**

- It is known as the main component because the controller is the component that enables the interconnection between the views and the model so it acts as an intermediary.

- The controller doesn’t have to worry about handling data logic, it just tells the model what to do.
- After receiving data from the model it processes it and then it takes all that information it sends it to the view and explains how to represent to the user. **Note: Views and Models can not talk directly.**

Model-**V**iew-**C**ontroller (MVC)

- What is MVC? Contd…

- In context to an application using MVC architecture:

- Model – Handles all the Data

- View- Handles all the Presentation

- Controller- Tells the Model and View What to do

Model-**V**iew-**C**ontroller (MVC)

- To summarize MVC:

- The **Model** contains only the pure application data, it contains no logic describing how to present the data to a user.

- The **View** presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.

- The **Controller** exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.

Model-**V**iew-**C**ontroller (MVC)

- To summarize MVC: (contd…) **key points**

- MVC is a popular way of organizing your code.
- The big idea behind MVC is that each section of your code has a purpose, and those purposes are different.

- Some of your code holds the data of your app, some of your code makes your app look nice, and some of your code controls how your app functions.

- MVC is a way to organize your code's core functions into their own, neatly organized boxes.
- This makes thinking about your app, revisiting your app, and sharing your app with others much easier and cleaner.

Model-**V**iew-**C**ontroller (MVC)

- **Analogy to understand MVC**

- MVC is a way to think about how an web application works.
- It’s kind of like how you prepare dinner for your party. **You have a fridge full of food, which is like the Model.** The fridge (Model) contains the raw materials we will use to make dinner.
- You also probably have a recipe or two. **A recipe (assuming you follow it exactly) is like the Controller of dinner you are preparing for your party**. Recipes dictate which stuff in the fridge you’ll take out, how you’ll put it together, and how long you need to cook it.
- Then, you have table-settings, silverware, dinner set etc., which are what your hungry friends and family use to eat dinner. **Table-top items are like the View. They let your guests interact with your Model and Controller’s creation.**

Model-**V**iew-**C**ontroller (MVC)

- **Another Analogy to understand MVC (contd…)**

- Let's assume you go to a restaurant. You will not go to the kitchen and prepare food which you can surely do at your home. Instead, you just go there and wait for the waiter to come on.
- Now the waiter comes to you, and you just order the food. The waiter doesn't know who you are and what you want he just written down the detail of your food order.
- Then, the waiter moves to the kitchen. In the kitchen waiter not prepare your food.
- The cook prepares your food. The waiter is given your order to him along with your table number.
- Cook then prepares food for you. He uses ingredients to cook the food. Let's assume that your order was for vegetable sandwich. Then he needs bread, tomato, potato, capsicum, onion, bit, cheese, etc. which he sources from the refrigerator.
- Cook finally hand’s over the food to the waiter. Now it is the job of the waiter to move this food outside the kitchen.
- Now waiter knows which food you have ordered and how they are served.

Model-**V**iew-**C**ontroller (MVC)

- **Another Analogy to understand MVC (contd…)**

- In the previous analogy:

- View= You
- Waiter= Controller
- Cook= Model
- Refrigerator= Data

Model-**V**iew-**C**ontroller (MVC)

- **Can you give a real life example application which can use MVC?**

Model-**V**iew-**C**ontroller (MVC)

Real life example to understand MVC

- MVC is helpful when planning your app, because it gives you an outline of how your ideas should be organized into actual code.
- For instance, let's imagine **you're creating a To-do list app**. This app will let users create tasks and organize them into lists.
- The **Model** in a To-Do app might define what a "task" is and that a "list" is a collection of tasks.
- The **View** code will define what the To-Do's and lists looks like, visually. The tasks could have large font, or be a certain color.
- Finally, the **Controller** could define how a user adds a task, or marks another as complete. The Controller connects the View's add button to the Model, so that when you click "add task," the Model adds a new task.

Model-**V**iew-**C**ontroller (MVC)

- Let us take an example in Java, to understand MVC:

- MVCPattern.java

Model-View-Controller (MVC)

```
class Student
{
    private String rollNo;
    private String name;
    public String getRollNo()
    {
        return rollNo;
    }
    public void setRollNo(String rollNo)
    {
        this.rollNo = rollNo;
    }

    public String getName()
    {
        return name;   }
```

Model-View-Controller (MVC)

```
class StudentView
{
    public void printStudentDetails(String studentName, String studentRollNo)
    {
        System.out.println("Student: ");
        System.out.println("Name: " + studentName);
        System.out.println("Roll No: " + studentRollNo);
    }
}
```

Model-View-Controller (MVC)

```
class StudentController
{
    private Student model;
    private StudentView view;

    public StudentController(Student model, StudentView view)
    {
        this.model = model;
        this.view = view;
    }

    public void setStudentName(String name)
    {
        model.setName(name);
    }

    public String getStudentName()
    {
        return model.getName();
    }
```

Model-View-Controller (MVC)

```
public void setStudentRollNo(String rollNo)
    {
        model.setRollNo(rollNo);
    }

    public String getStudentRollNo()
    {
        return model.getRollNo();
    }

    public void updateView()
    {
        view.printStudentDetails(model.getName(), model.getRollNo());
    }
}
```

Model-View-Controller (MVC)

```
class MVCPattern

{

    public static void main(String[] args)

    {

        Student model = retriveStudentFromDatabase();

        StudentView view = new StudentView();

        StudentController controller = new StudentController(model, view);

        controller.updateView();

        controller.setStudentName("Dvijesh Bhatt");



        controller.updateView();

    }
```

Model-**V**iew-**C**ontroller (MVC)

- **Just to conclude:**

- MVC is a framework for thinking about programming, and for organizing your program's files.

- To signify the idea that your code should be organized by its function, developers will create folders for each part of MVC.

- The idea that apps should be divided based on the *function* of each part of the code is sometimes referred to as *separation of concerns.*

Model-**V**iew-**C**ontroller (MVC)

- **Just to conclude: (contd…)**

- MVC gives you a starting place to translate your ideas into code, and it also makes coming back to your code easier, since you will be able to identify which code does what.

- In addition, the organizational standard MVC promotes makes it easy for other developers to understand your code.

Model-**V**iew-**C**ontroller (MVC)

- **Just to conclude: (contd…)**

- Thinking about how code interacts with other code is a significant part of programming, and learning to collaborate with other developers is an important skill.

- Taking the time to think about how your app fits into the MVC framework will level-up your skills as a developer by teaching you both.

Post Lecture Tasks

- **Identify, why MVC was introduced to replace traditional application development model?**
- **List out advantages and dis-advantages of MVC architecture.**
- **Compare & Contrast 3-Tier architecture and MVC architecture.**

Model-**V**iew-**T**emplate (MVT)

- **What is MVT?**

- **M-** Model

- **V-** View

- **T-** Template

- Django prefers to use its own logic implementation in its web app and hence its framework handles all the controller parts by itself.

Model-**V**iew-**T**emplate (MVT)

- **What is MVT? (contd…)**

- **Model:-** It is same as in MVC, here as well it has the same functionality of providing the interface for the data stored in the database**.**

- **View:-** In Django, Views act as a link between the **Model data** and the **Templates.**

  - Note: Just like the controller in MVC, views in Django MVT are responsible for handling all the business logic behind the web app. It acts as a bridge between Models and Templates
  - It sees the user request, retrieves appropriate data from the database, then renders back the template along with retrieved data.
  - Therefore there is no separate controller in Django MVT architecture and everything is based on **Model -View – Template** itself and hence the name **MVT**.

Model-**V**iew-**T**emplate (MVT)

- **What is MVT? (contd…)**

- **Template:- Just as View** in MVC, Django uses templates in its framework. Templates are responsible for the entire User Interface. It handles all the static parts of the webpage along with the HTML, which the users visiting the webpage will perceive.

Model-**V**iew-**T**emplate (MVT)

- **What is MVT? (contd…)**

- **Template:- Just as View** in MVC, Django uses templates in its framework. Templates are responsible for the entire User Interface.

- It handles all the static parts of the webpage along with the HTML, which the users visiting the webpage will perceive.

- Templates are used to specify a structure for an output.
- Data can be populated in a template using placeholders. It defines how the data is presented. An example is a Generic list view that we can use to display a set of records from the database.

Model-View-Template (MVT)

- **Difference between MVC and MVT**

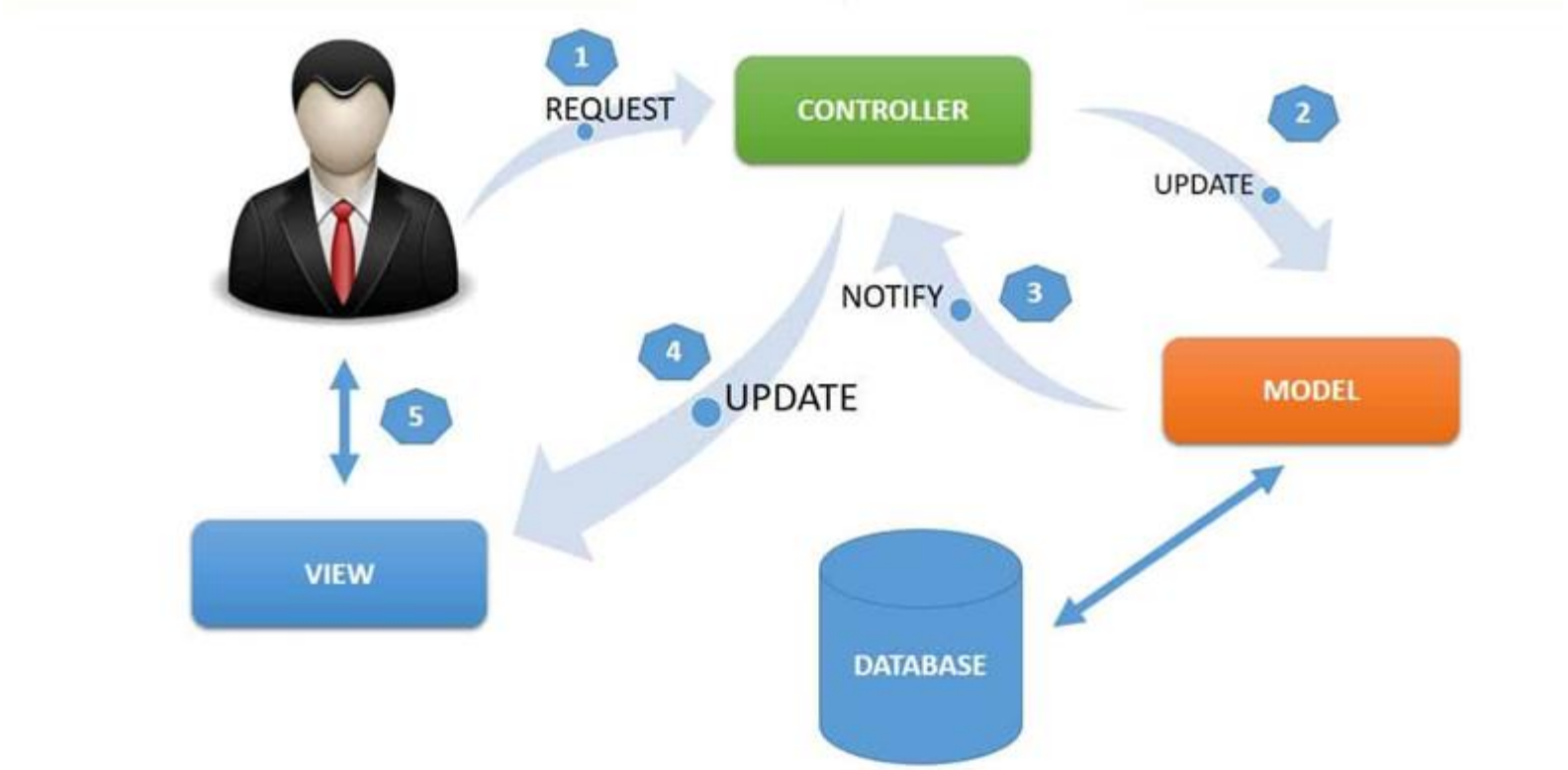


Figure:-5 MVC Architecture

Model-View-Template (MVT)

- **Difference between MVC and MVT**

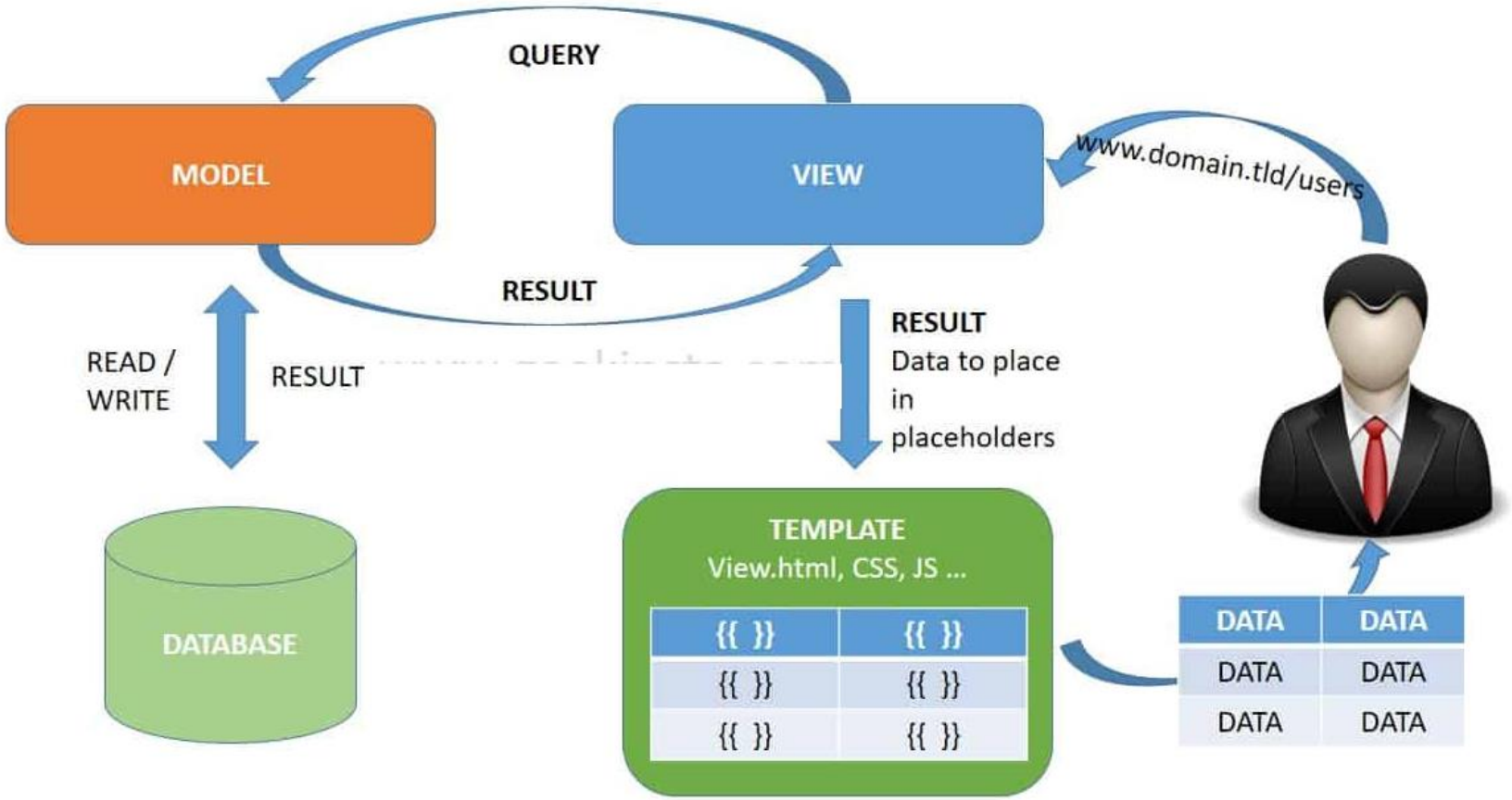


Figure:-6 MVT Architecture

Model-**V**iew-**T**emplate (MVT)

- **Difference between MVC and MVT**

- The main difference between MVC and MVT is that in a Model View Controller pattern, we have to write all the control specific code.

- But in an MVT, the controller part is taken care of by the framework itself.

- Let us take an example to understand this difference.

Model-**V**iew-**T**emplate (MVT)

- **Difference between MVC and MVT (contd…)**

- Assume that you have to display a list of books you have in a library, stored is a table named *books*. In MVC architecture, you have to write the code to fetch the list of books from the database, write the presentation layer (i.e HTML, CSS), map it with a URL and send it to the user.

- But if MVT is followed, you don't have to write any code related to fetching data from the database and mapping it with the URL. All these activities are handled by the framework itself. The only thing that you have to do is to tell the framework what data should be presented to the user (Books table). The framework will then create a view based on the data and send it to the user.

Model-**V**iew-**T**emplate (MVT)

- **Difference between MVC and MVT (contd…)**

- The classic MVC pattern works by managing the state of an application. When a user performs an action or makes a request, an action in the Controller is called. The Controller then either tells Model to make changes and update the View or returns a View based on a Model. Hence we can say that the View is controlled by the Controller and Model.

- MVT takes a slightly different approach. When a user makes an HTTP request, the corresponding View performs a query on the Model and collects the result set from the Model. The View then fills the result in a template and sends it to the user.

Model-**V**iew-**T**emplate (MVT)

- **Difference between MVC and MVT (contd…)**

- MVC helps to organize the core functions of your code neatly. It will be easy to make any changes to your app later as you will be able to identify which code does want.

Model-**V**iew-**T**emplate (MVT)

- **Lab Task for the week – 28 August 2022 to 03 September 2022**

- Compare and Contrast MVC & MVT.
- Django Application Structure

2CSDE86 Application Development Frameworks (ADF)

Lecture-3

Django Project Structure, Django Apps Structure

7th CSE

Daiwat Vyas & Ajaykumar Patel

Disclaimer

- Some of the content in the ppt is taken from various online sources and reference books and after referring it was considered for including it in the ppt slides.

Pre-requisite for this session

- **Students should have already installed Python, vscode and setup Django on their system.**

- **A file named Django Installation Steps was already sent for Lab task, and it had all step by step procedure mentioned for installing Python, vscode and setting up Django in their system.**

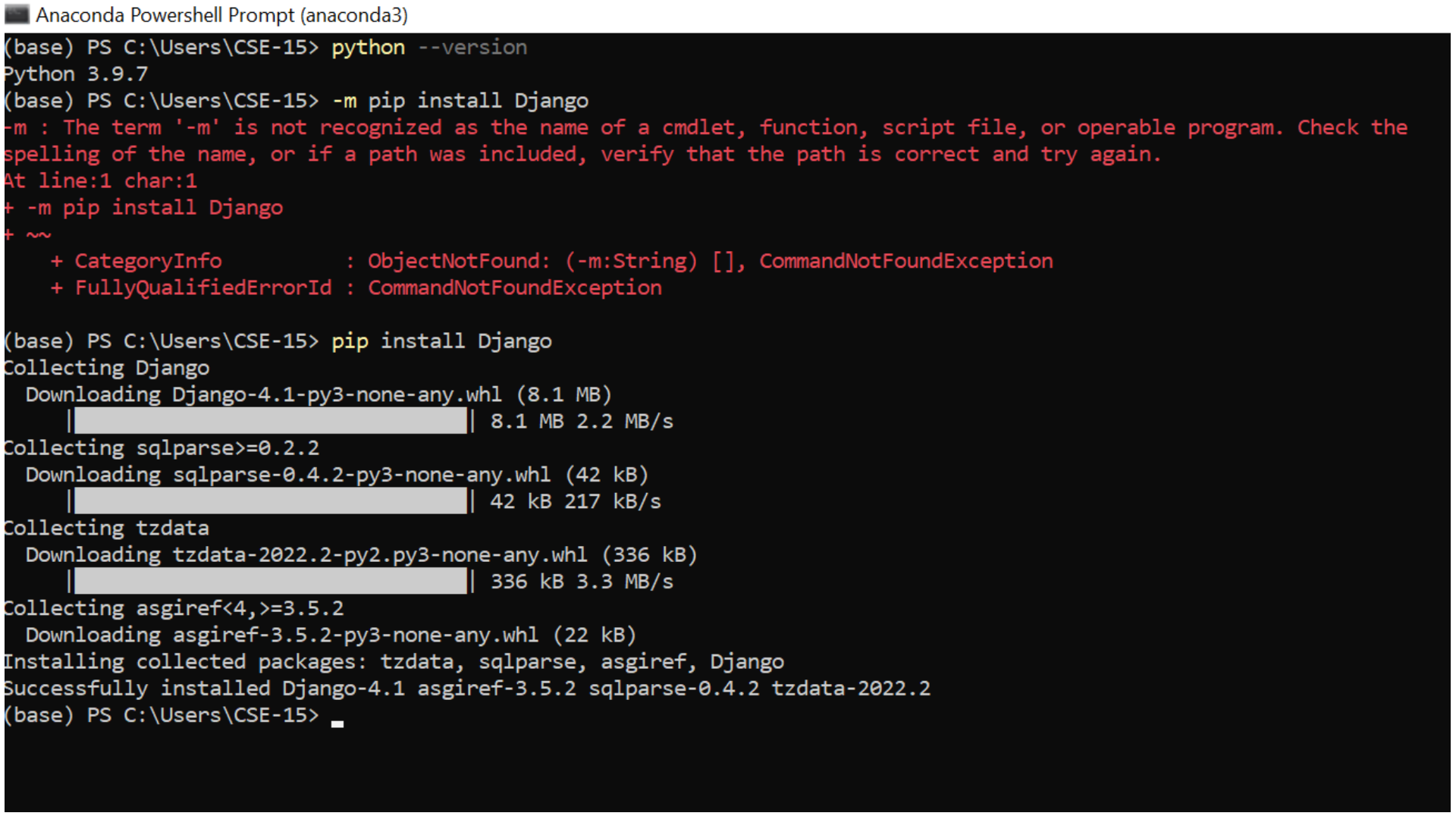
Django Installation

- After installing Python successfully, open Anaconda Prompt and use following command:

- Pip install Django

Django Installation

- Once Django is successfully installed following will appear:

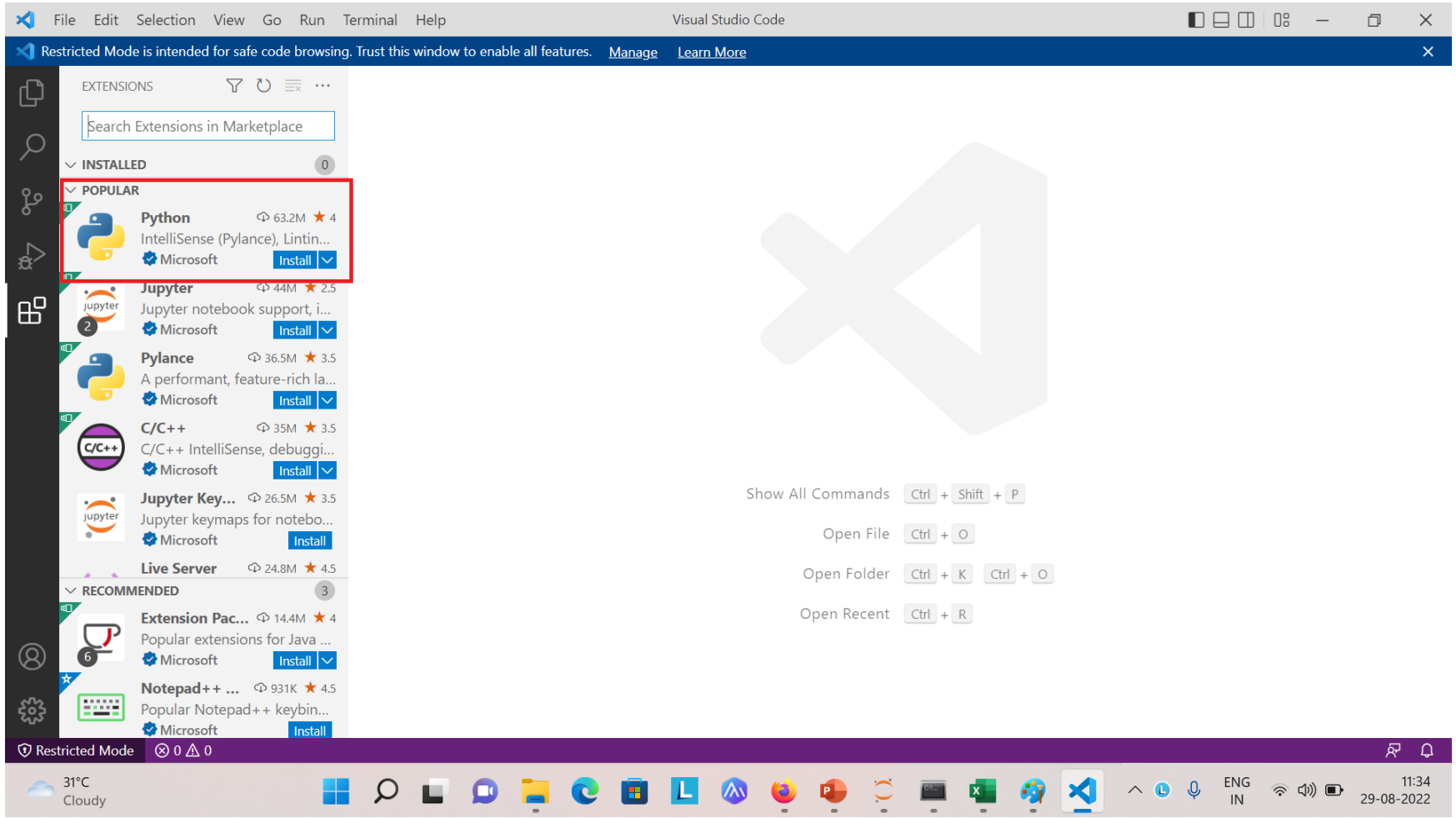


Django Installation

- After this, use following command (This will create a new project):
- *django -admin startproject django_demo*

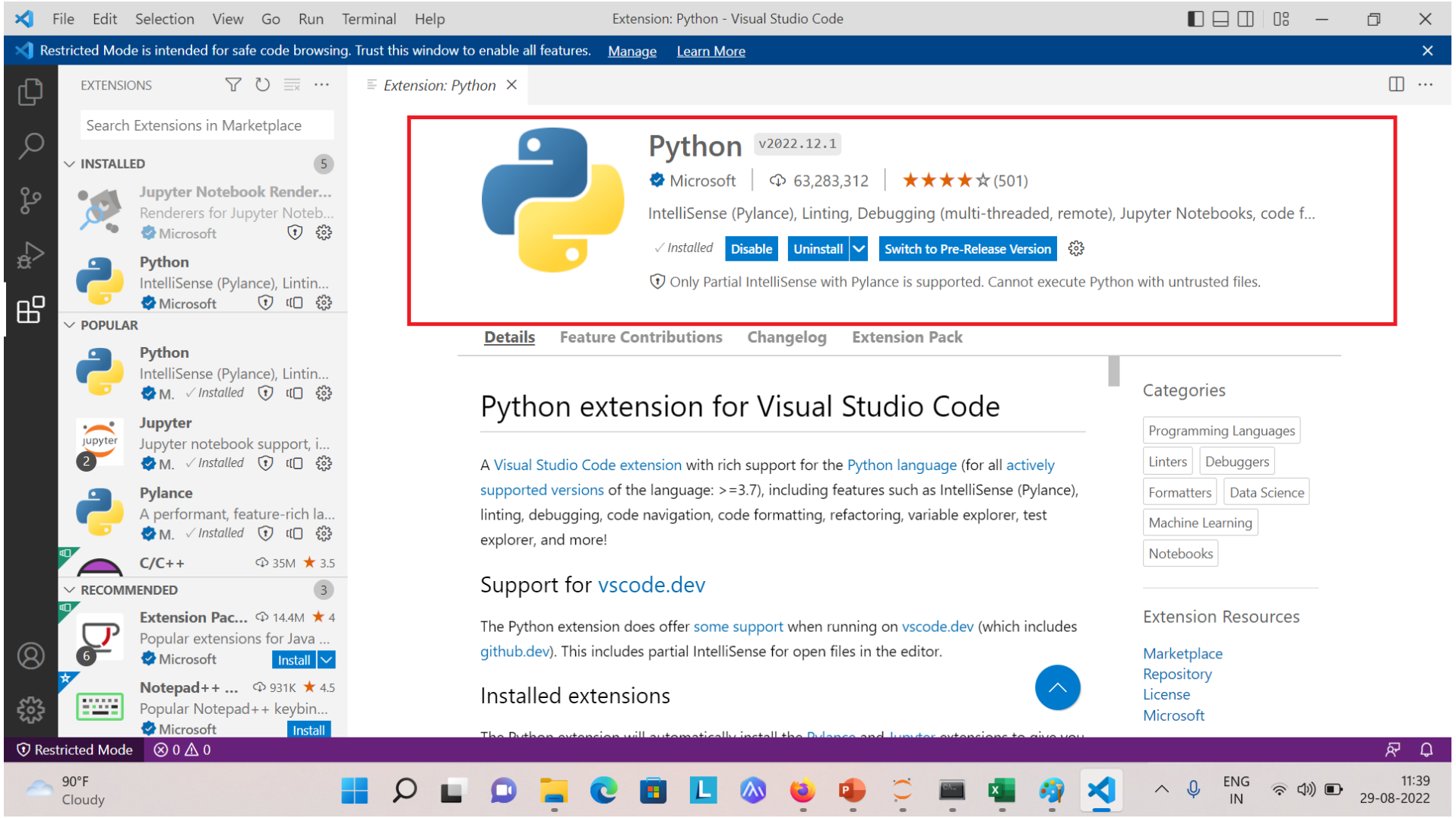
Django Installation

- Install vsCode Editor
- After installing it open vsCode Editor and Click on View menu option and select Extensions.
- From the available extensions, select the Python extension and install it:



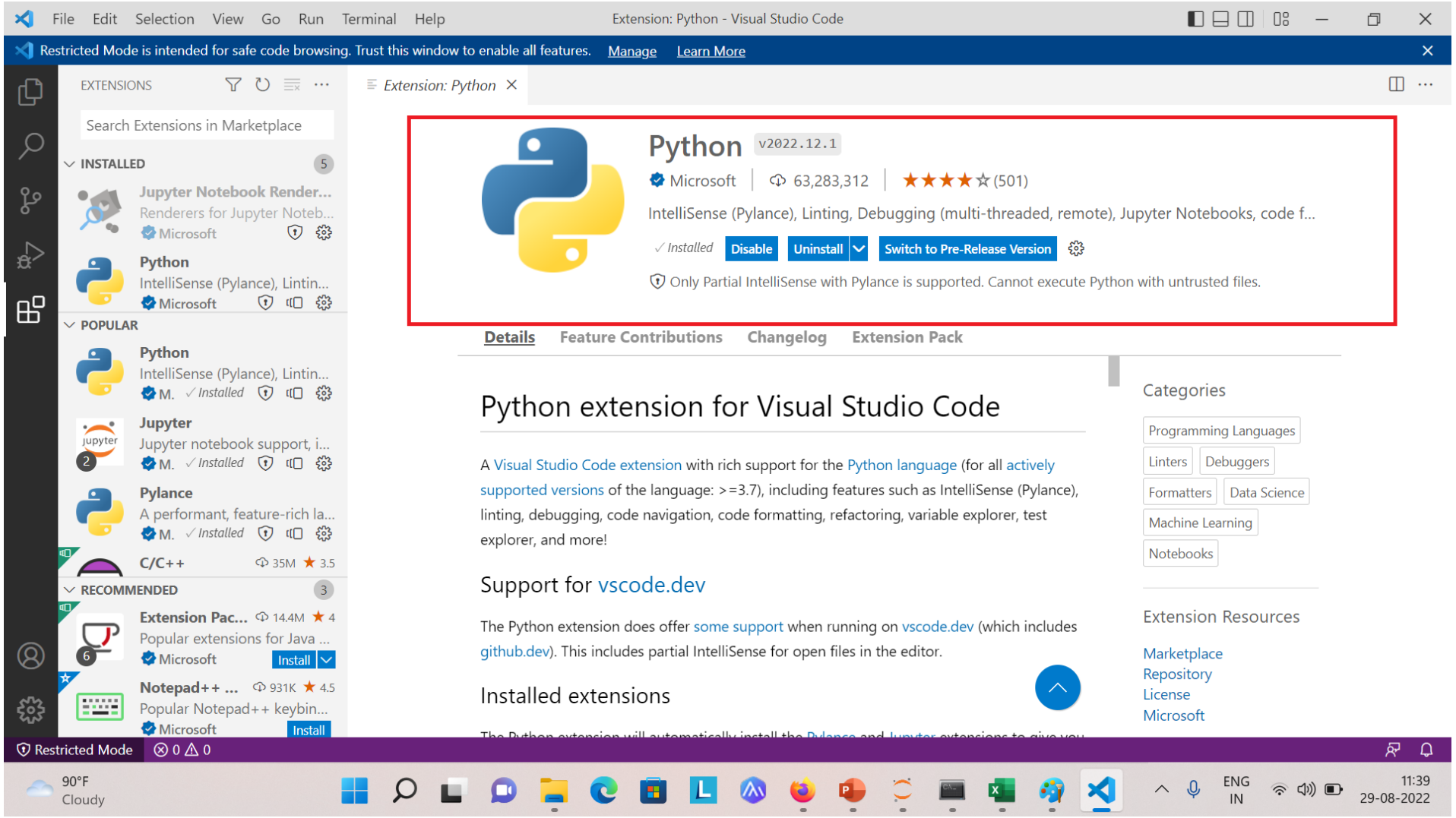
Django Installation

- After it is successfully installed, following will appear:



Django Installation

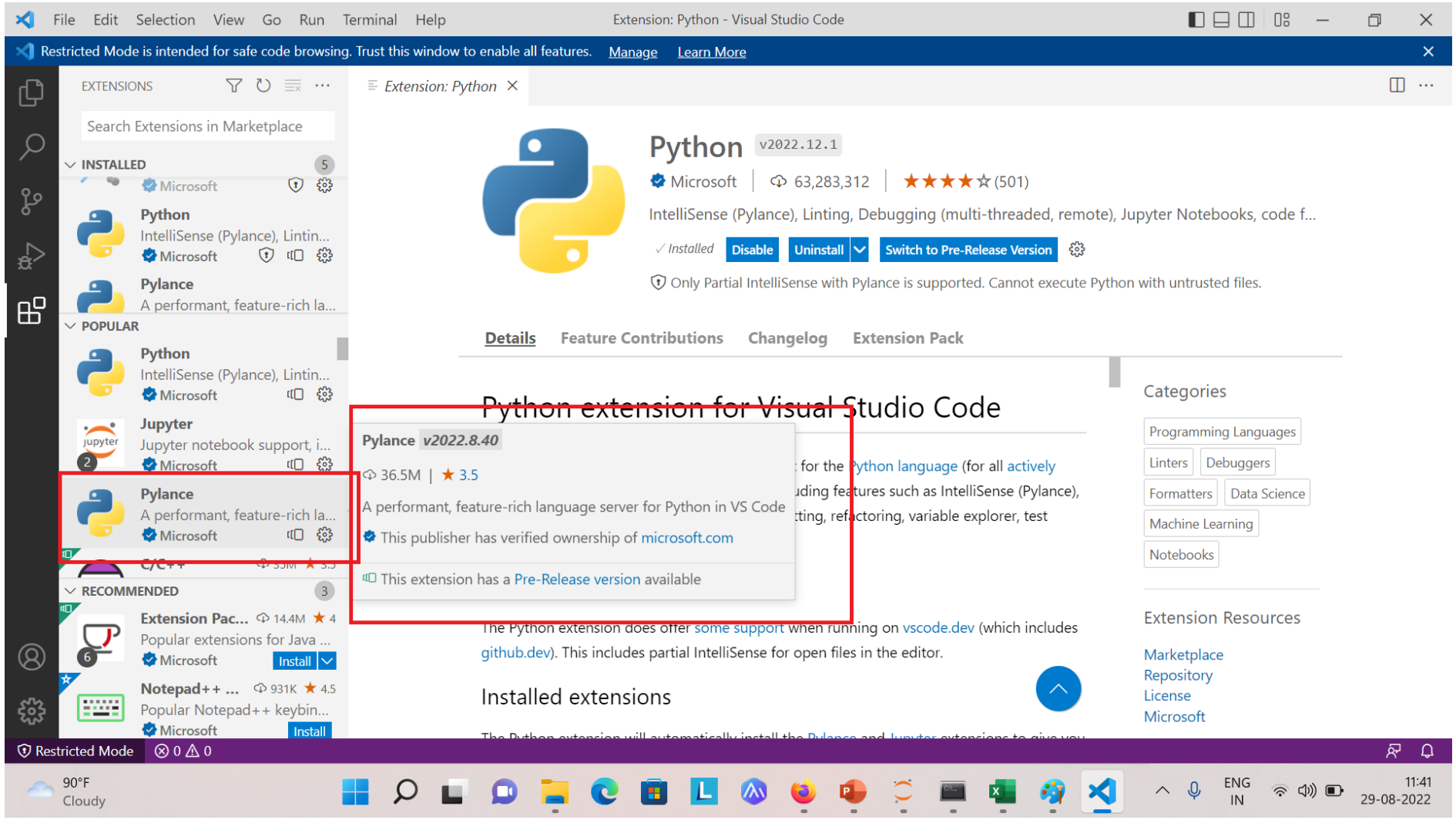
- After it is successfully installed, following will appear:



Django Installation

- Then install Pylance extension:

- Pylance is a new language server for Python, which uses the Language Server Protocol to communicate with VS Code.



Django Installation

- **Pylance Features:**

- With auto-imports, you are now able to get smart import suggestions in your completions list for installed and standard library modules.

- Type information is now available in function signatures and when hovering on symbols, providing you with helpful information to ensure that you are correctly invoking functions, to improve the quality of the code you write.

- Pylance natively supports multi-root workspaces, meaning that you can open multiple folders in the same Visual Studio Code session and have Pylance functionality in each folder.

Django Project Structure

- When you create a Django project, the Django framework itself creates a root directory of the project with the project name on it.

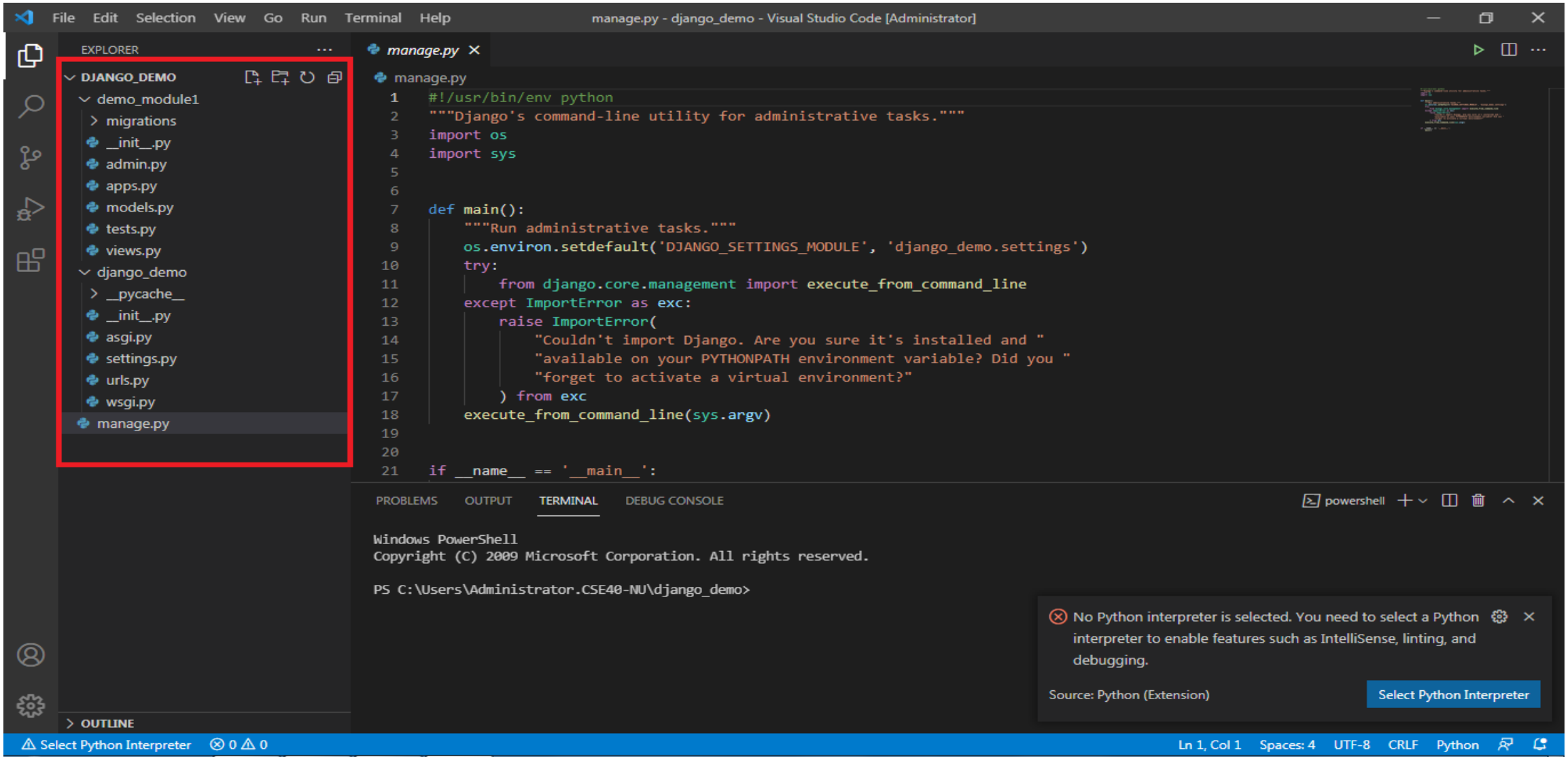
- It contains some files and folder, which provide the very basic functionality to your website and on that strong foundation you will be building your full scaled website.

- By root directory, we mean about the directory which contains your manage.py file.

Django Project Structure (contd…)

- Additional files like db.sqlite, which is a database file may be present when we will be migrating your project.
- Django root directory is the default app which Django provides you.
- It contains the files which will be used in maintaining the whole project.
- The name of Django root directory is the same as the project name you mentioned in django-admin startproject [projectname].
- This root directory is the project’s connection with Django.

Django Project Structure (contd…)



Django Project Structure (contd…)

- **manage.py**

- This file is used basically as a command-line utility and for deploying, debugging, or running our web application.

- This file contains code for **runserver, or makemigrations or migrations**, etc. that we use in the shell.

- **There is no need to make any changes to the file.**

Django Project Structure (contd…)

- **manage.py**

- The file contains the code for starting the server, migrating and controlling the project through command-line.

- This file provides all the functionality as with the django-admin and it also provides some project specific functionalities

Django Project Structure (contd…)

- **manage.py**

- **runserver:** This command is used to run the server for our web application.

- **Migrate:** This is used for applying the changes done to the models into the database. That is if we make any changes to the database then we use **migrate** command.
- This is used the first time we create a database.

Django Project Structure (contd…)

- **manage.py**

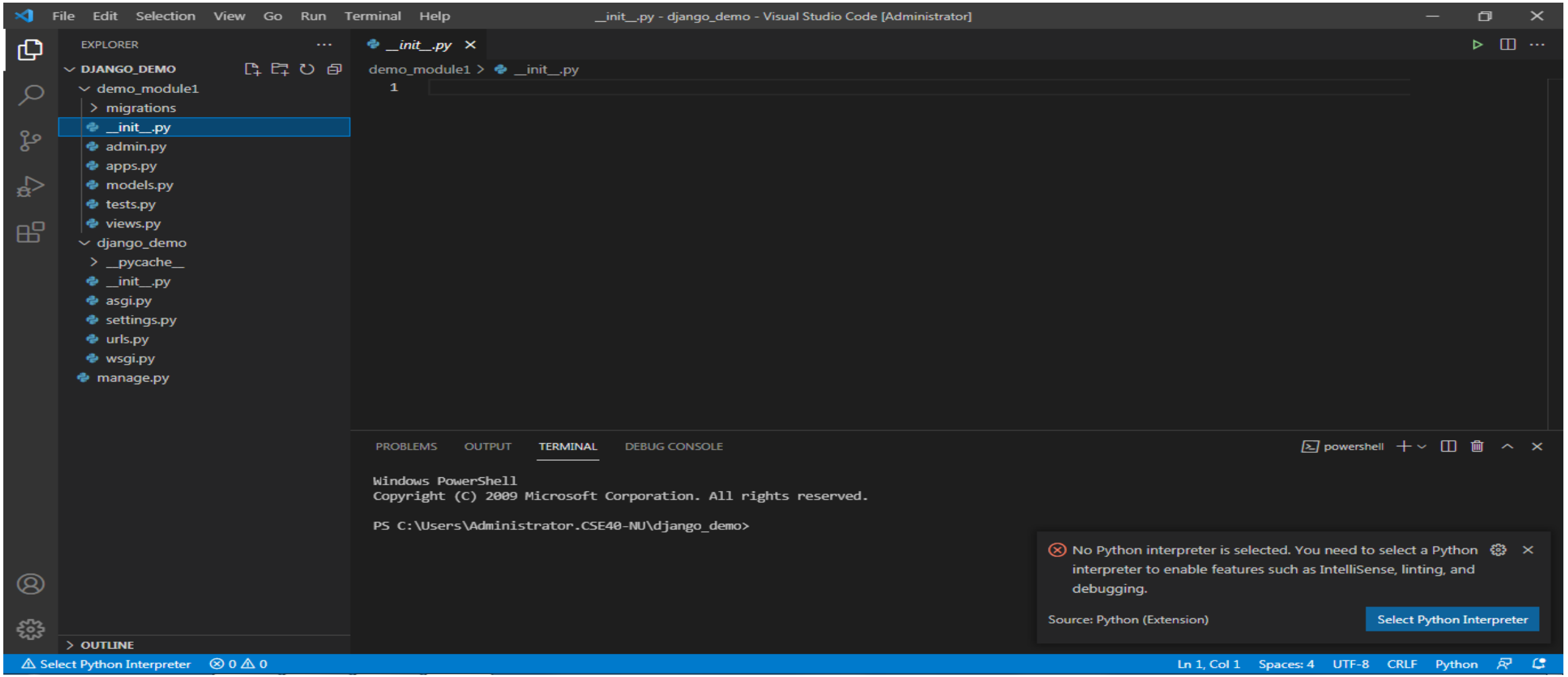
- **Makemigration:** this is done to apply new migrations that have been carried out due to the changes in the database.

- It is the command for integrating your project with files or apps you have added in it.

- This command will actually check for any new additions in your project and then add that to the same.

Django Project Structure (contd…)

- _init_.py



Django Project Structure (contd…)

- **_init_.py**

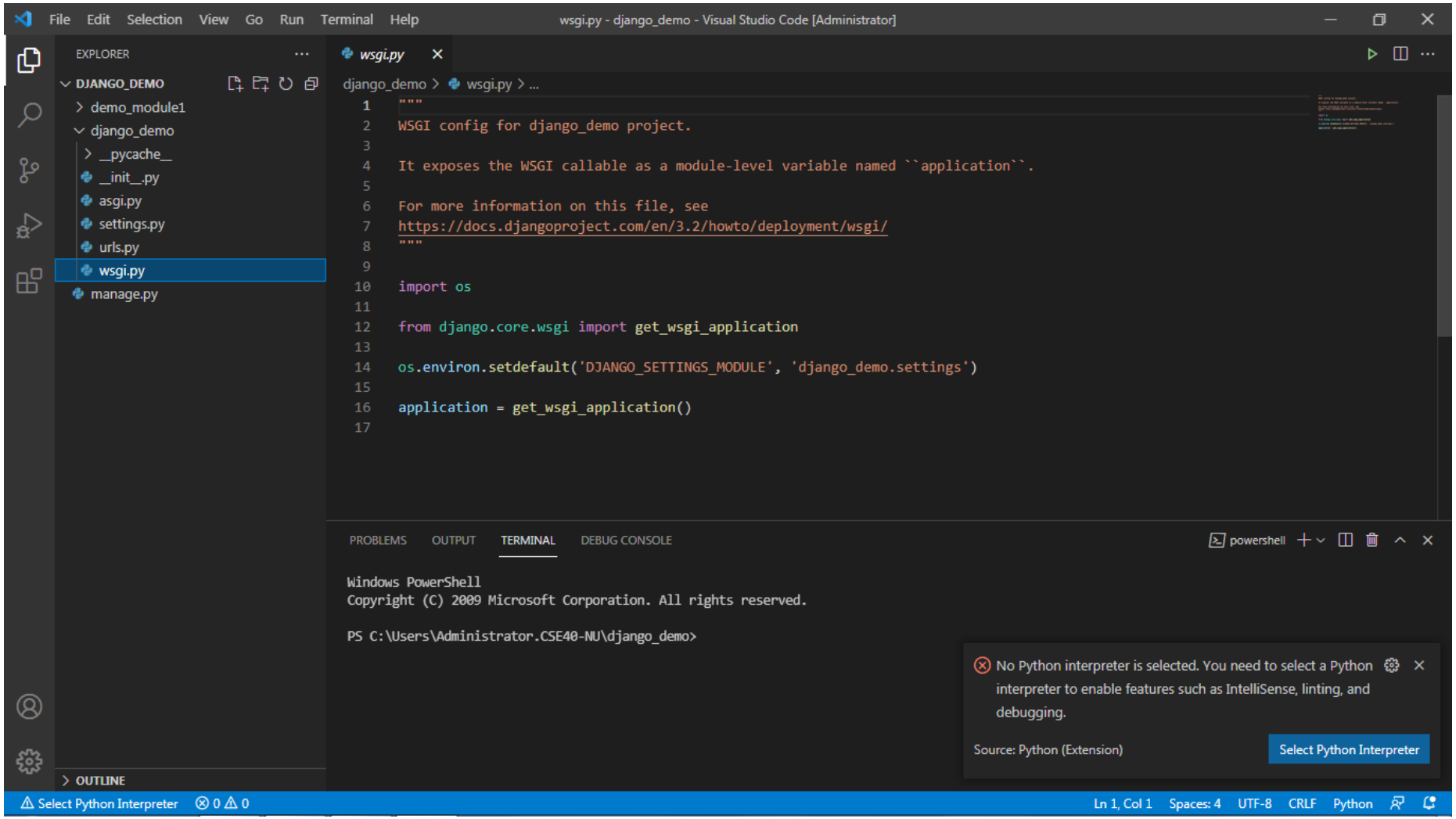
- This file remains empty and is present their only to tell that this particular **directory**(in this case django_project) is a **package**.

- **There is no need to make any changes to the file.**

- The function of this file is to tell the Python interpreter that this directory is a package and involvement of this __init.py_ file in it makes it a python project.

Django Project Structure (contd…)

- **wsgi.py**



Django Project Structure (contd…)

- **wsgi.py**

- This file mainly concerns with the **WSGI server** and is used for deploying your applications on to servers like Apache etc.

- **WSGI, short for Web Server Gateway Interface** can be thought of as a specification that describes how the servers interact with web applications.

- **There is no need to make any changes to the file.**

Django Project Structure (contd…)

- **wsgi.py**

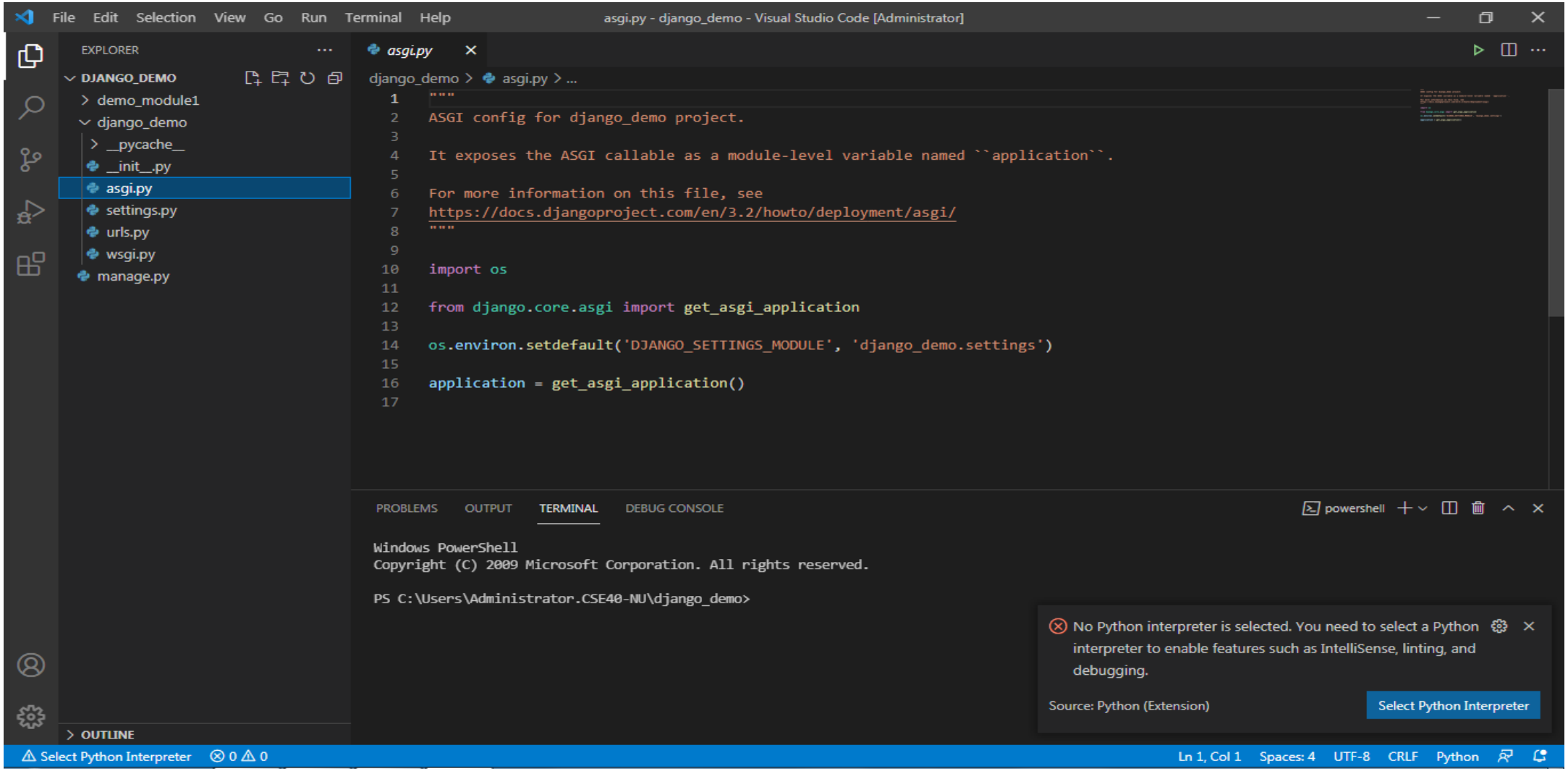
- Django is based on python which uses WSGI server for web development.
- This file is mainly concerned with that and we will not be using this file much.

- wsgi is still important though if you want to deploy the applications on Apache servers or any other server because Django is still backend and you will need its support with different servers.

- But you need not to worry because for every server there is a Django middleware out there which solves all the connectivity and integration issues and you just have to import that middleware for your server, it’s very simple.

Django Project Structure (contd…)

- asgi.py



Django Project Structure (contd…)

- **asgi.py**

- In the newer versions of Django, you will also find a file named as **asgi.py** apart from **wsgi.py**. **ASGI** can be considered as a succeeder interface to the **WSGI**.

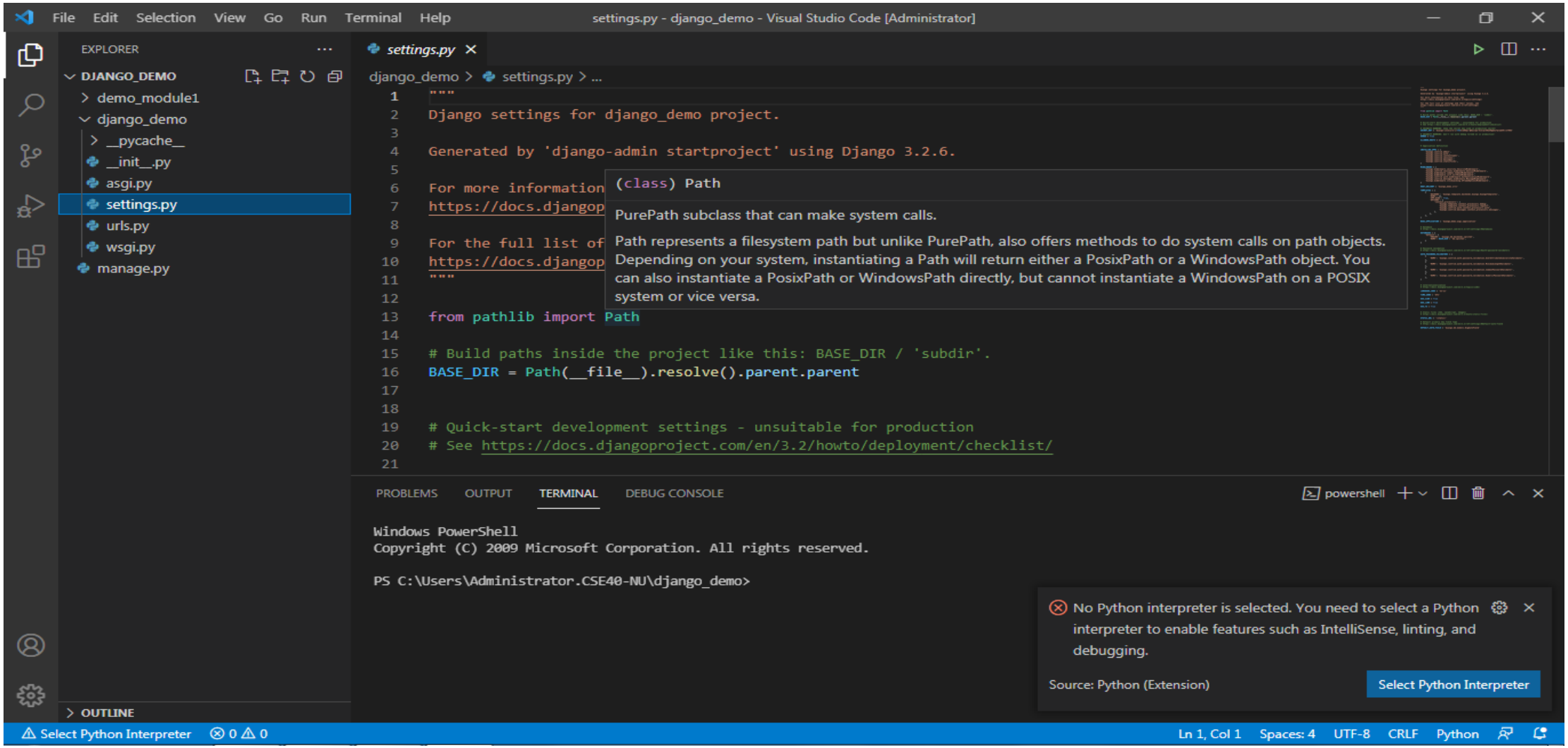
- **ASGI, short for Asynchronous Server Gateway interface** also has the work similar to WSGI but this is better than the previous one as it gives better freedom in Django development.

- That’s why WSGI is now being increasingly replaced by **ASGI.**

- **There is no need to make any changes to the file.**

Django Project Structure (contd…)

- **settings.py**



Django Project Structure (contd…)

- **settings.py**

- The settings.py is the main file where we will be adding all our applications and middleware applications.

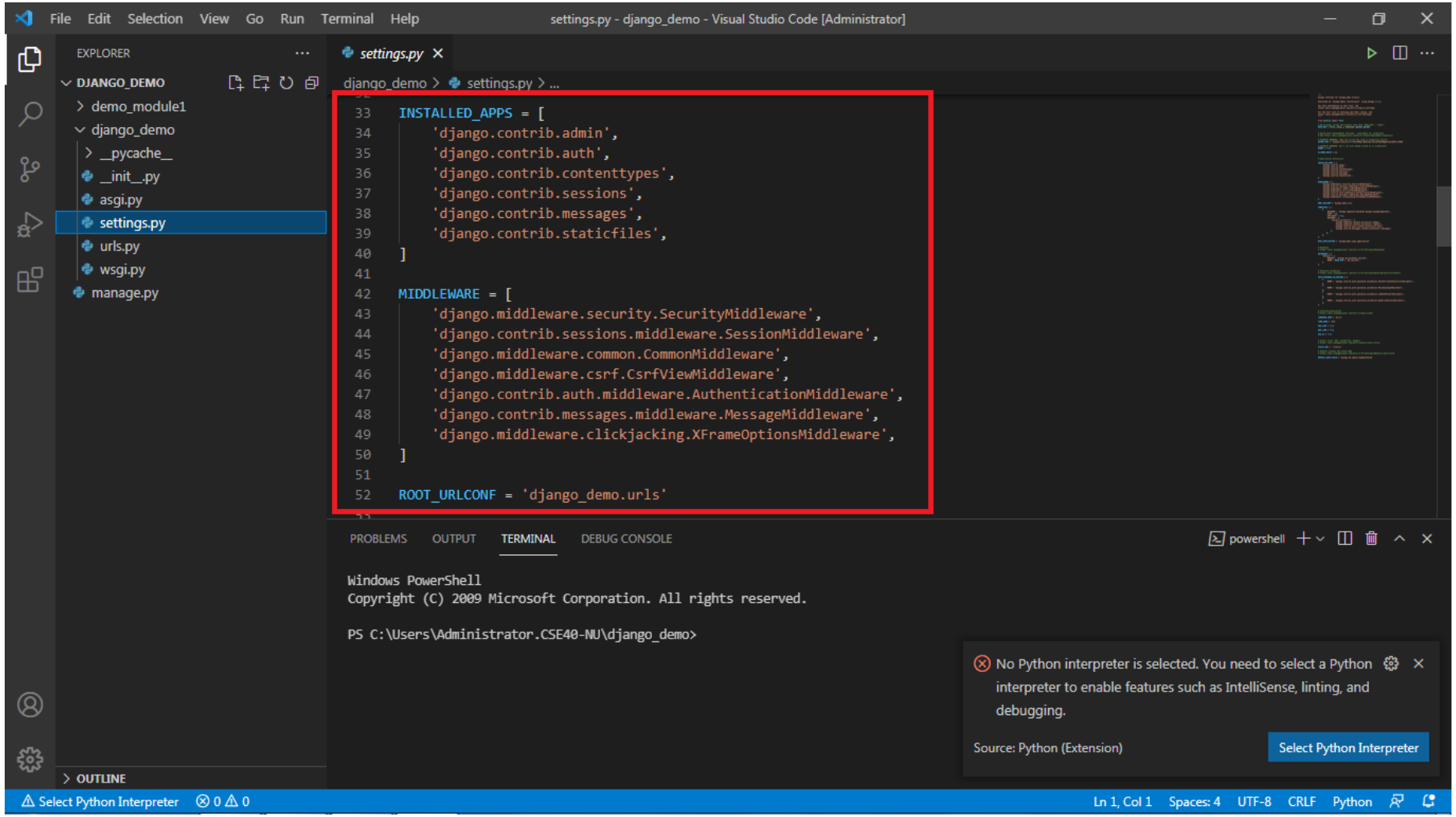
- As the name suggests this is the main settings file of the Django project.

- This file contains the installed applications and middleware information which are installed on this Django project.

- Every time you install a new app or custom application you will be adding that in this file.

Django Project Structure (contd…)

- **settings.py**



Django Project Structure (contd…)

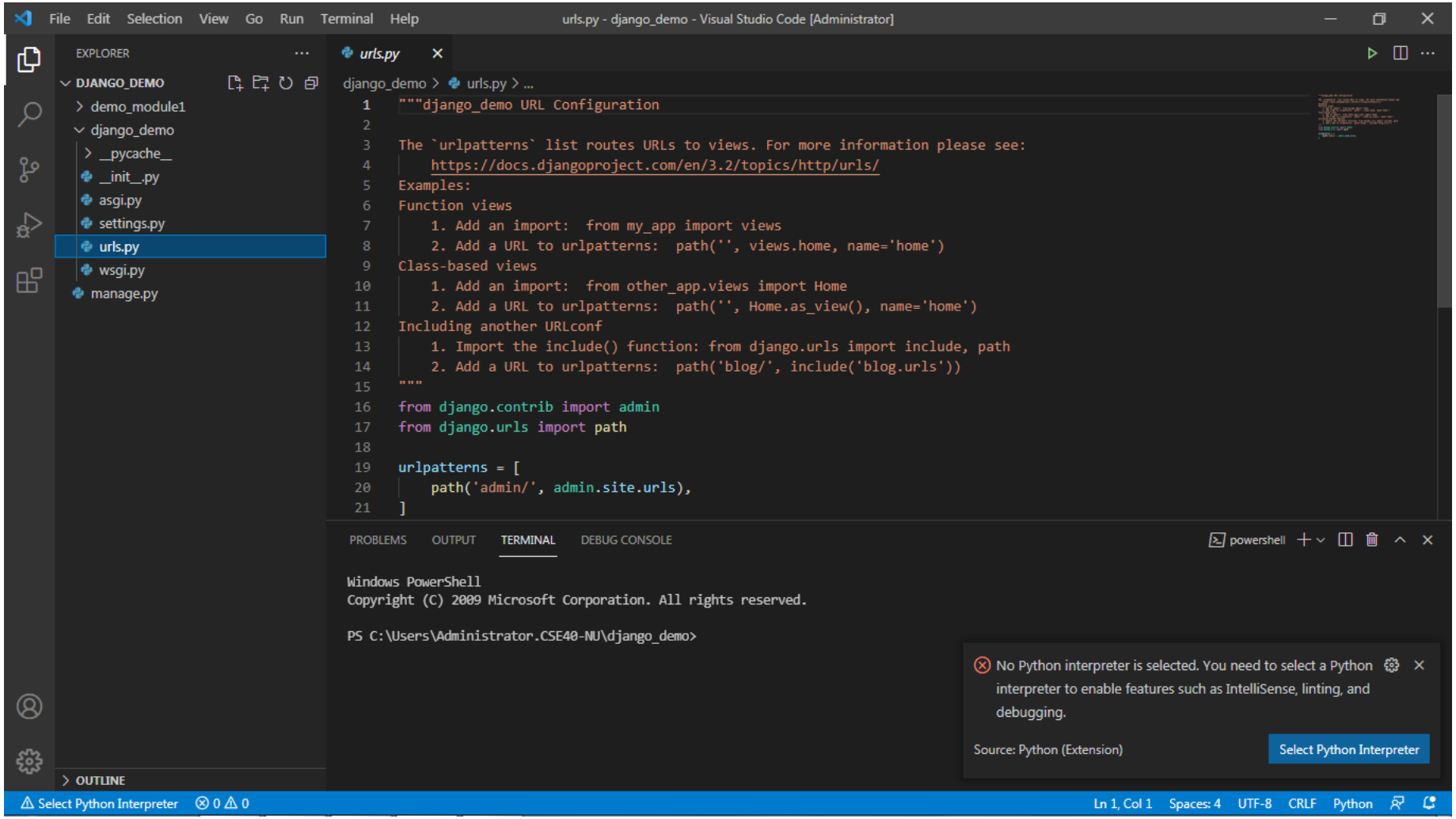
- **settings.py**

- You can see that there are some pre-installed applications. (Refer image in previous slide to this slide)

- These applications are by default there to provide all the basic functionality you will ever need for your website like Django admin app

Django Project Structure (contd…)

- **urls.py**



Django Project Structure (contd…)

- **urls.py**

- urls.py file contains the project level URL information.

- URL is universal resource locator and it provides you with the address of the resource (images, webpages, web-applications) and other resources for your website.

- The main purpose of this file is to connect the web-apps with the project. Anything you will be typing in the URL bar will be processed by this urls.py file. Then, it will correspond your request to the designated app you connected to it

Django Project Structure (contd…)

- **urls.py**

```
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```

- Here this file by default adds one URL to the admin app. The path () takes two arguments.

- 1st is the URL to be searched in the URL bar on the local server and 2nd is the file you want to run when that URL request is matched, the admin is the pre-made application and the file is URL's file of that app.

- This file is the map of your Django project.

Django Project Structure (contd…)

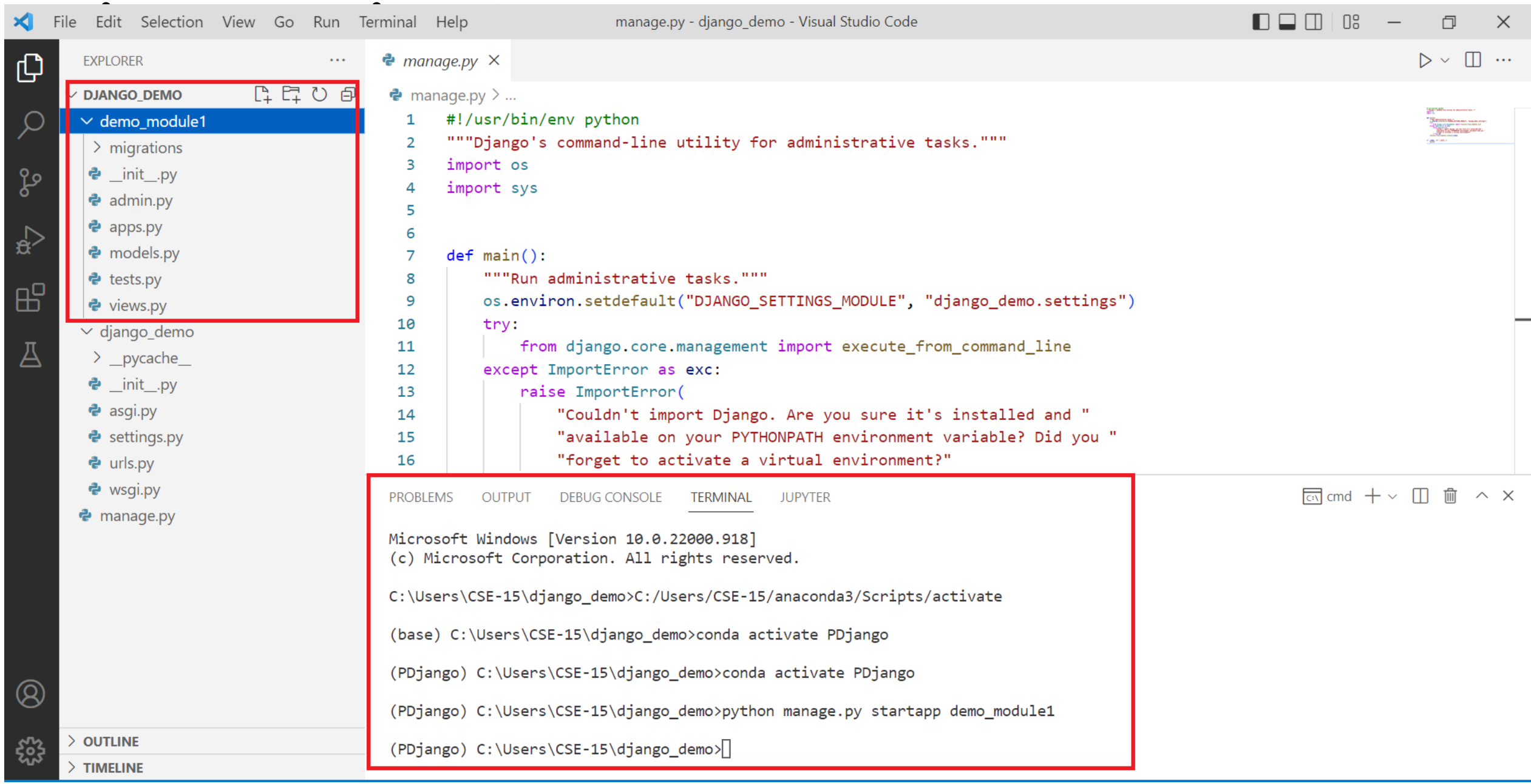
- **urls.py**

- It is used to provide the addresses of the resources (like image, website, etc) that are present there on the internet.

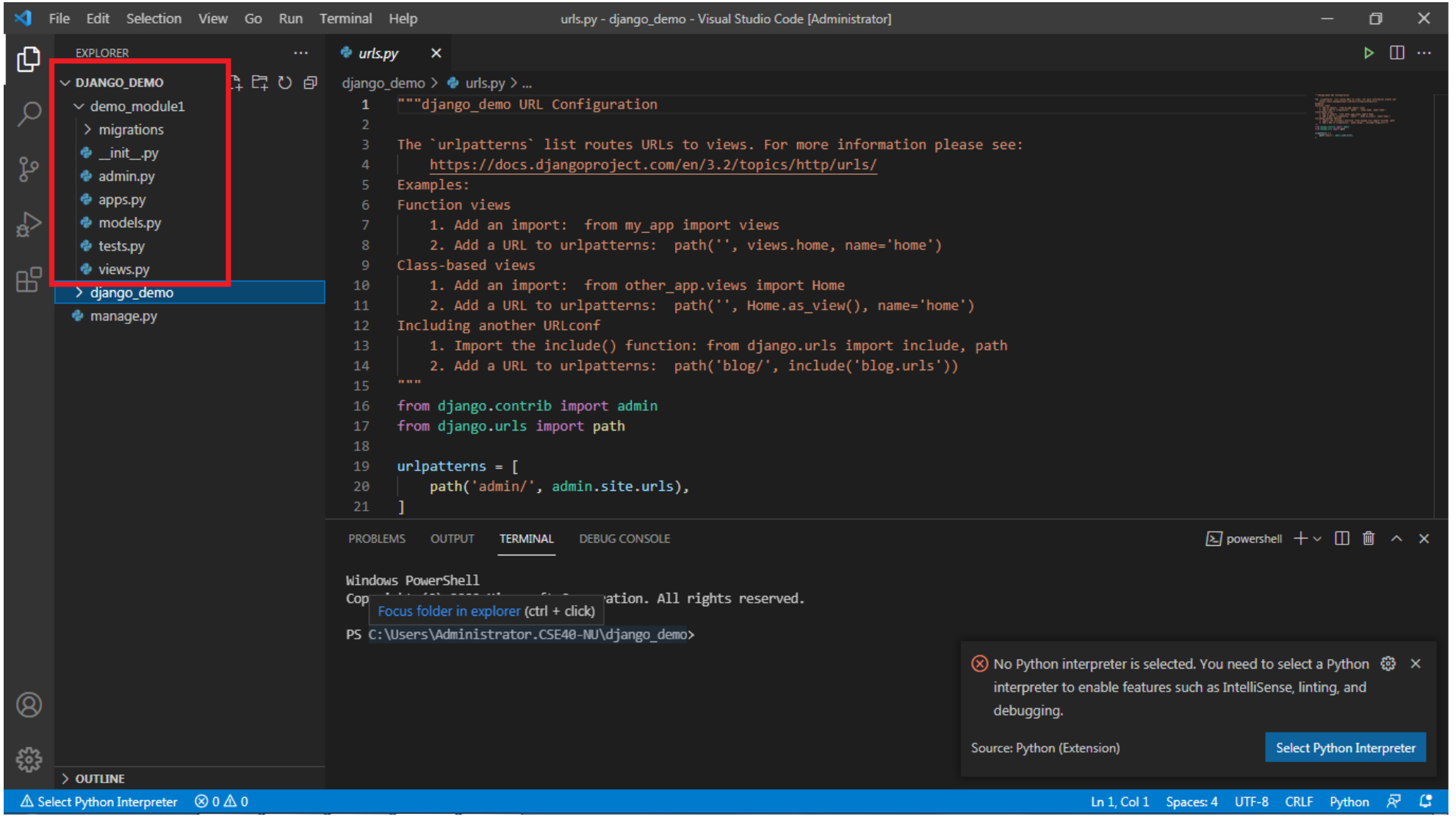
Django Project Structure (contd…)

- App creation
- Now in the terminal, run

- python manage.py startapp demo_module1 command
- This will help in creating a module/app for the Django based web application that you want to create.


File Edit Selection View Go Run Terminal Help
manage.py - django_demo - Visual Studio Code
EXPLORER
DJANGO_DEMO
demo_module1
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
django_demo
__pycache__
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
manage.py
OUTLINE
TIMELINE
manage.py
manage.py > ...
1 #!/usr/bin/env python
2 """Django's command-line utility for administrative tasks."""
3 import os
4 import sys
5
6
7 def main():
8 """Run administrative tasks."""
9 os.environ.setdefault("DJANGO_SETTINGS_MODULE", "django_demo.settings")
10 try:
11 from django.core.management import execute_from_command_line
12 except ImportError as exc:
13 raise ImportError(
14 "Couldn't import Django. Are you sure it's installed and "
15 "available on your PYTHONPATH environment variable? Did you "
16 "forget to activate a virtual environment?"
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
cmd
Microsoft Windows [Version 10.0.22000.918]
(c) Microsoft Corporation. All rights reserved.
C:\Users\CSE-15\django_demo>C:/Users/CSE-15/anaconda3/Scripts/activate
(base) C:\Users\CSE-15\django_demo>conda activate PDjango
(PDjango) C:\Users\CSE-15\django_demo>conda activate PDjango
(PDjango) C:\Users\CSE-15\django_demo>python manage.py startapp demo_module1
(PDjango) C:\Users\CSE-15\django_demo>

Django App Structure



Django App Structure (contd…)

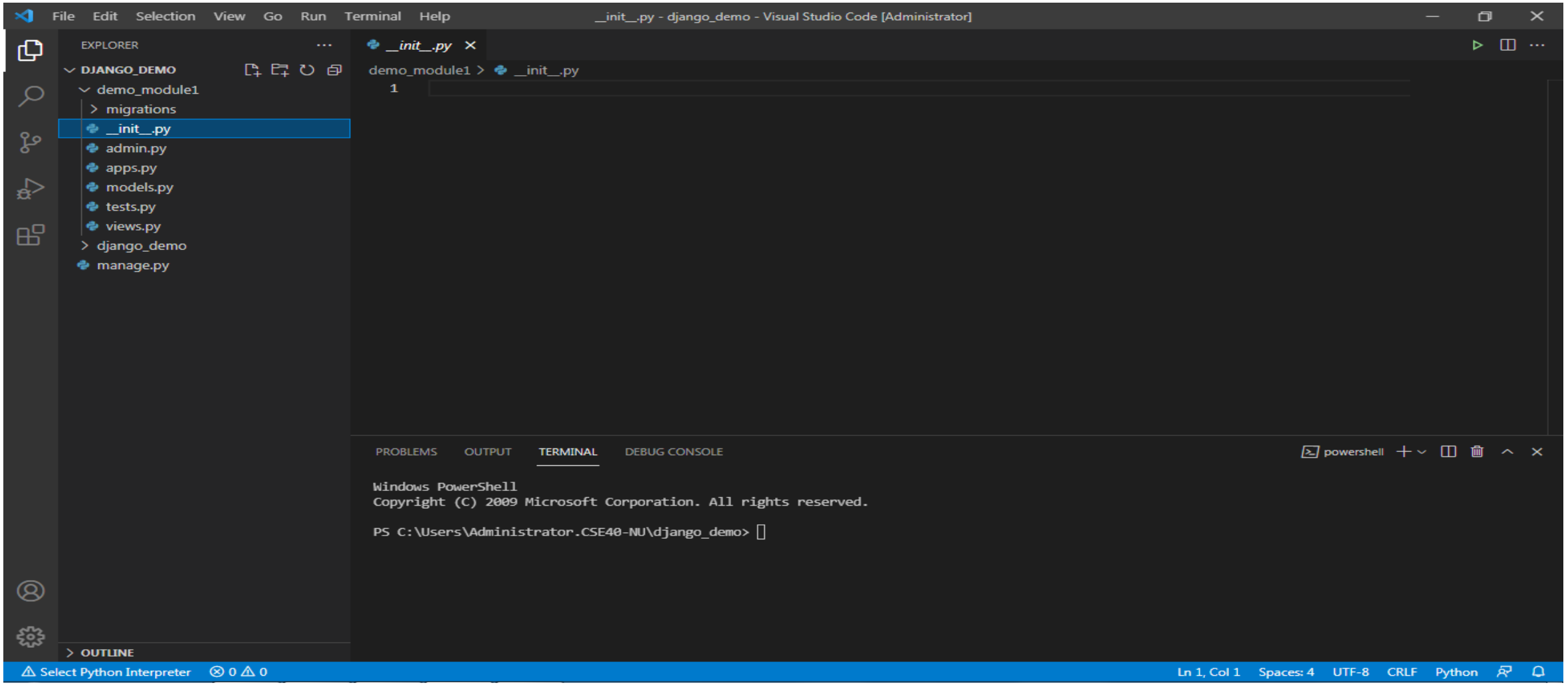
- Django uses the concept of Projects and apps for managing the codes and presents them in a readable format.

- A Django project contains one or more apps within it, which performs the work simultaneously to provide a smooth flow of the web application.

- For example, a real-world Django e-commerce site will have one app for user authentication, another app for payments, and a third app for item listing details: each will focus on a single functionality.

Django App Structure (contd…)

- _init_.py



Django App Structure (contd…)

- **_init_.py**

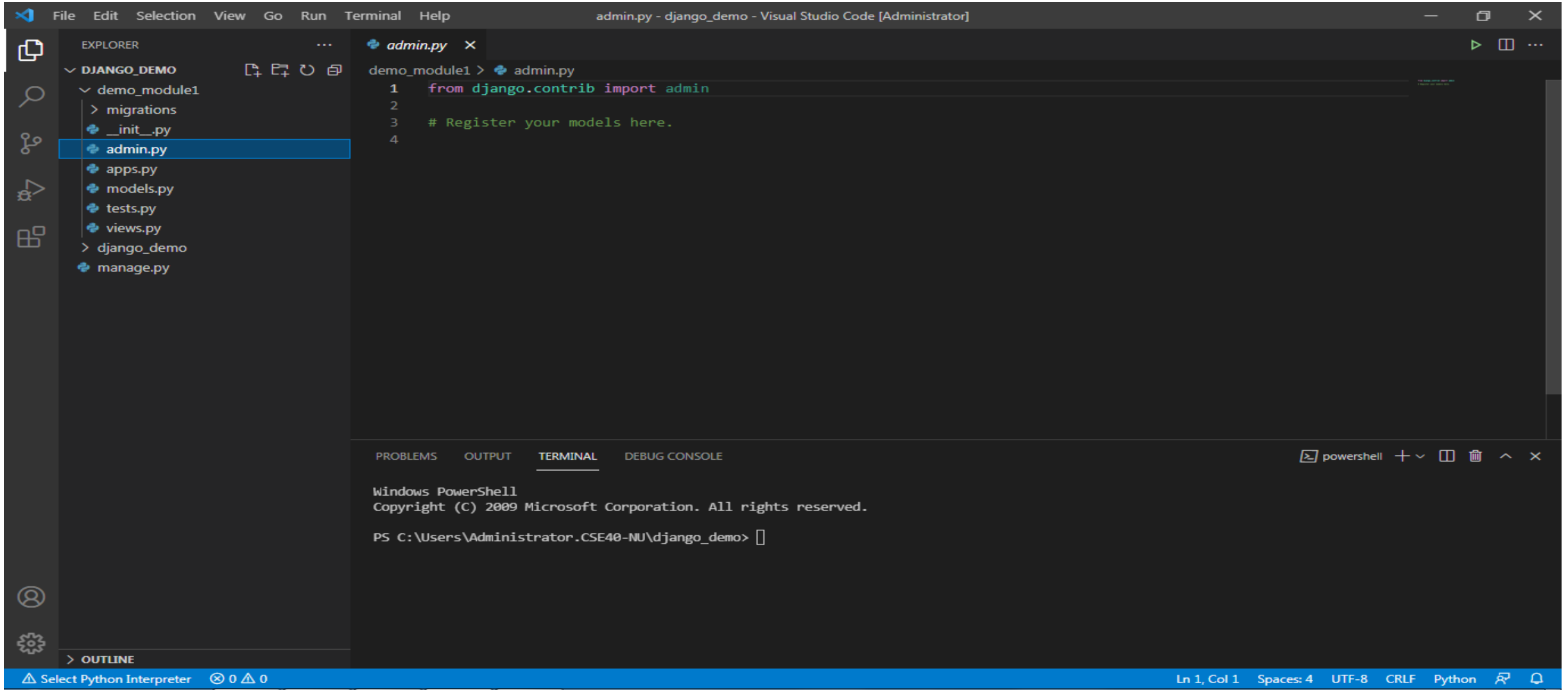
- This file has the same functionality just as in the _init_.py file in the Django project structure.

- It remains empty and is present just to indicate that the specific app directory is a package.

- **There is no need to make any changes to the file.**

Django App Structure (contd…)

- **admin.py**



Django App Structure (contd…)

- **admin.py**

- Admin.py file is used for registering the Django models into the Django administration.
- It is used to display the Django model in the Django admin panel. It performs three major tasks:

  - a. Registering models
  - b. Creating a Superuser
  - c. Logging in and using the web application

- **We will learn more about the admin panel in the next sessions.**

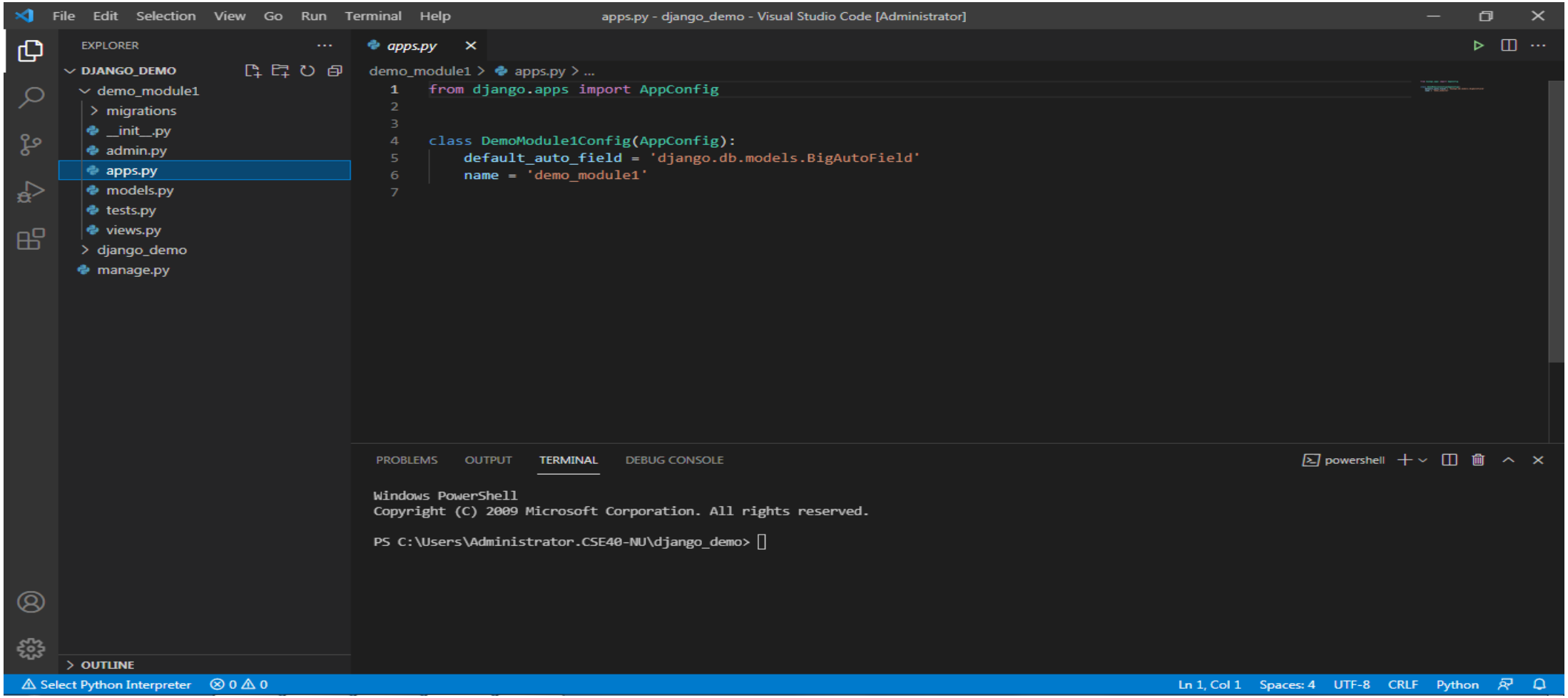
Django App Structure (contd…)

- **admin.py**

- The models that are present have a superuser/admin who can control the information that is being stored.

Django App Structure (contd…)

- apps.py



Django App Structure (contd…)

- **apps.py**
- apps.py is a file that is used to help the user include the application configuration for their app.

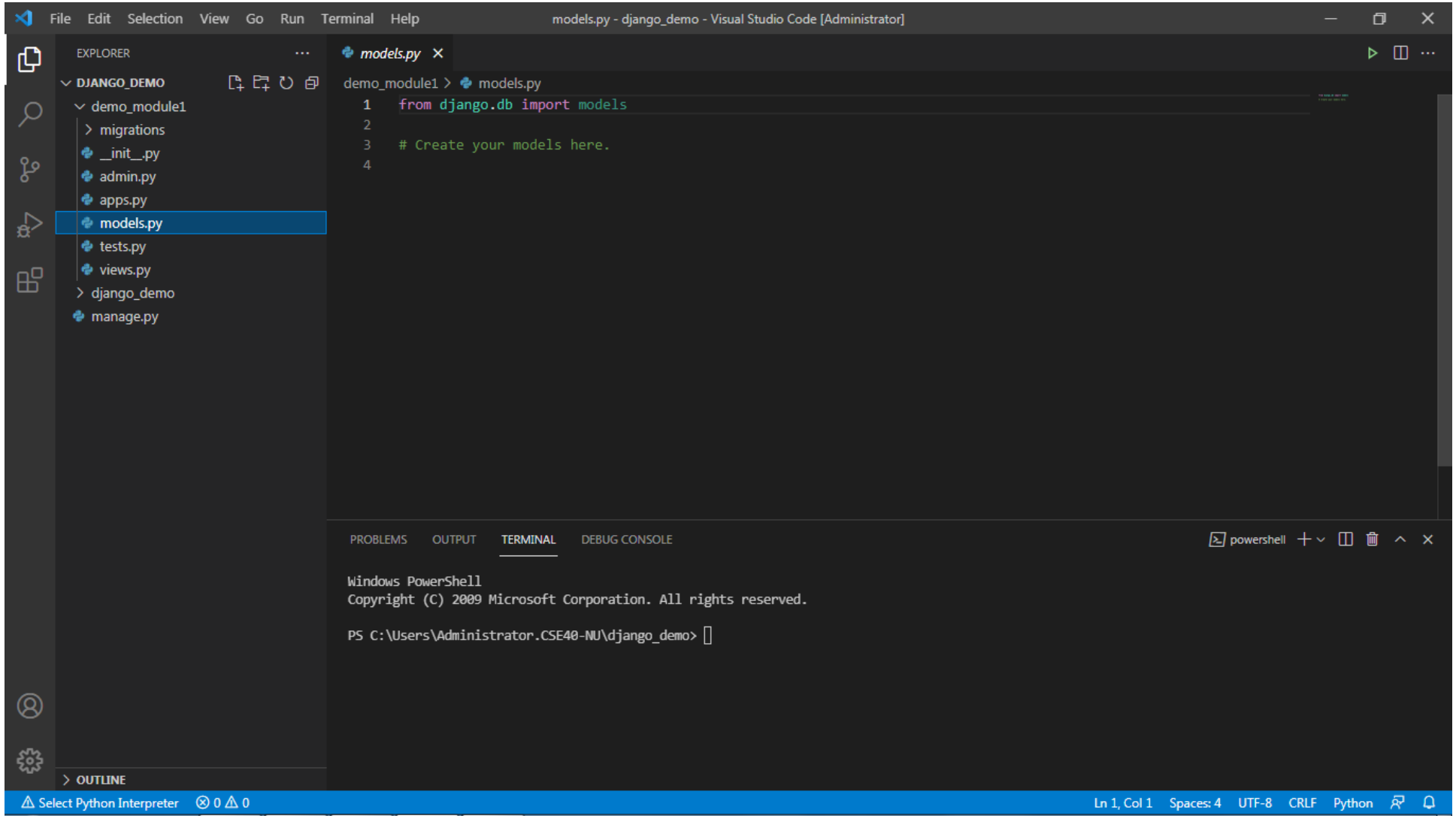
- Users can configure the attributes of their application using the apps.py file.

- However, configuring the attributes is a rare task a user ever performs, because most of the time the default configuration is sufficient enough to work with.

- The default configuration is sufficient enough in most of the cases and **hence we won’t be doing anything in the beginning with this file.**

Django App Structure (contd…)

- **models.py**



Django App Structure (contd…)

- **models.py**

- Models.py represents the models of web applications in the form of classes. It is considered the most important aspect of the App file structure.

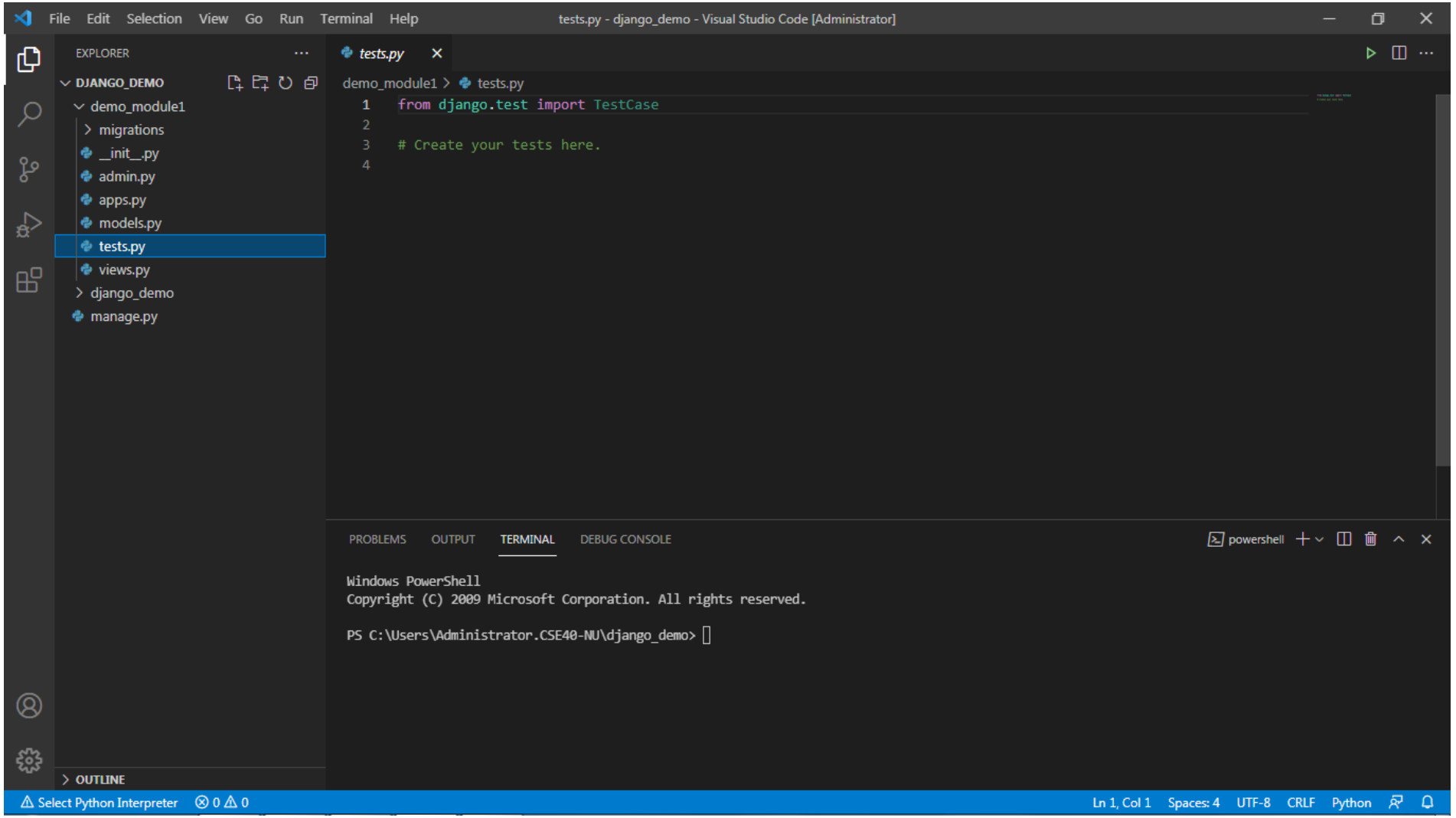
- Models define the structure of the database. It tells about the actual design, relationships between the data sets, and their attribute constraints.

- This file contains the models of our web applications (usually as classes).

- Models are basically the blueprints of the database we are using and hence contain the information regarding attributes and the fields etc of the database.

Django App Structure (contd…)

- **tests.py**



Django App Structure (contd…)

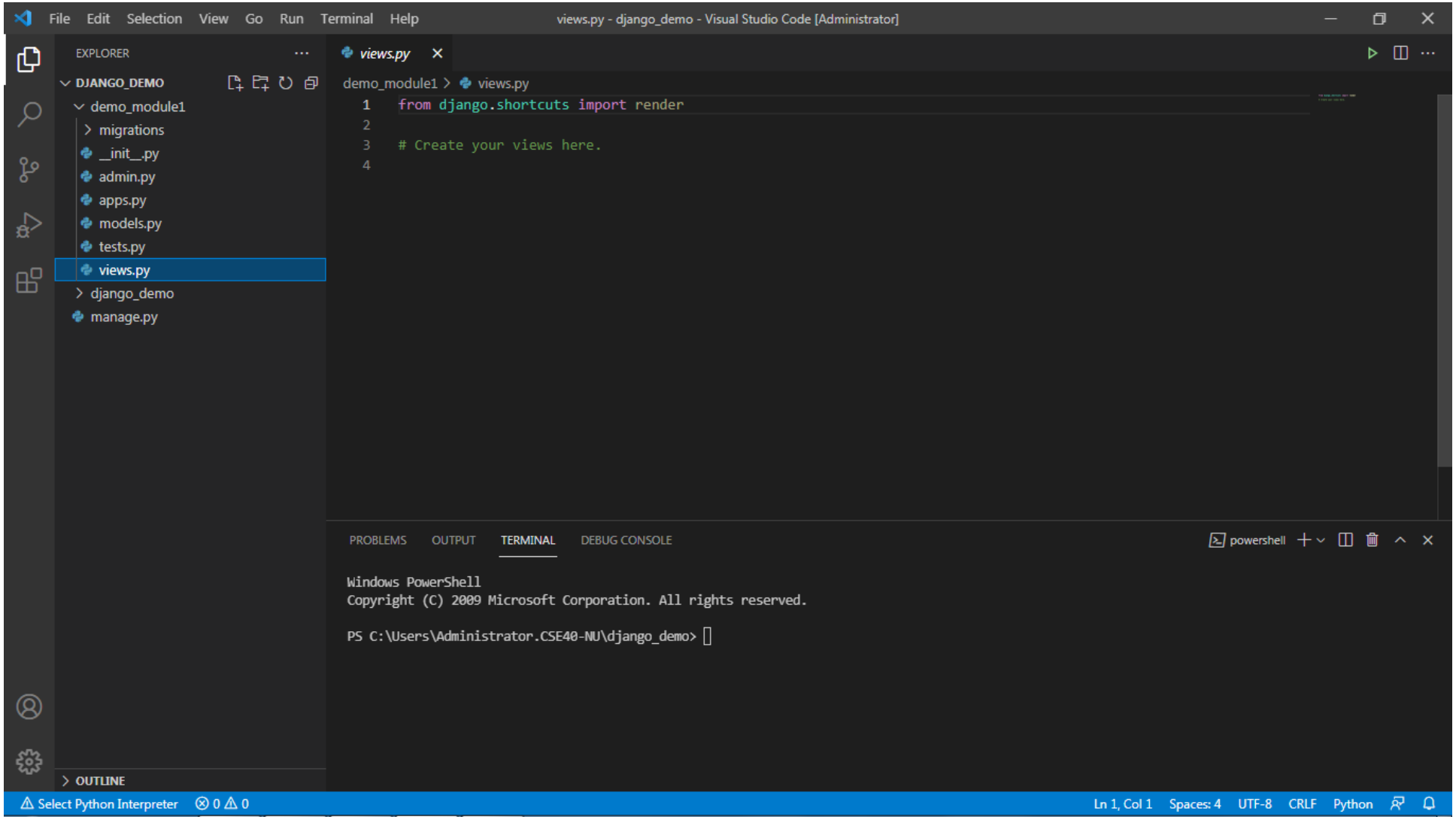
- **tests.py**

- It allows the user to write test code for their web applications. It is used to test the working of the app.

- This file contains the code that contains different test cases for the application. It is used to test the working of the application.

Django App Structure (contd…)

- **views.py**



Django App Structure (contd…)

- **views.py**

- Views are also an important part when we talk about the Django app structure.

- Views provide an interface through which a user interacts with a Django web application. It contains all the views in the form of classes.

- it contains all the Views(usually as classes). Views.py can be considered as a file **that interacts with the client**. Views are a user interface for what we see when we render a Django Web application.

Summary

- All the files we have discussed above are within every Django application you create.
- The main aim of these files is to provide you with backend support.
- However, the settings.py and urls.py are the two main files we will be working with.
- Making changes to these files will bring unique functionalities to the web application you create.

2CSDE86 Application Development Frameworks (ADF)

Lecture-4

Django Views

7th CSE

Daiwat Vyas & Ajaykumar Patel

Disclaimer

- Some of the content in the ppt is taken from various online sources and reference books and after referring it was considered for including it in the ppt slides.

Pre-requisite for this session

- **Students should have already installed Python, vscode and setup Django on their system.**

- **A file was sent for Django installation and it had all step by step procedure mentioned for installing Python, vscode and setting up Django in their system.**

Django Views

- Django views are part of the user interface — they usually render the HTML/CSS/Javascript in your Template files into what you see in your browser when you render a web page.

- (Note that if you've used other frameworks based on the MVC (Model-View-Controller), do not get confused between Django views and views in the MVC paradigm.

- Django views roughly correspond to controllers in MVC, and Django templates to views in MVC.)

Django Views

- Django Views are one of the vital participants of M**V**T Structure of Django.

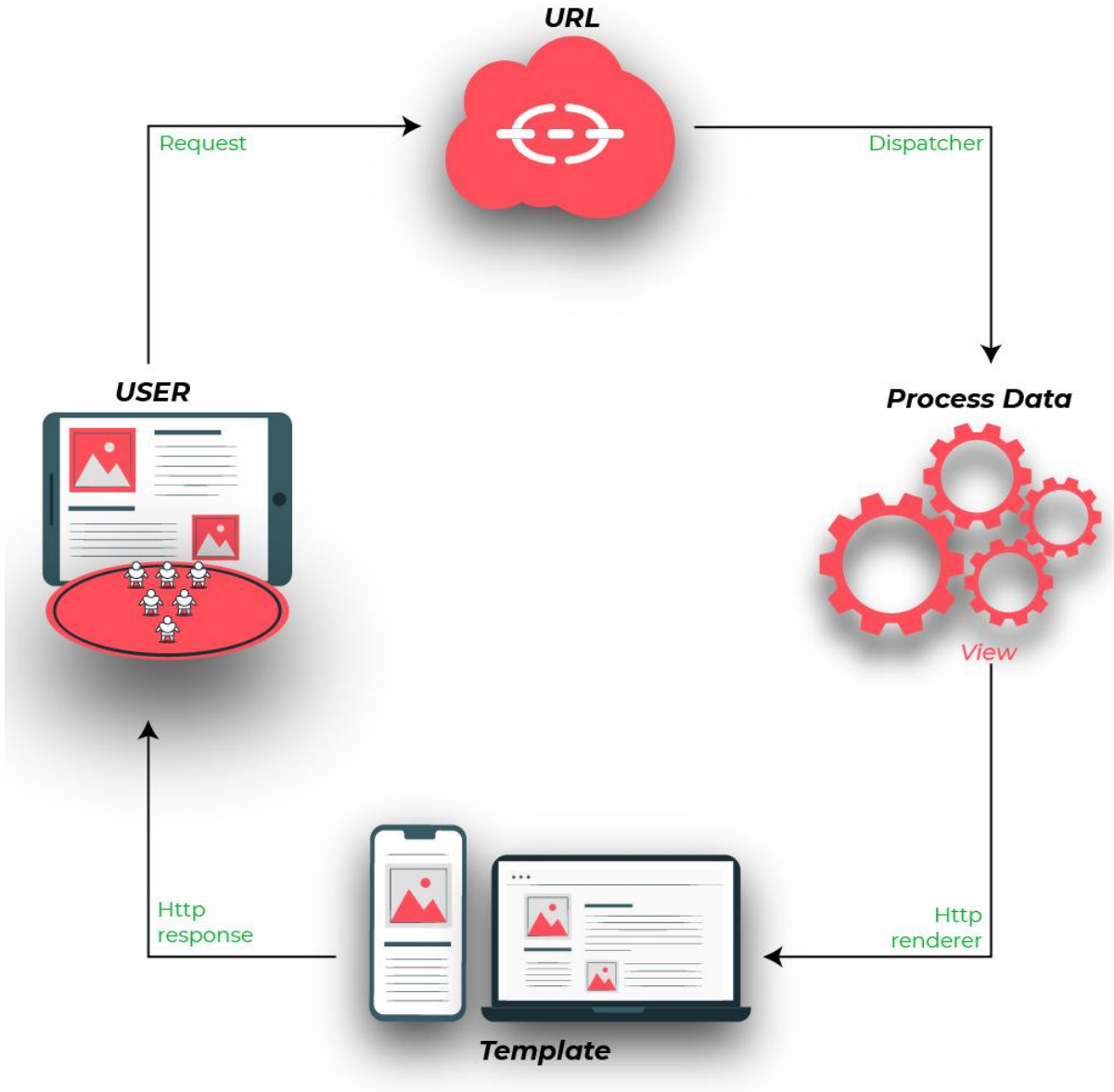
- A View is the user interface — what you see in your browser when you render a website.

- It is represented by HTML/CSS/ JavaScript files.

- As per Django Documentation, A view function is a Python function that takes a Web request and returns a Web response.

- This **response** can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, anything that a web browser displays.

Django Views (contd…)



Django Views (contd…)

- **Example:**
- django-admin startproject DjangoViewsDemo (For creating a new Django Project)

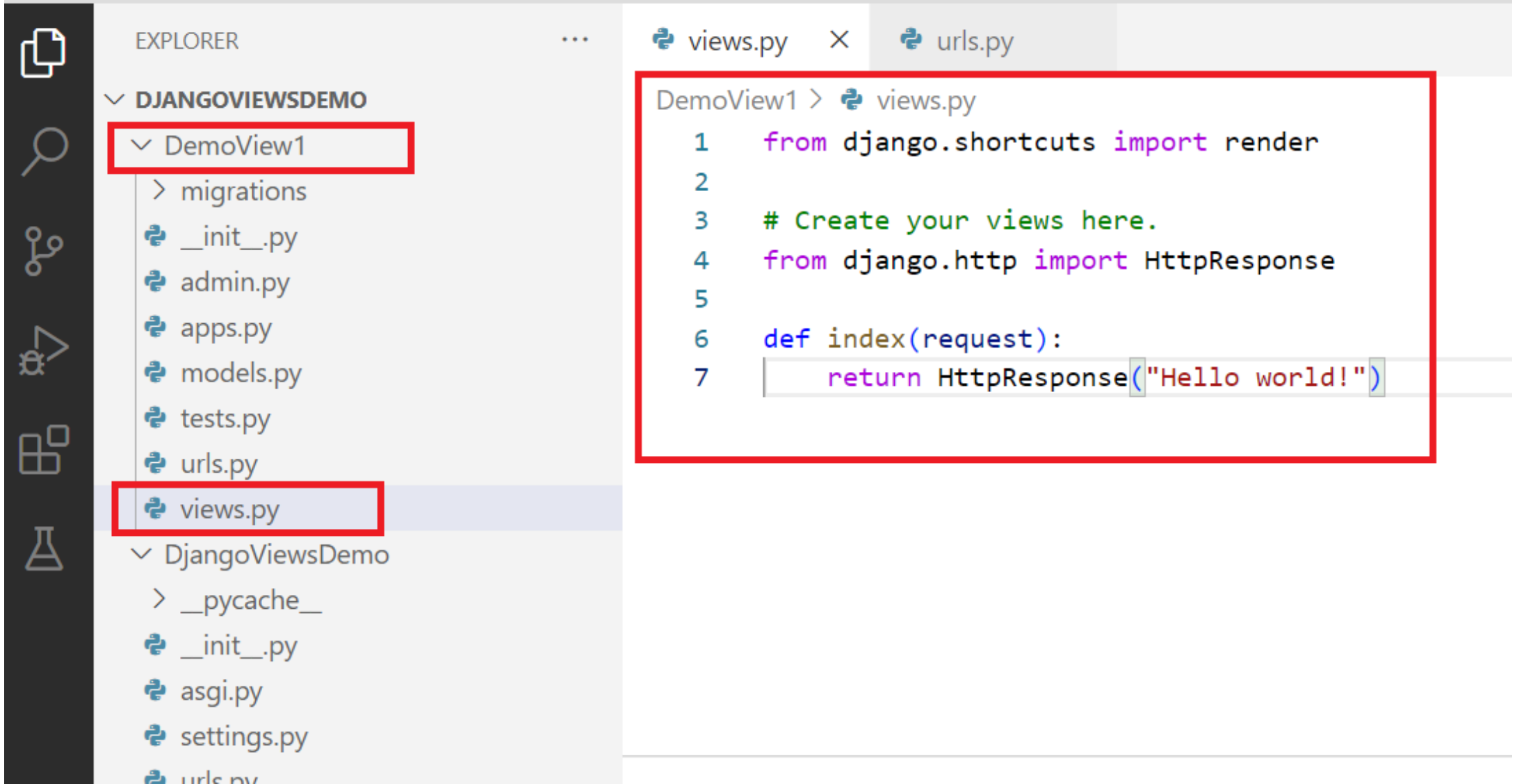
- python manage.py startapp DemoView1

- Ensure that you have created a Project in Django and an App (module) in that Project. *(Recall last lab session and lecture session)*

- Go to views.py file in the Django App (Module) folder. *(We will make changes in it)*

Django Views (contd…)

- **Example:** *views.py*



Django Views (contd…)

- **Example:** *urls.py*

- This is a simple example on how to send a response back to the browser.

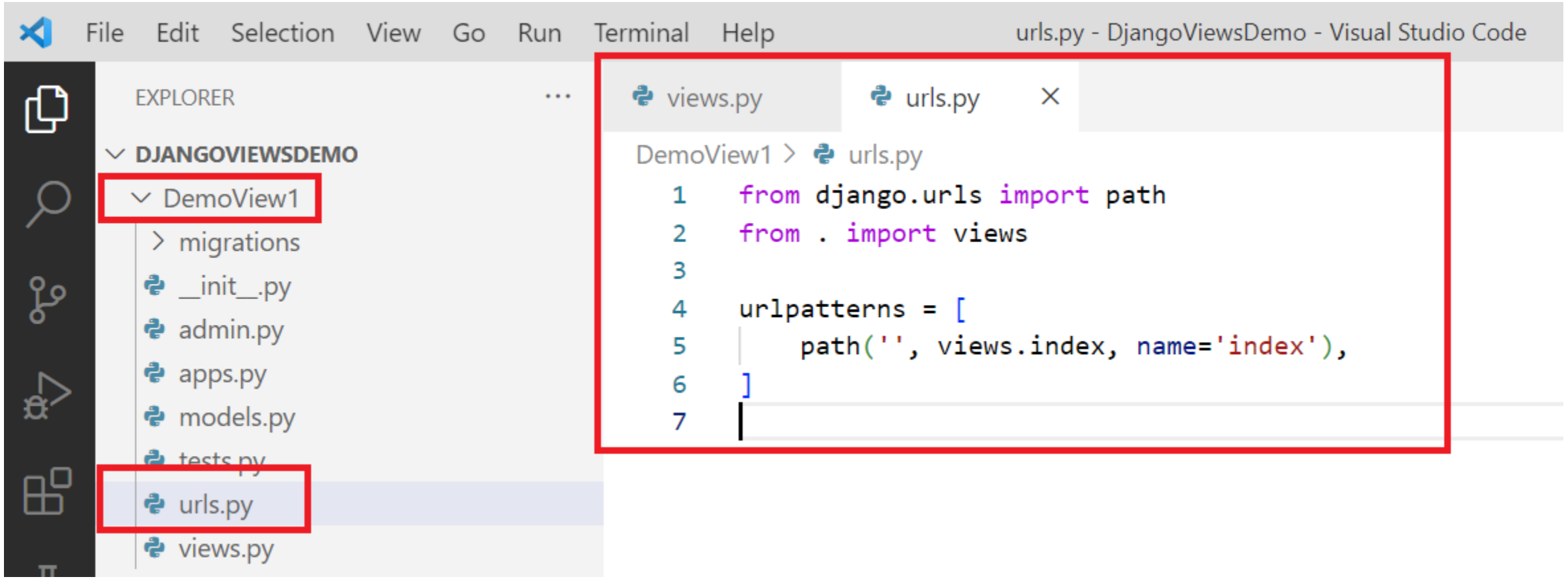
- But how can we execute the view? Well, we must call the view via a URL.

- Create a file named urls.py in the same folder as the views.py i.e in App folder

- Update the urls.py file

Django Views (contd…)

- **Example:** *urls.py*



Django Views (contd…)

- **Example:** *urls.py*

- *path(route,view, kwargs, name)*

- This function returns an element that needs to be included in urlpatterns. That is, path acts as a link between an element (for eg Views) and its URL.

- **route:** This is the URL for a particular view. For eg: ‘<name>/’ is a route.

- So when we request this page from the URL, the server will return the view linked to it.

Django Views (contd…)

- **Example:** *urls.py*

- *path(route,view, kwargs, name)*

- ***view:*** Here we need to write, the view name that we need to link. Or use the function “include” to add another urls.py file. (Like in the project/urls.py file)

Django Views (contd…)

- **Example:** *urls.py*

- *path(route,view, kwargs, name)*

- *****kwargs and *args:*** args in function definition in python is a syntax used for the variable input argument list. It is used with a single asterisk.
- That is if for example, we need to input any variable number of arguments for a function, then we use *args. Eg:

```
def myFun(*args):
    for arg in args:
        print(arg)


myFun("Hello","There","Hi","There")
```

Django Views (contd…)

- **Example:** *urls.py*

- *path(route,view, kwargs, name)*

- ****kwargs and *args:***
- So in the example we can give any number of arguments and *args will take up all of them. We can think that all the arguments are being saved by arg as some list, thereby using the line for arg in args, we are taking each element from the list.

- the * is splitting up the list into elements thus *args gives you all the elements separately and args will give the elements as a list.

Django Views (contd…)

- **Example:** *urls.py*

- *path(route,view, kwargs, name)*

- ****kwargs and *args:*** kwargs in function definitions in Python are used for key-worded, variable arguments list.

- It is used with double asterisk. Eg:

```
def myFun(**kwargs):
    for item,price in kwargs.items():
        print(f"{item}={price}")

myFun(Book=100,Pen=10,Watch=4000)
```

Django Views (contd…)

- **Example:** *urls.py*

- *path(route,view, kwargs, name)*

- *****kwargs and *args:*** Eg:

- As you can see in the above example, we are able to pass complete values. The variable names and the values held by those variables with the use of the assignment operator.

- We can think of the arguments being saved as a Python dictionary by the kwargs thus by using the line for the item, price in kwargs.items(), we are taking up the item(Book, Pen, etc) and their corresponding price(100,10, etc).

- The ** splits the dictionary into its elements. Thus **kwargs gives you all the key-worded elements separately while kwargs gives you the key-worded elements as a dictionary.

Django Views (contd…)

- **Example:** *urls.py*

- *path(route,view, kwargs, name)*

- ***name:*** *name is used to specify the name of the particular view that the URL is linking.*

- ***include(module,namespace=None)***
- This function takes another URL conf file that should be included by using this include function. That is to form a link with another urls.py file you should use include function. The namespaces can also be written inside but we don’t need to do that for now.

Django Views (contd…)

- **Example:** *urls.py*

- The urls.py file we just created is specific for the *DemoView1* application of the project *DjangoViewsDemo*.

- There will be a need to do some routing in the root directory i.e project root folder. as well. Just follow the instructions below:

- There is a file called urls.py in main project folder, open that file and add the include module in the import statement, and also add a path() function in the urlpatterns[] list, with arguments that will route users that comes in via 127.0.0.1:8000/DemoView1/

Django Views (contd…)

- **Example: Create a new app DemoView2 in the project DjangoViewsDemo**

- *python manage.py startapp DemoView2*

Django Views (contd…)

- **Example:** *views.py in DemoView2 app*

DemoView2 > views.py

```
2
3   # Create your views here.
4   # import Http Response from django
5   from django.http import HttpResponse
6   # get datetime
7   import datetime
8
9   # create a function
10  def demo_view(request):
11      # fetch date and time
12      now = datetime.datetime.now()
13      # convert to string
14      html = "Time is {}".format(now)
15      # return response
16      return HttpResponse(html)
```

Django Views (contd…)

- **Example:** *views.py in DemoView2 app*

- First, we import the class HttpResponse from the django.http module, along with Python's datetime library.

- Next, we define a function called demo_view. This is the view function. Each view function takes an HttpRequest object as its first parameter, which is typically named request.

- The view returns an HttpResponse object that contains the generated response. Each view function is responsible for returning an HttpResponse object.

- In next session we will discuss in details regarding, Django's HTTP Request-Response cycle.

Django Views (contd…)

- **Example:** (*Line by line explanation*)

- **Django Request and Response cycle – HttpRequest and HttpResponse Objects will be discussed later on.**

- Let us now get the demo_view working:

- Create a urls.py file in the Django App (Module).

Django Views (contd…)

- **Example: Create** *urls.py in DemoView2 app*

DemoView2 > urls.py

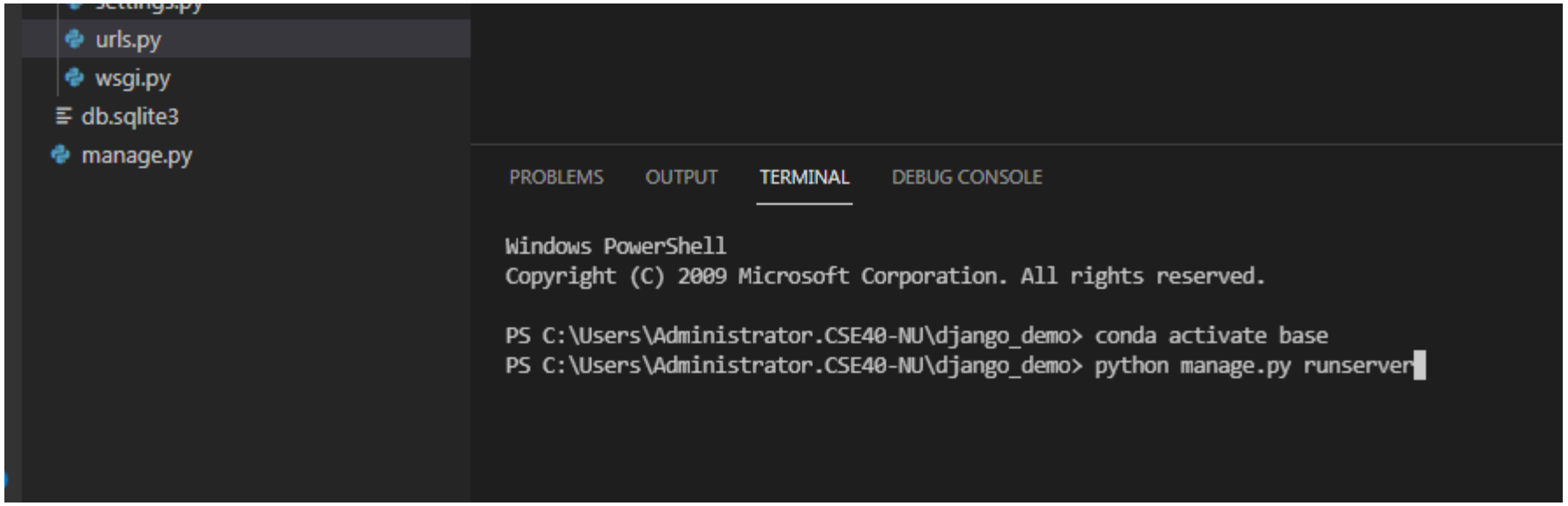
```
1   from django.urls import path
2
3   # importing views from views..py
4   from .views import demo_view
5
6   urlpatterns = [
7       path('', demo_view),
8   ]
```

Django Views (contd…)

- **Example:**

- Now, go to in-built terminal and run the server using following command:

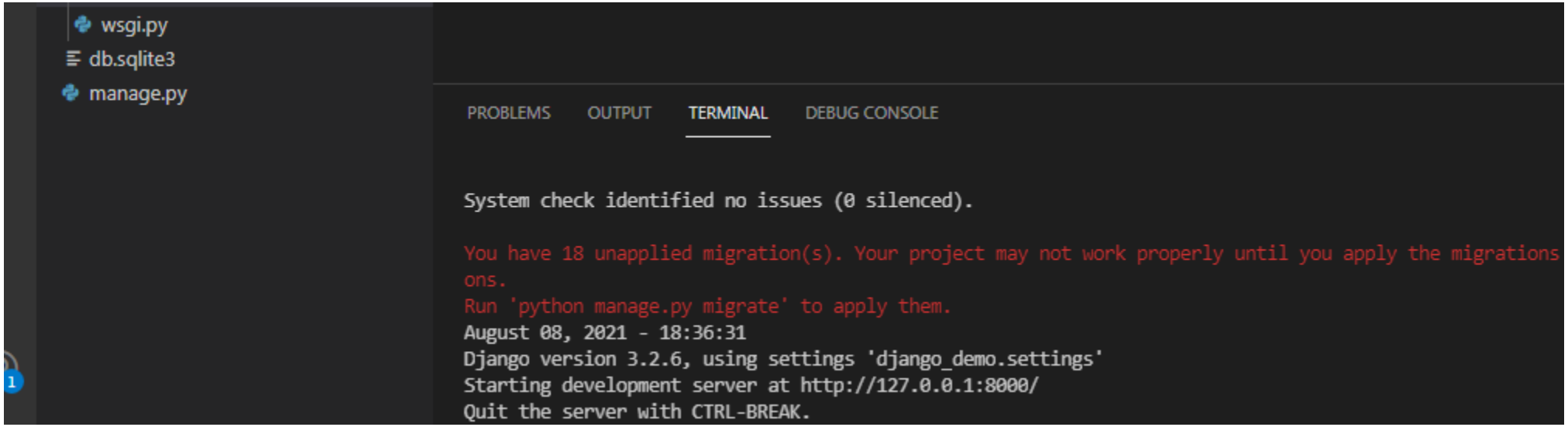
- **python manage.py runserver**



Django Views (contd…)

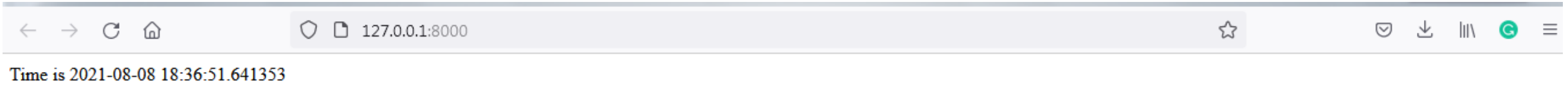
- **Example:**

- Press ctrl and click on the http://127.0.0.1:8000 in terminal window



Django Views (contd…)

- **Example:** *Output*



Django Views (contd…)

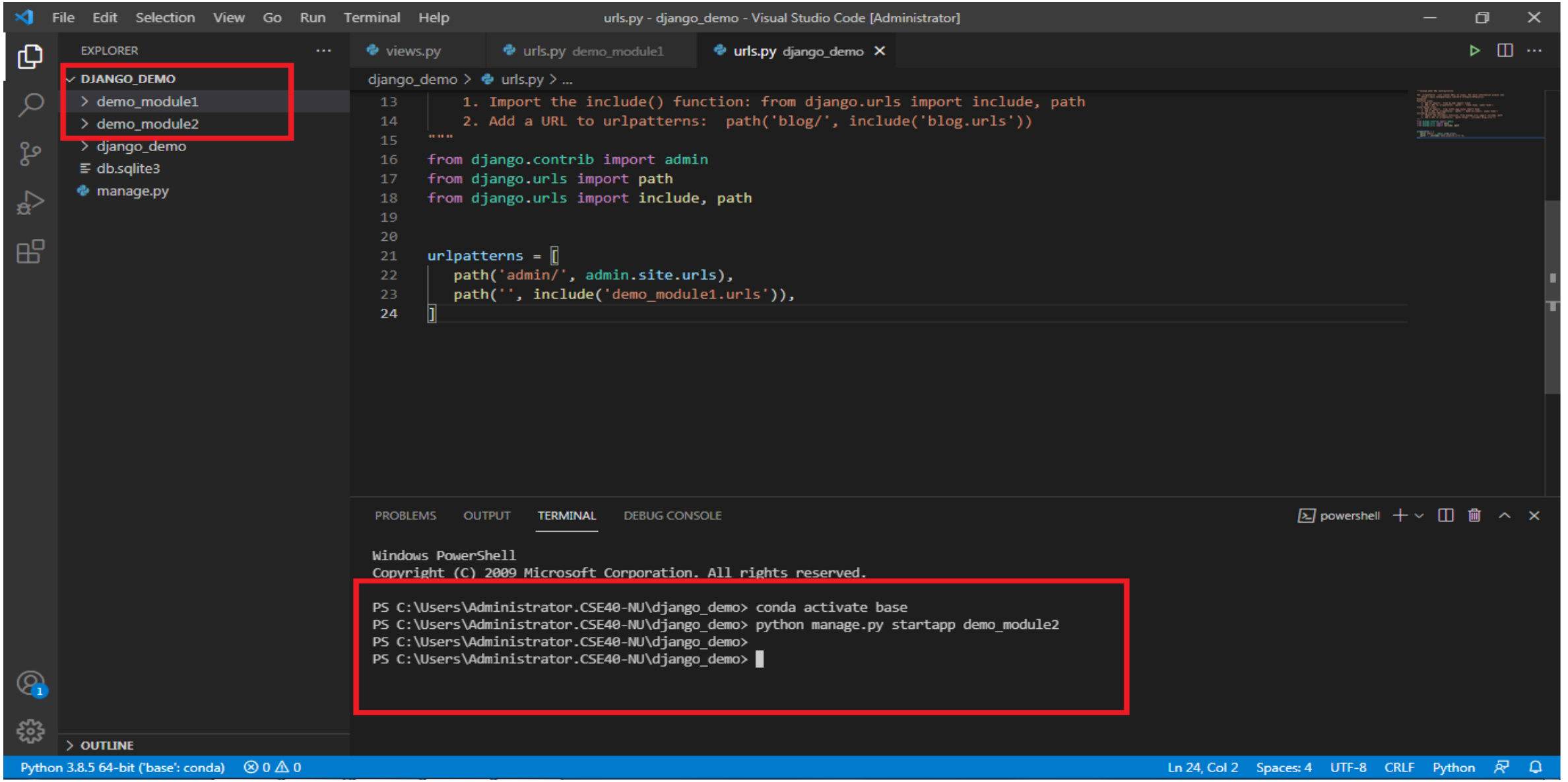
- Let us take one more example and understand the views in detail.

- Go to terminal and run following command:

- *python manage.py startapp demo_module2*

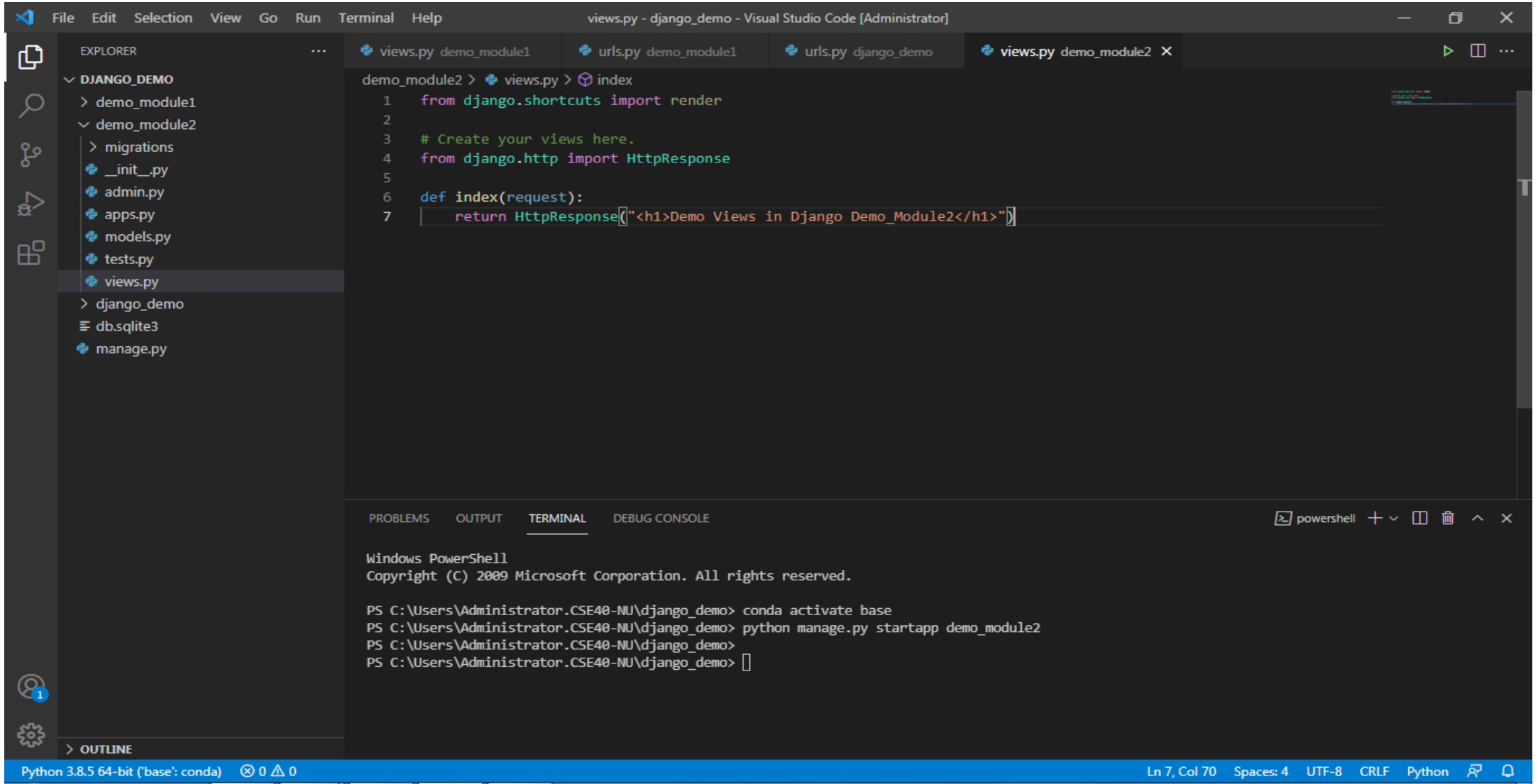
- **This will help in creating a module/app for the Django based web application that you want to create**

Django Views (contd…)



Django Views (contd…) views example in detail

- **Example 2:** *views.py django_module2*



Django Views (contd…) views example in detail

- **Example 2:** *views.py django_module2*

```
from django.shortcuts import render

# Create your views here.
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h1>Demo Views in Django Demo_Module2</h1>")
```

Django Views (contd…) views example in detail

- **Example 2:** *views.py django_module2*

```
from django.shortcuts import render

# Create your views here.
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h1>Demo Views in Django Demo_Module2</h1>")
```

This is your view function. It's an example of a function-based view. It takes a request from your web browser and returns a response. In this simple case, it's just a line of text formatted as an HTML heading. (Types of views will be discussed in next slides)

Django Views (contd…) views example in detail

- **Example 2:** *views.py django_module2*

- ***Configuring urls***

- If you started the development server now, you will notice it would still display the welcome page.

- For Django to use your new view, you need to tell Django the index view is the view you want to display when someone navigates to the site root (home page). ***We do this by configuring our URLs.***

Django Views (contd…) views example in detail

- **Example 2:** *views.py django_module2*

- ***Configuring urls***

- In Django, the *path()* function is used to configure URLs.

- In its basic form, the *path()* function has a very simple syntax:

```
path(route, view)
```

Django Views (contd…) views example in detail

- **Example 2:** *views.py django_module2*

- ***Configuring urls***

- A practical example of the basic path() function would be:

- path('mypage/', views.myview)

- In this example, a request to *http://example.com/mypage* would route to the myview function in the application's views.py file.

- The path() function statements live in a special file called urls.py.

Django Views (contd…) views example in detail

- **Example 2:** *views.py django_module2*

- ***Configuring urls***

- When startproject created our website i.e django_demo project, it created a urls.py file in our site folder (\django_demo\urls.py).

- This is the correct place for site-wide navigation, but is rarely a good place to put URLs relating to individual applications.

- Not only is having all our URLs in the one file more complex and less portable, but it can lead to strange behavior if two applications use a view with the same name.

Django Views (contd…) views example in detail

- **Example 2:** *views.py django_module2*

- ***Configuring urls***

- To solve this problem, we create a new urls.py file for each module/application.

- If you are wondering why startapp didn’t create the file for us, not all apps have public views accessible via URL.

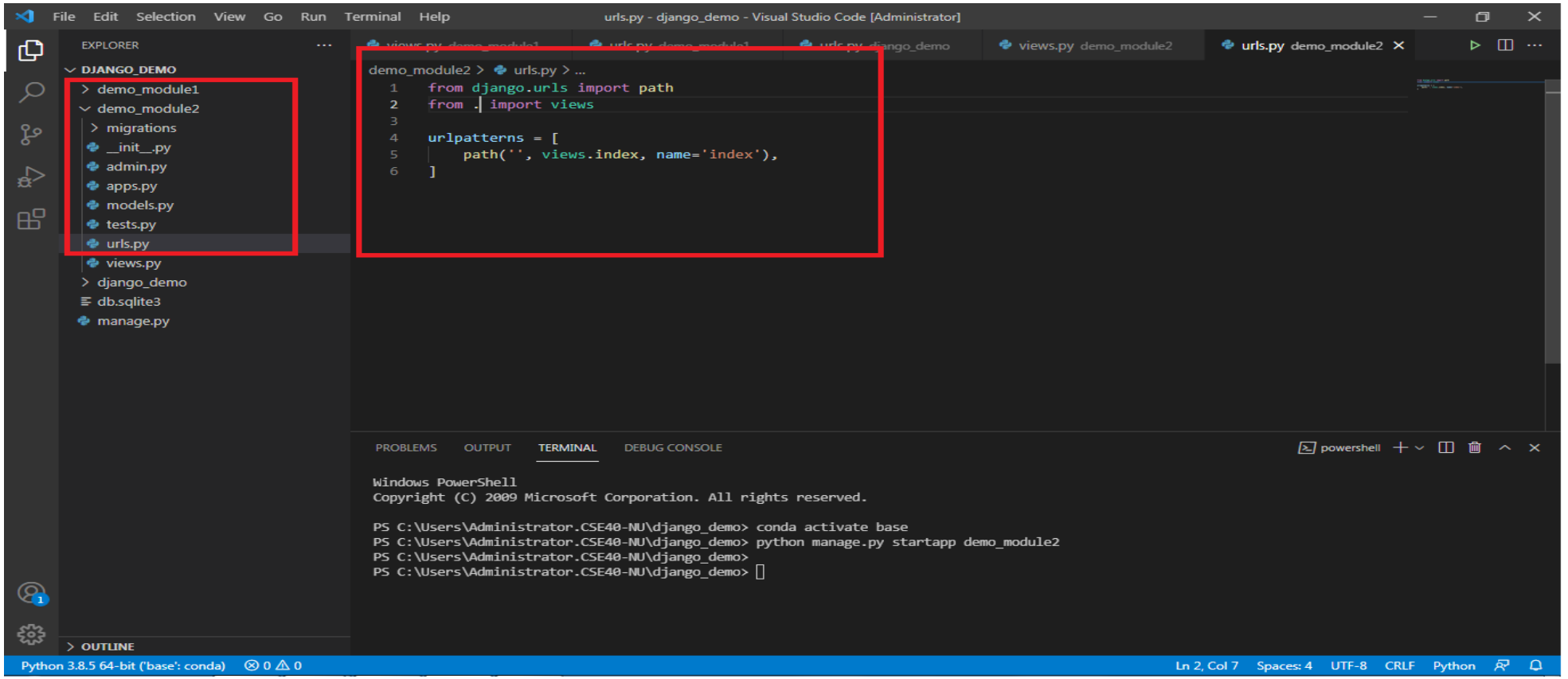
- For example, a utility program that performs background tasks would not need a urls.py file.

- For this reason, Django lets you decide whether your app needs its own urls.py file.

Django Views (contd…) views example in detail

- **Example 2:** *urls.py django_module2*

- *Create a urls.py file in the django_module2 app/module*



Django Views (contd…) views example in detail

- **Example 2:** *urls.py django_module2*

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Django Views (contd…) views example in detail

- **Example 2:** *urls.py django_module2 (Line by Line explanation of code)*

- **Line 1** imports the path() function. This import is necessary for the URL dispatcher to work and is common to all urls.py files.

- **Line 2** imports the local views.py file. The dot operator (“.”) in this case is shorthand for the current package, so this is saying “import all views from the current package (events)”.

- **Line 4** lists the URL patterns registered for this app. For readability, the list is broken into multiple lines, with one URL pattern per line.

Django Views (contd…) views example in detail

- **Example 2:** *urls.py django_module2 (Line by Line explanation of code)*

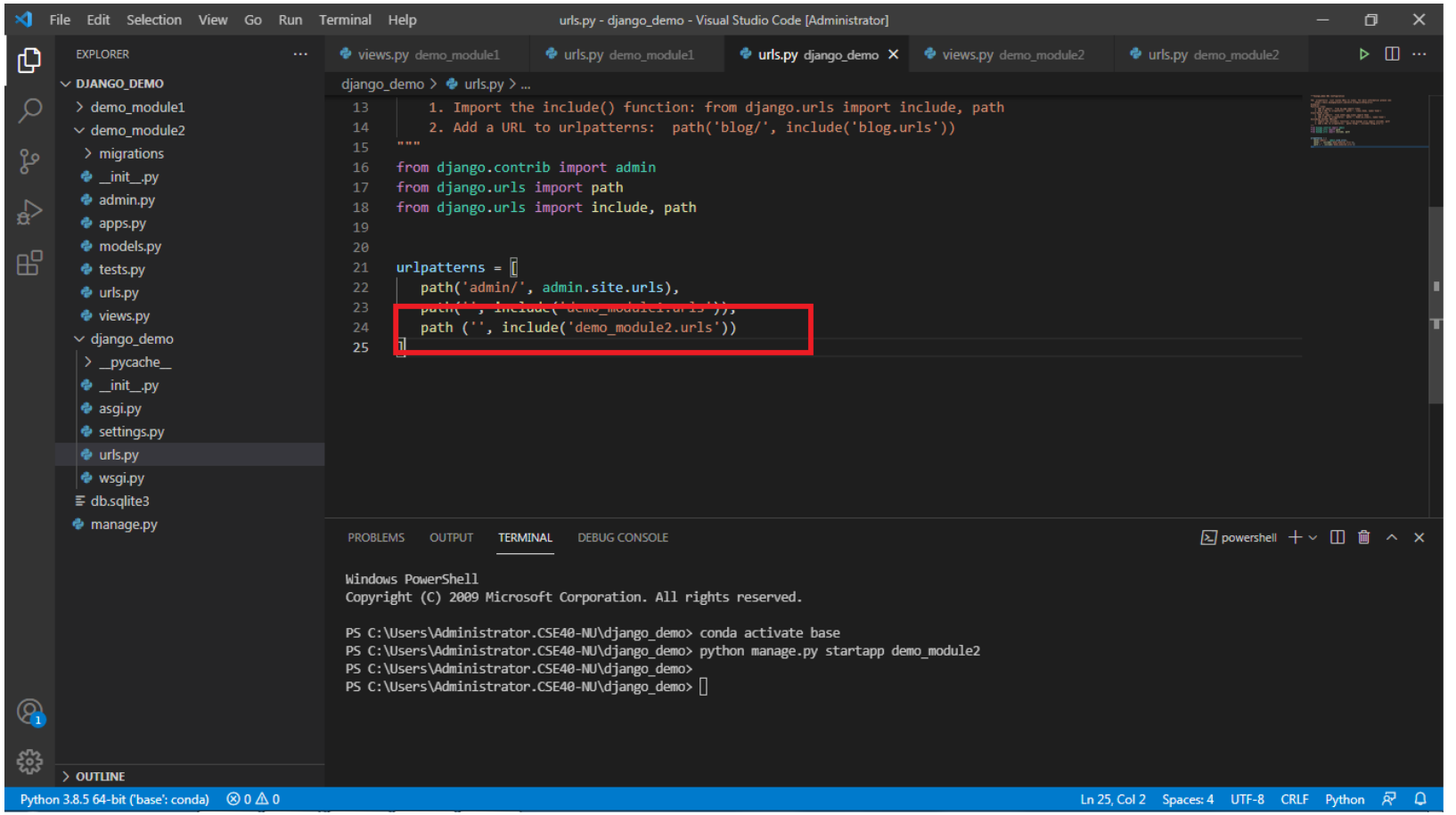
- **Line 5** is the actual URL dispatcher: '' matches an empty string. It will also match the "/" as Django automatically removes the slash.

- In other words, this matches both http://example.com and http://example.com/. views.index points to our index view.
- i.e., the dot operator is pointing to the index view inside the views.py file that we imported in line 2. name='index'.
- While it's optional, you should always name your URLs. We name URLs so they can be referred to in code (reverse lookup). URL reversing is common in both templates and views, so you will see several examples as we see more examples.

Django Views (contd…) views example in detail

- **Example 2:** *urls.py django_demo*

- Go to urls.py file in the django_demo project



Django Views (contd…) views example in detail

- **Example 2:** *urls.py django_demo*

```
from django.contrib import admin
from django.urls import path
from django.urls import include, path


urlpatterns = [
   path('admin/', admin.site.urls),
   path('', include('demo_module1.urls')),
   path ('', include('demo_module2.urls'))
]
```

Django Views (contd…) views example in detail

- **Example 2:** *urls.py django_demo (Line by Line explanation)*
- **Line 18** We have added the include() function to our imports.

- **Line 24** We have added a new URL dispatcher. In this file, the dispatcher is including the urls.py file from the demo_module2 app.

- The empty string (' ') will match everything after the domain name.

- This pattern must be the last entry in the urlpatterns list, otherwise Django's shortcut logic will switch to the events app before trying to match any of the other site URLs.

Django Views (contd…) views example in detail

- **Example 2:** *urls.py django_demo (Line by Line explanation)*
- **Line 18** We have added the include() function to our imports.

- **Line 24** We have added a new URL dispatcher. In this file, the dispatcher is including the urls.py file from the demo_module2 app.

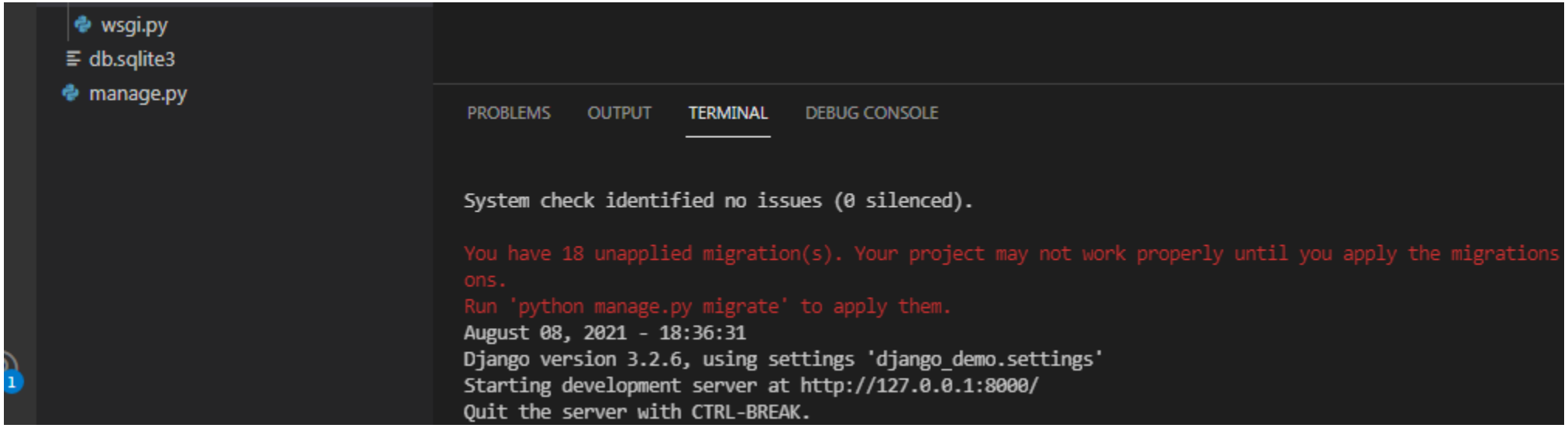
- The empty string (' ') will match everything after the domain name.

- This pattern must be the last entry in the urlpatterns list, otherwise Django's shortcut logic will switch to the events app before trying to match any of the other site URLs.

Django Views (contd…)

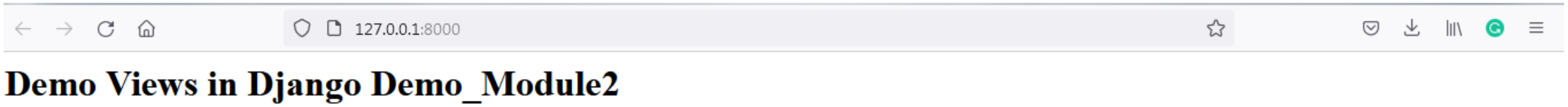
- **Example 2:**

- Press ctrl and click on the http://127.0.0.1:8000 in terminal window



Django Views (contd…)

- **Example 2: output**



Django Views (contd…)

- **Summary: How it worked?**

- **Think and answer…**

Django Views (contd…)

- **Summary: How it worked?**

- Our browser sent a message to the Django development server requesting it return content located at the root URL (http://127.0.0.1:8000/).

- Django then looked for a URL pattern matching the request, by first searching the site level urls.py, and then each of the apps for a urls.py file containing a pattern that matches.

- Django checks the first pattern (admin/) in our site level urls.py which doesn’t match and moves on to the second line in which the empty string (root URL) matches.

Django Views (contd…)

- **Summary: How it worked?**
- The matching pattern includes the urls.py from the events app. Basically, this include says “go look in the events app for a pattern that matches”.

- Once in the app-level urls.py, the empty string matches again. But this time, the request is sent to the index view.

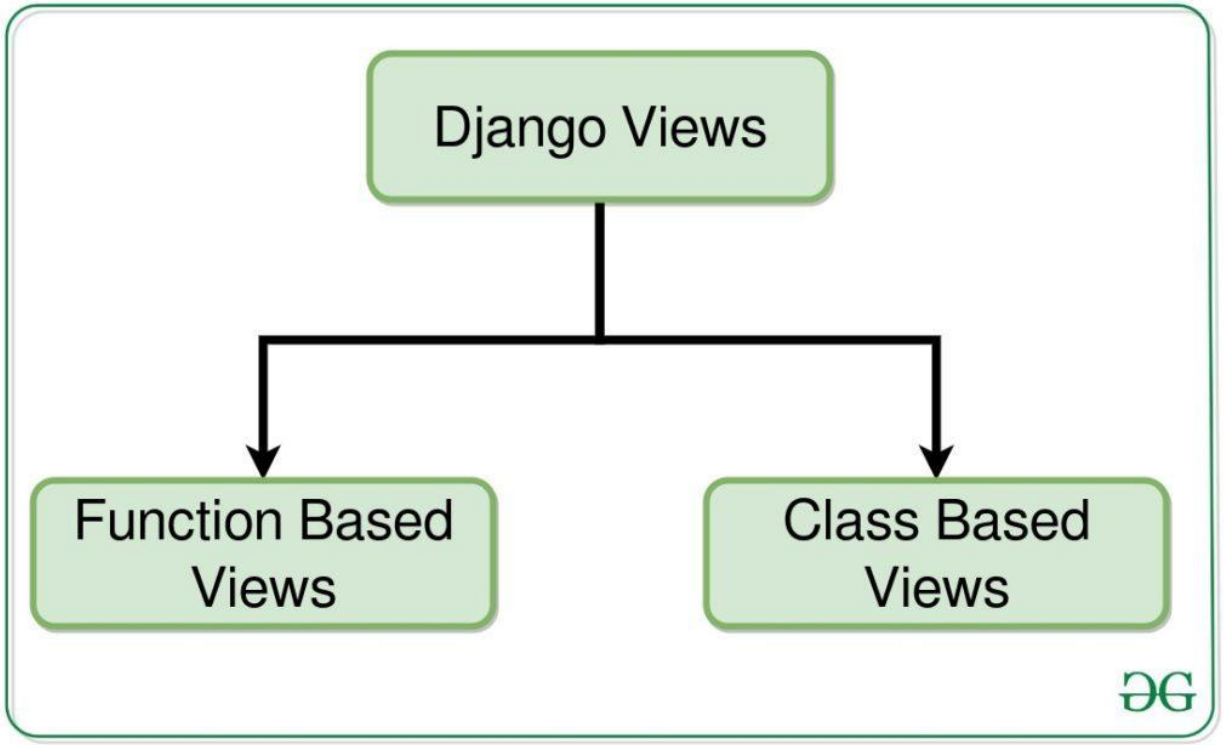
- The index view then renders our simple HTML message to a HttpResponse and sends it to the browser.

- The browser renders the response and we see our page heading.
- Every Django application follows this same basic process each time it receives a request from the browser.

Types of Views

- Django views are divided into two major categories :-

  - Function Based Views
  - Class Based Views



Types of Views (contd…)

- **Function Based Views**
- Function based views are writer using a function in python which receives as an argument HttpRequest object and returns an HttpResponse Object.

- Function based views are generally divided into 4 basic strategies, i.e., CRUD (Create, Retrieve, Update, Delete).

- CRUD is the base of any framework one is using for development.

Types of Views (contd…)

- **Class Based Views**
- Class-based views provide an alternative way to implement views as Python objects instead of functions.

- They do not replace function-based views, but have certain differences and advantages when compared to function-based views:

  - Organization of code related to specific HTTP methods (GET, POST, etc.) can be addressed by separate methods instead of conditional branching.

  - Object oriented techniques such as mixins (multiple inheritance) can be used to factor code into reusable components.

Types of Views (contd…)

- **Class Based Views**

- Class-based views are simpler and efficient to manage than function-based views.

- A function-based view with tons of lines of code can be converted into a class-based views with few lines only.

- This is where Object-Oriented Programming comes into impact.

Next Lecture Agenda

- **Django Request and Response cycle – HttpRequest and HttpResponse Objects will be discussed later on.**

- **Types of Views with example**