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:

Anaconda Powershell Prompt (anaconda3)

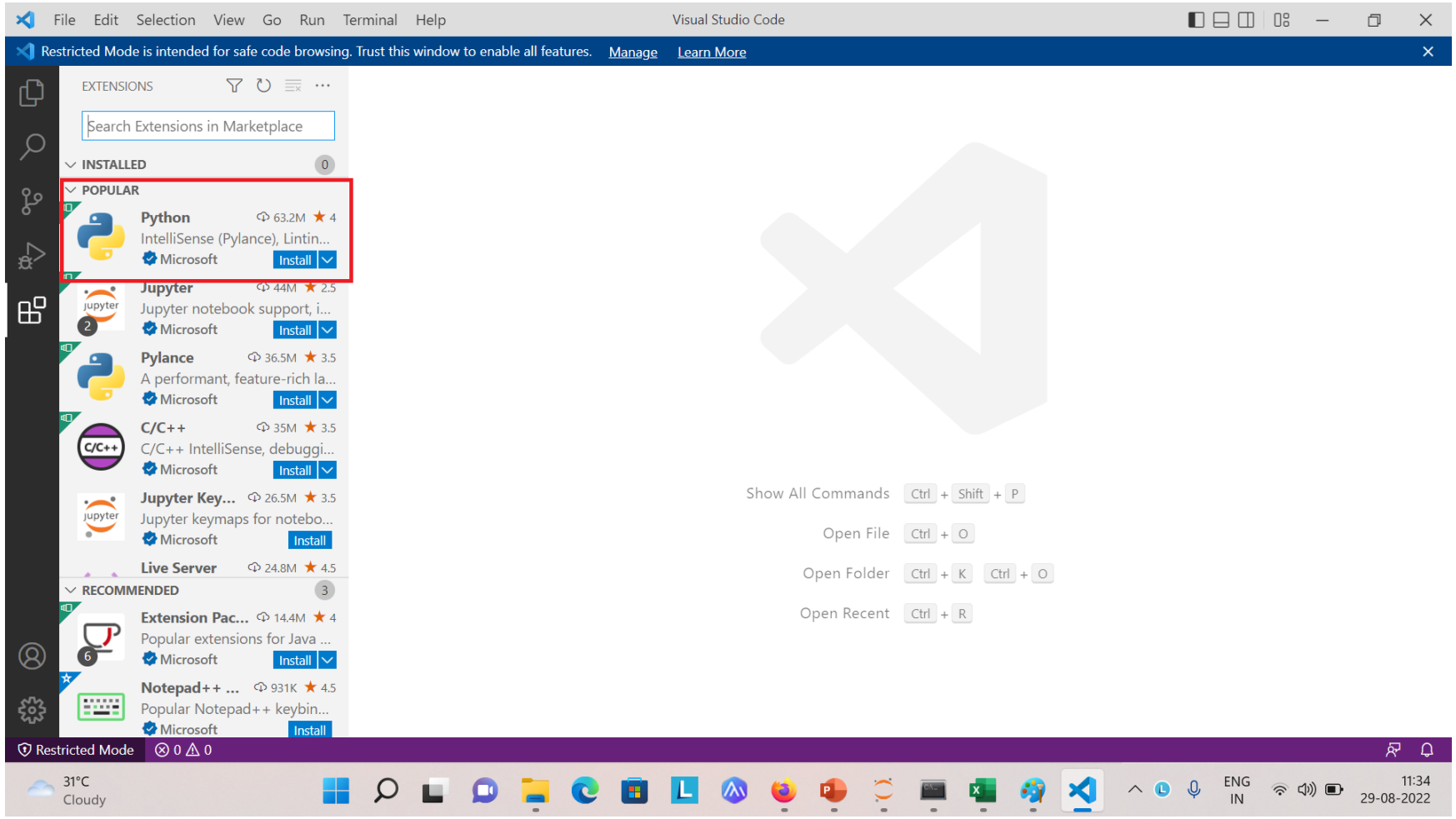
```
(base) PS C:\Users\CSE-15> python --version
Python 3.9.7
(base) PS C:\Users\CSE-15> -m pip install Django
-m : The term '-m' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the
spelling of the name, or if a path was included, verify that the path is correct and try again.
At line:1 char:1
+ -m pip install Django
+ ~~
    + CategoryInfo          : ObjectNotFound: (-m:String) [], CommandNotFoundException
    + FullyQualifiedErrorId : CommandNotFoundException

(base) PS C:\Users\CSE-15> pip install Django
Collecting Django
  Downloading Django-4.1-py3-none-any.whl (8.1 MB)
     |████████████████████████████████| 8.1 MB 2.2 MB/s
Collecting sqlparse>=0.2.2
  Downloading sqlparse-0.4.2-py3-none-any.whl (42 kB)
     |████████████████████████████████| 42 kB 217 kB/s
Collecting tzdata
  Downloading tzdata-2022.2-py2.py3-none-any.whl (336 kB)
     |████████████████████████████████| 336 kB 3.3 MB/s
Collecting asgiref<4,>=3.5.2
  Downloading asgiref-3.5.2-py3-none-any.whl (22 kB)
Installing collected packages: tzdata, sqlparse, asgiref, Django
Successfully installed Django-4.1 asgiref-3.5.2 sqlparse-0.4.2 tzdata-2022.2
(base) PS C:\Users\CSE-15>
```

Django Installation

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

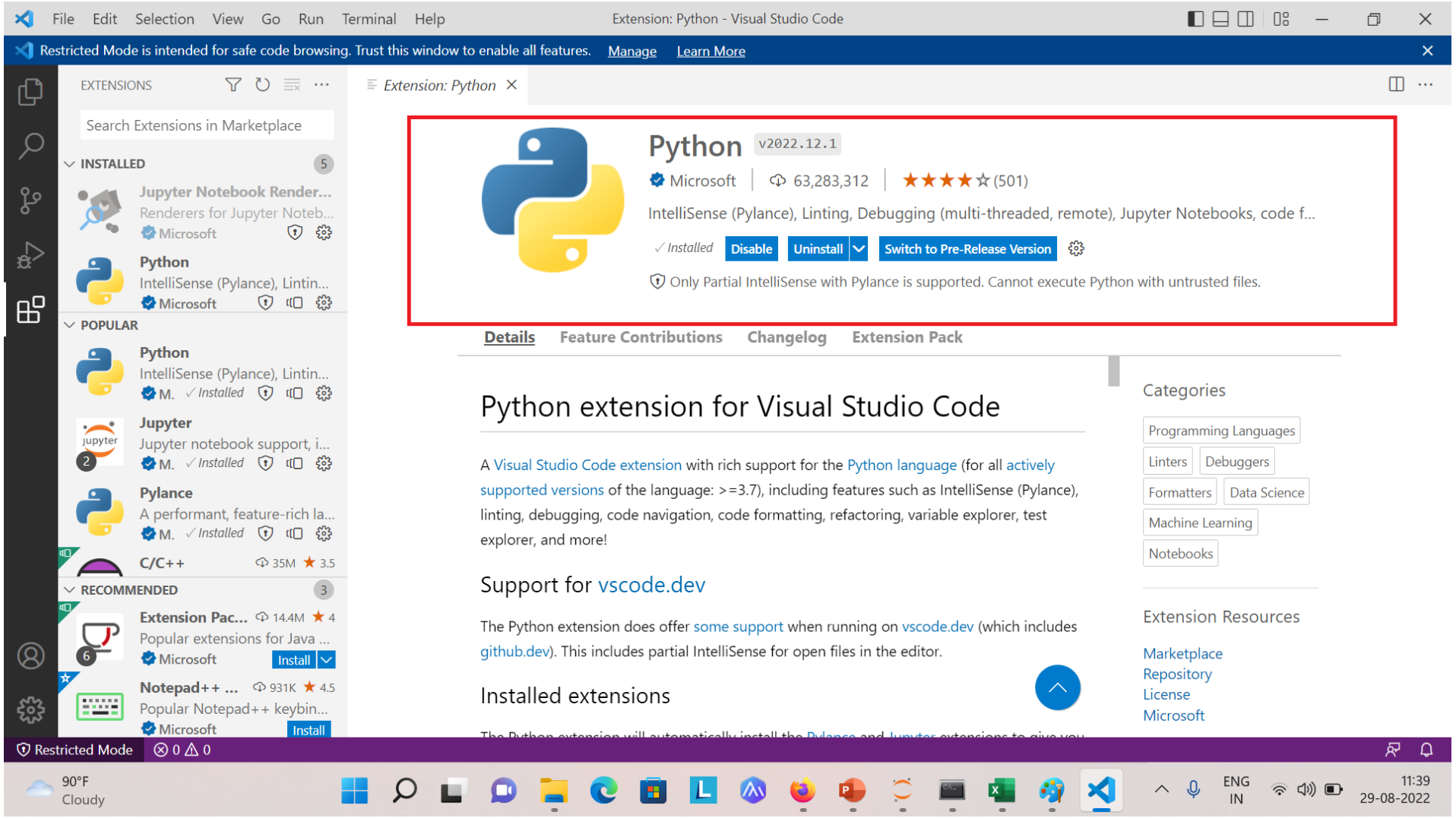
Django Installation

- 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:



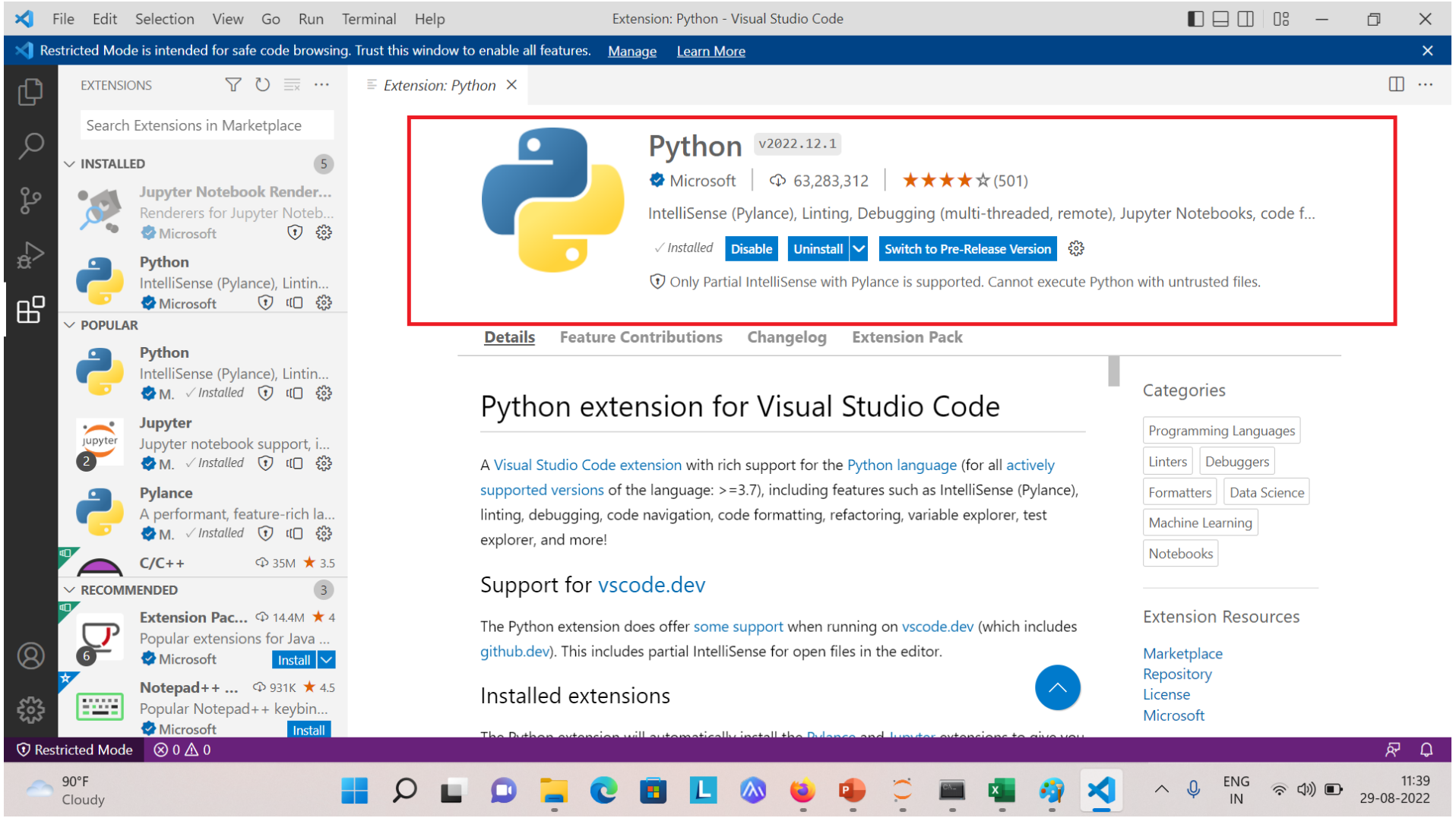
Django Installation

- After it is successfully installed, following will appear:



Django Installation

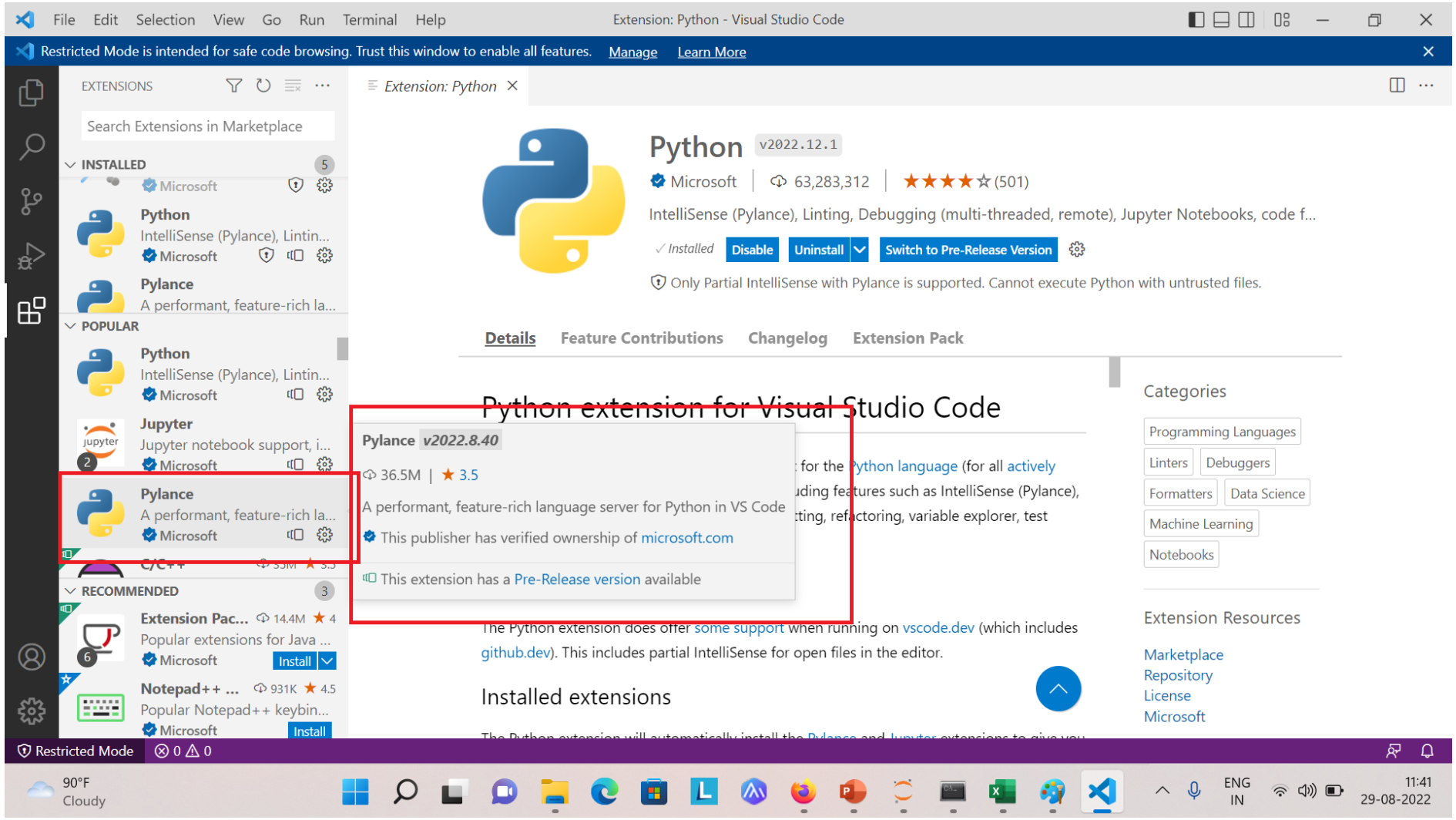
- After it is successfully installed, following will appear:



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…)

File Edit Selection View Go Run Terminal Help — manage.py - django_demo - Visual Studio Code [Administrator]

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

manage.py

```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'django_demo.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE — powershell

```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator.CSE40-NU\django_demo>
```

No Python interpreter is selected. You need to select a Python interpreter to enable features such as IntelliSense, linting, and debugging.

Source: Python (Extension) — Select Python Interpreter

OUTLINE

Select Python Interpreter ⊗ 0 ⚠ 0 — Ln 1, Col 1 — Spaces: 4 — UTF-8 — CRLF — Python

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**

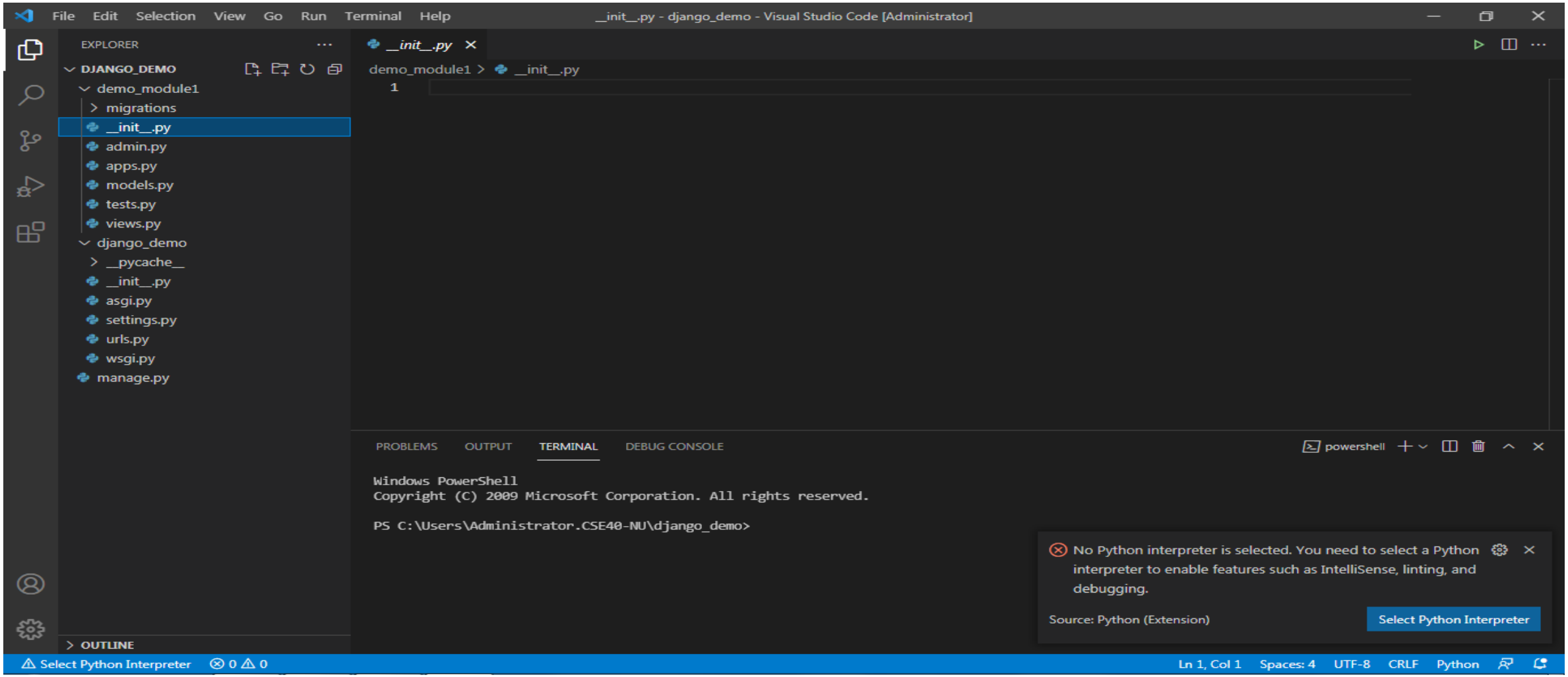
- **Makemigration:** this is done to apply new migrations that have been carried out due to the changes in the database.

- 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**

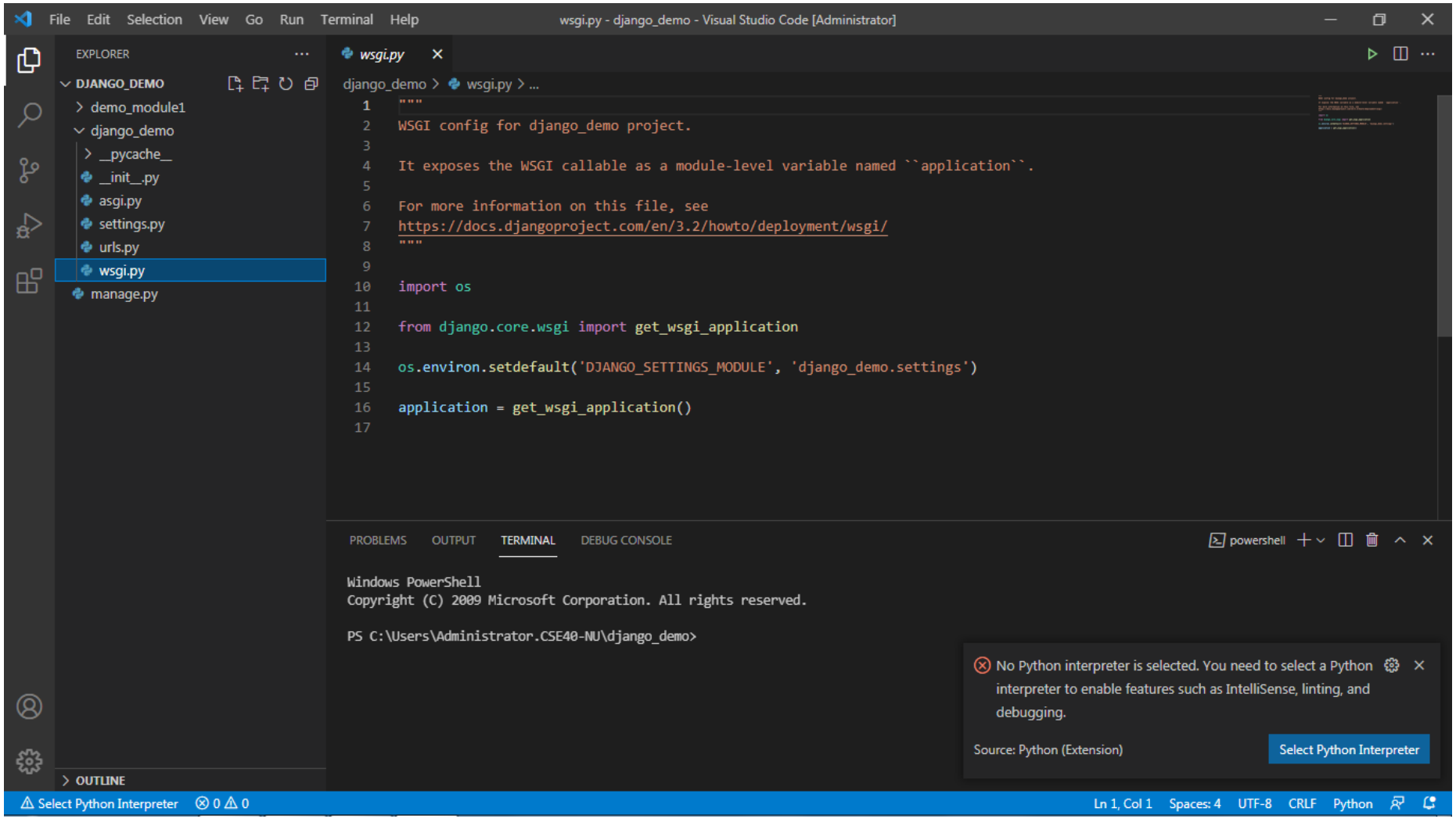
- This file remains empty and is present their only to tell that this particular **directory**(in this case django_project) is a **package**.

- **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**

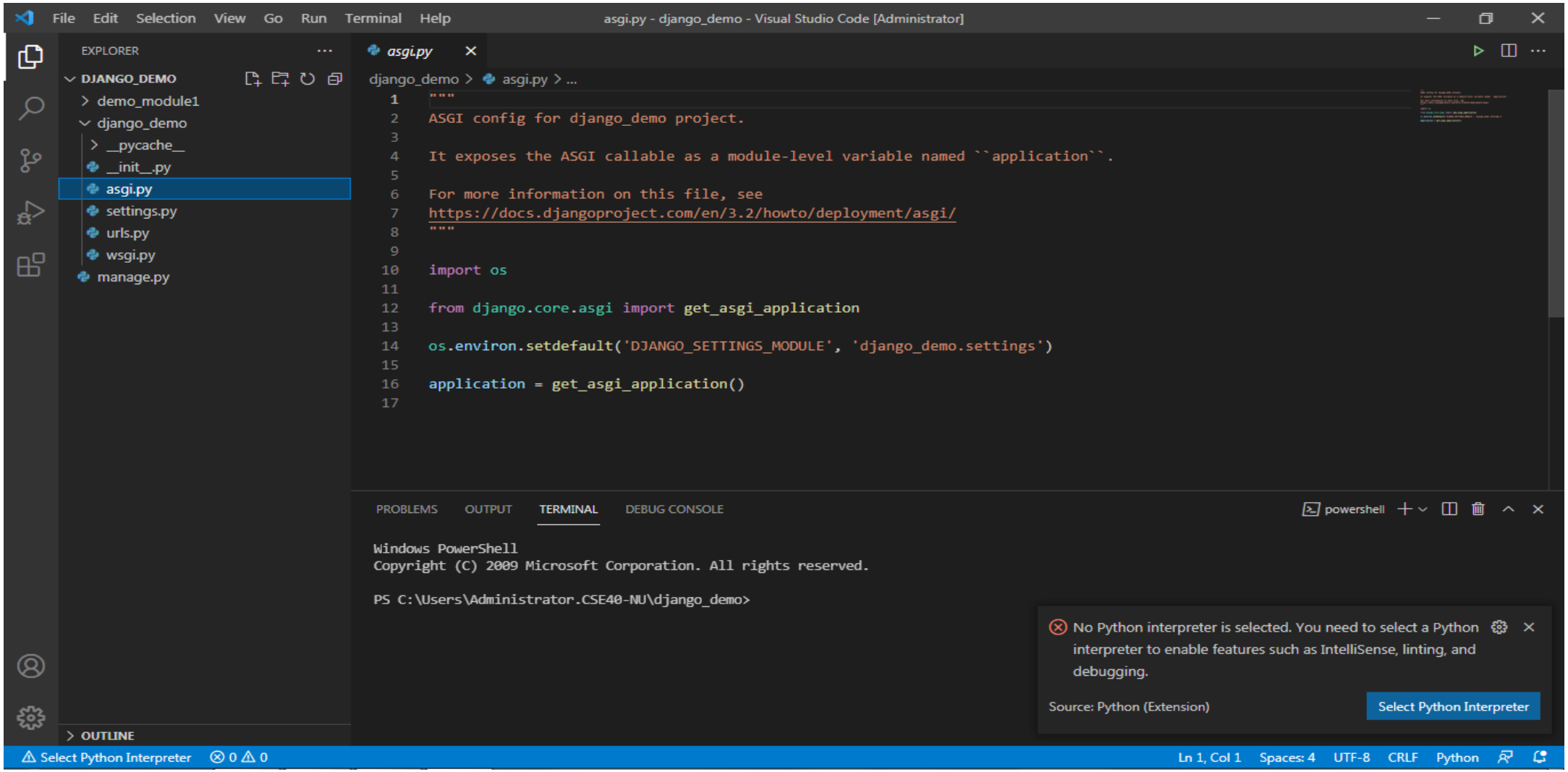
- Django is based on python which uses WSGI server for web development.
- This file is mainly concerned with that and we will not be using this file much.

- 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**.

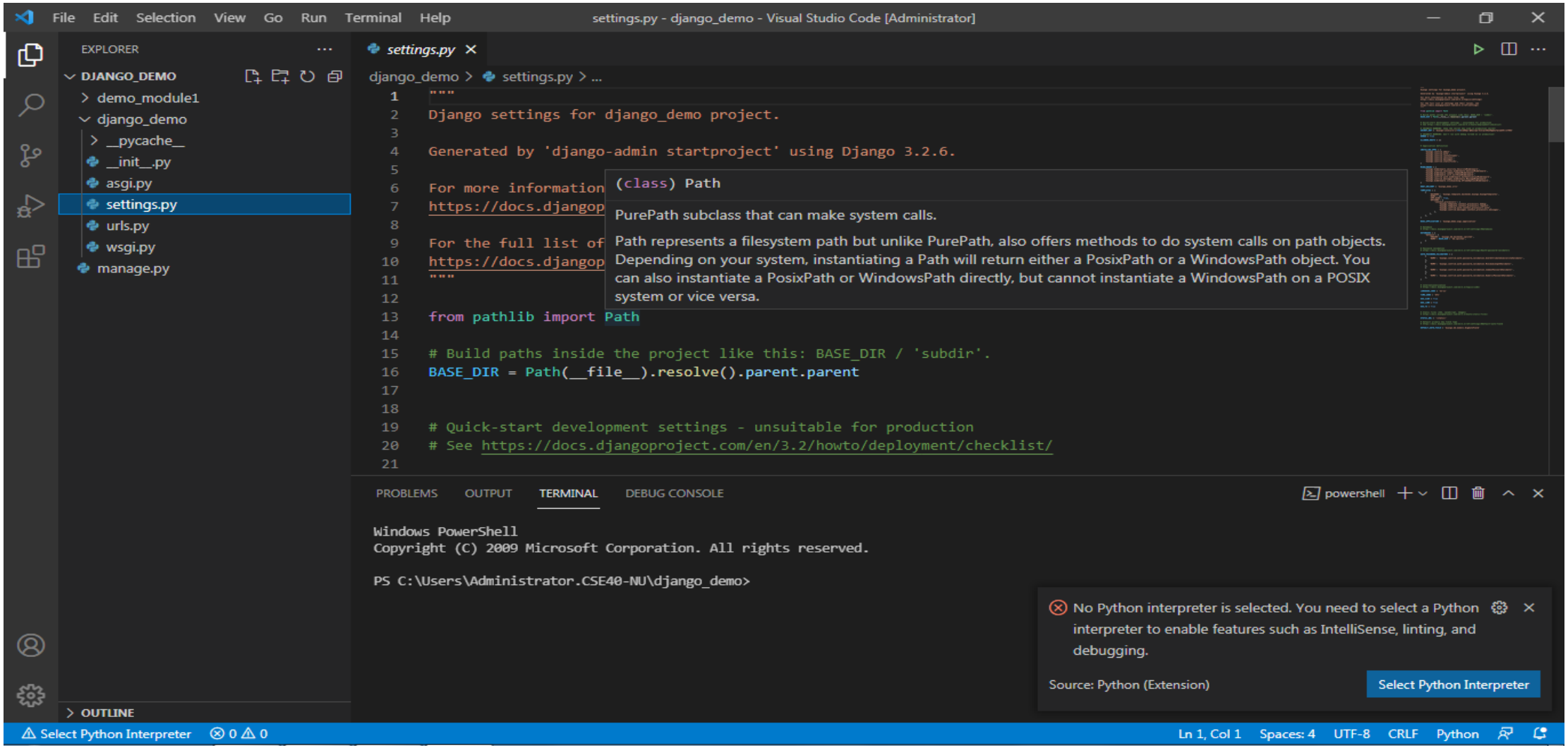
- **ASGI, short for Asynchronous Server Gateway interface** also has the work similar to WSGI but this is better than the previous one as it gives better freedom in Django development.

- 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.

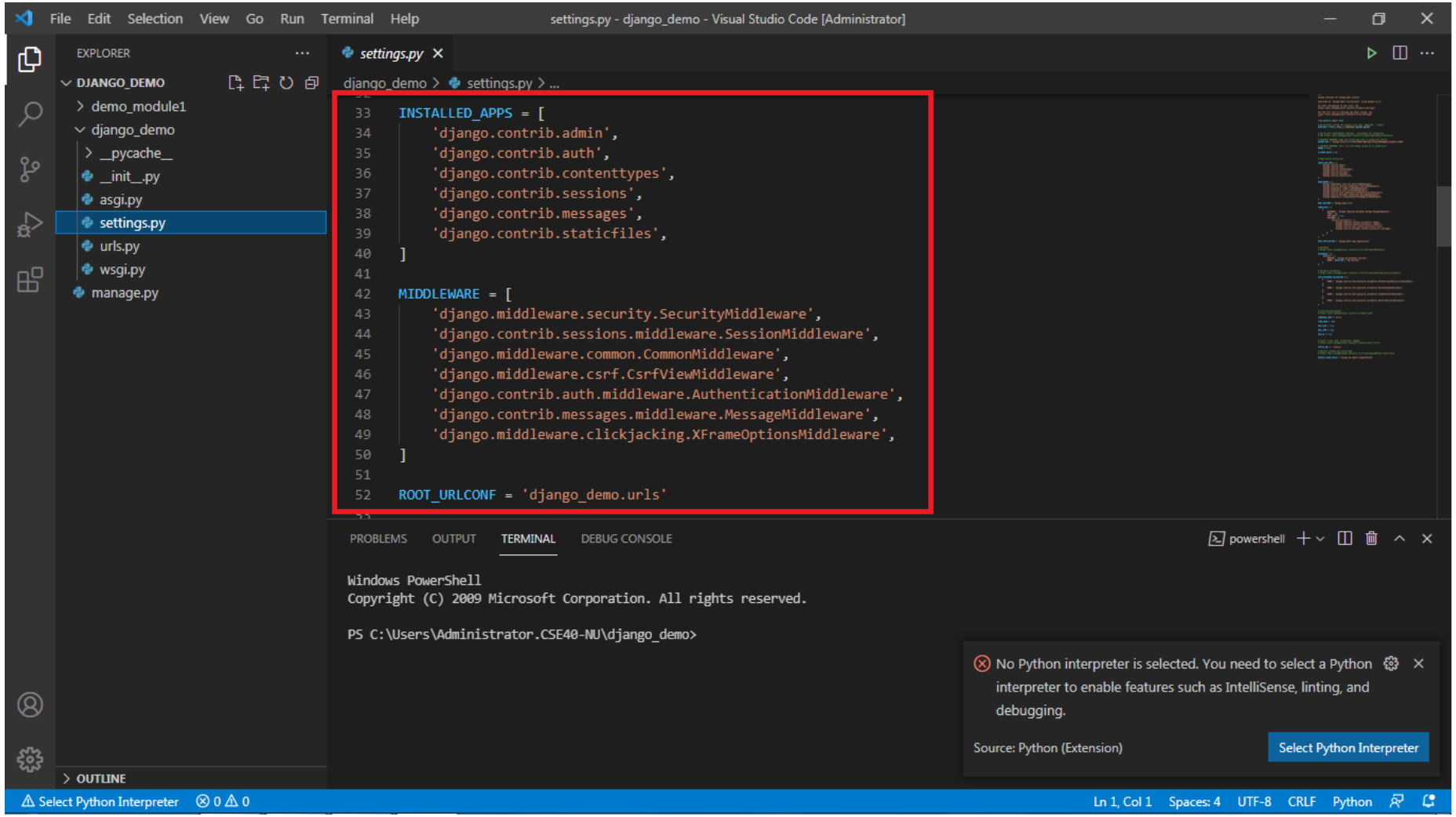
- As the name suggests this is the main settings file of the Django project.

- 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…)

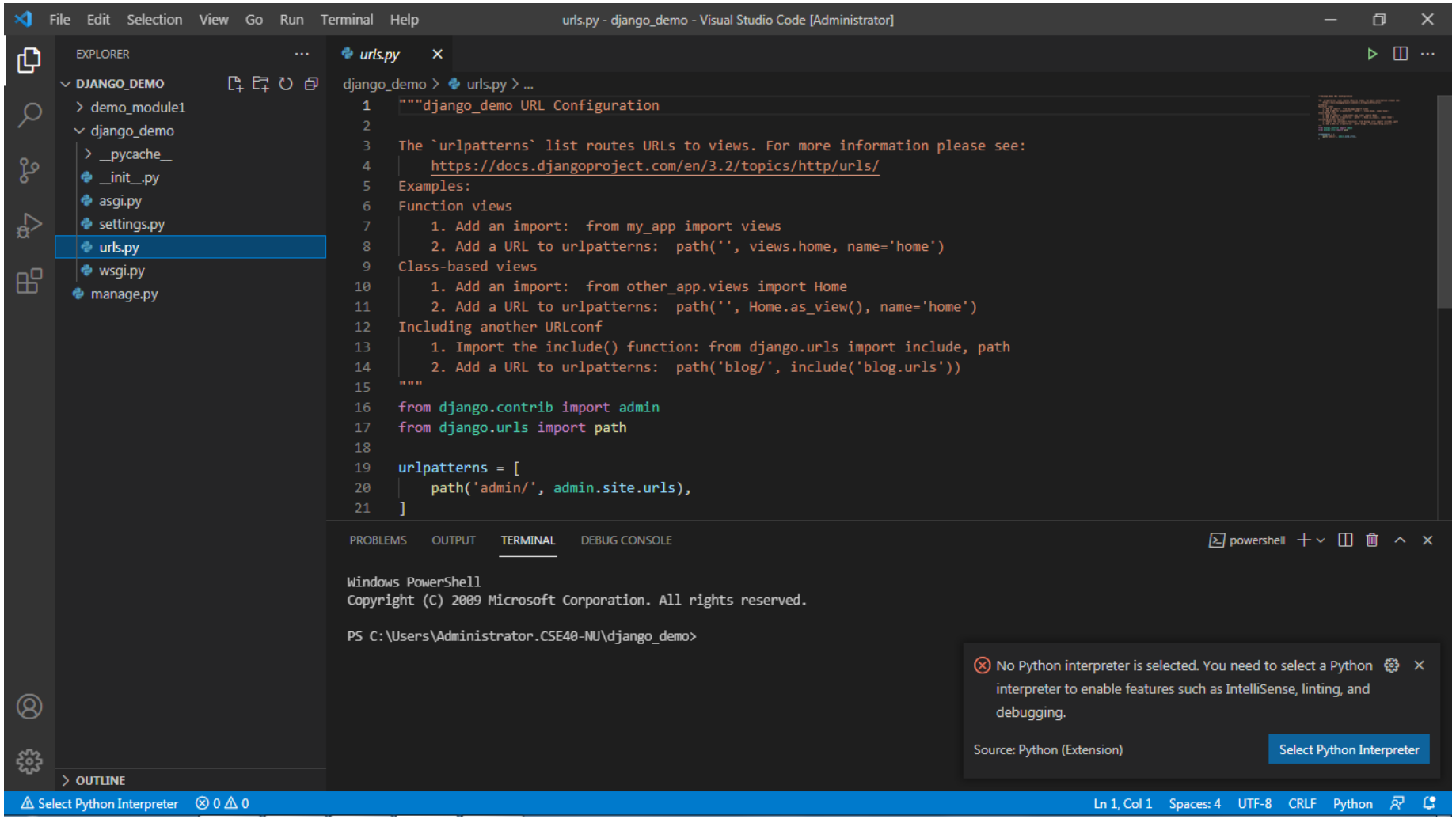
- **settings.py**

- 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 > ...

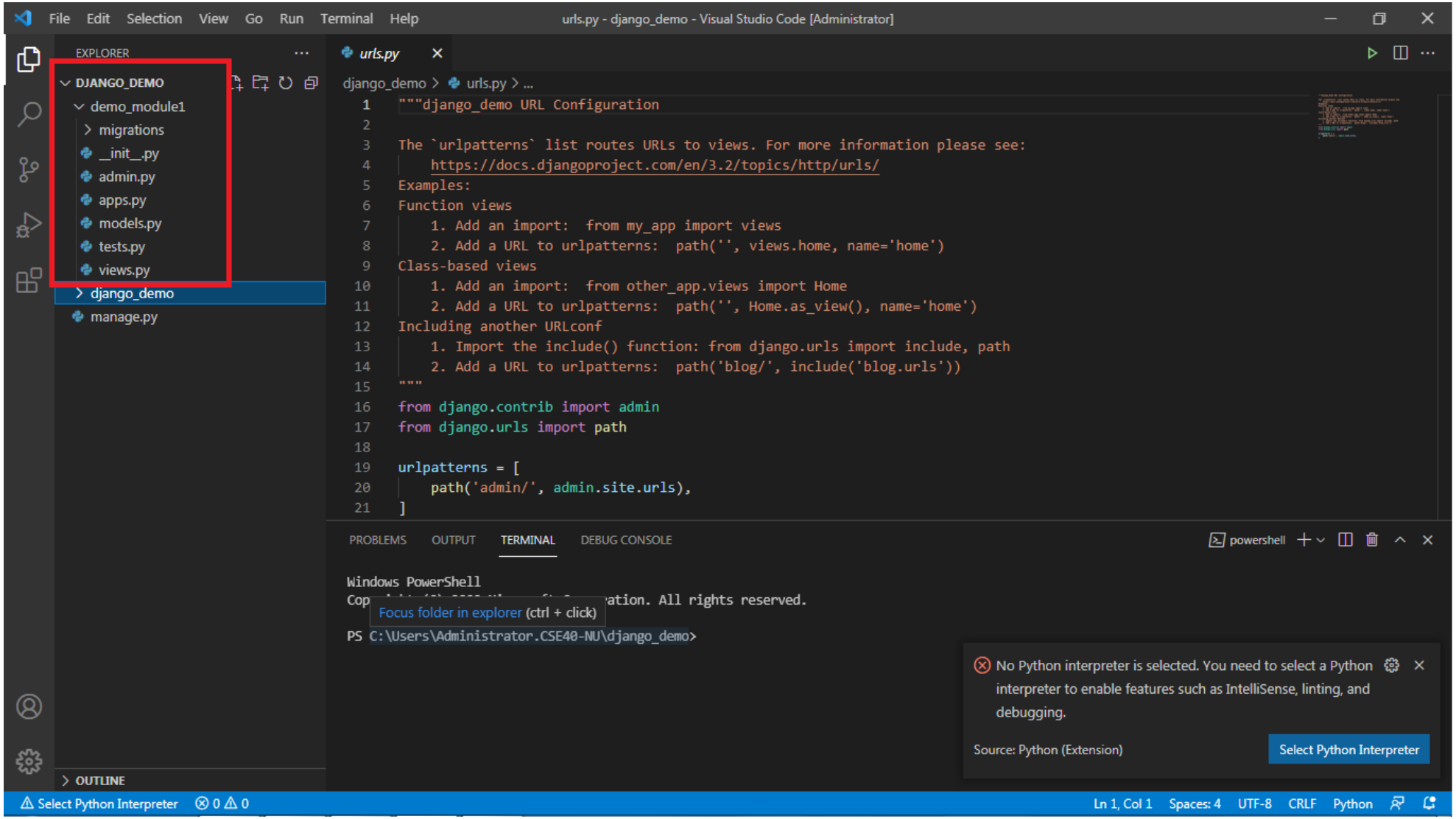
```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "django_demo.settings")
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "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…)

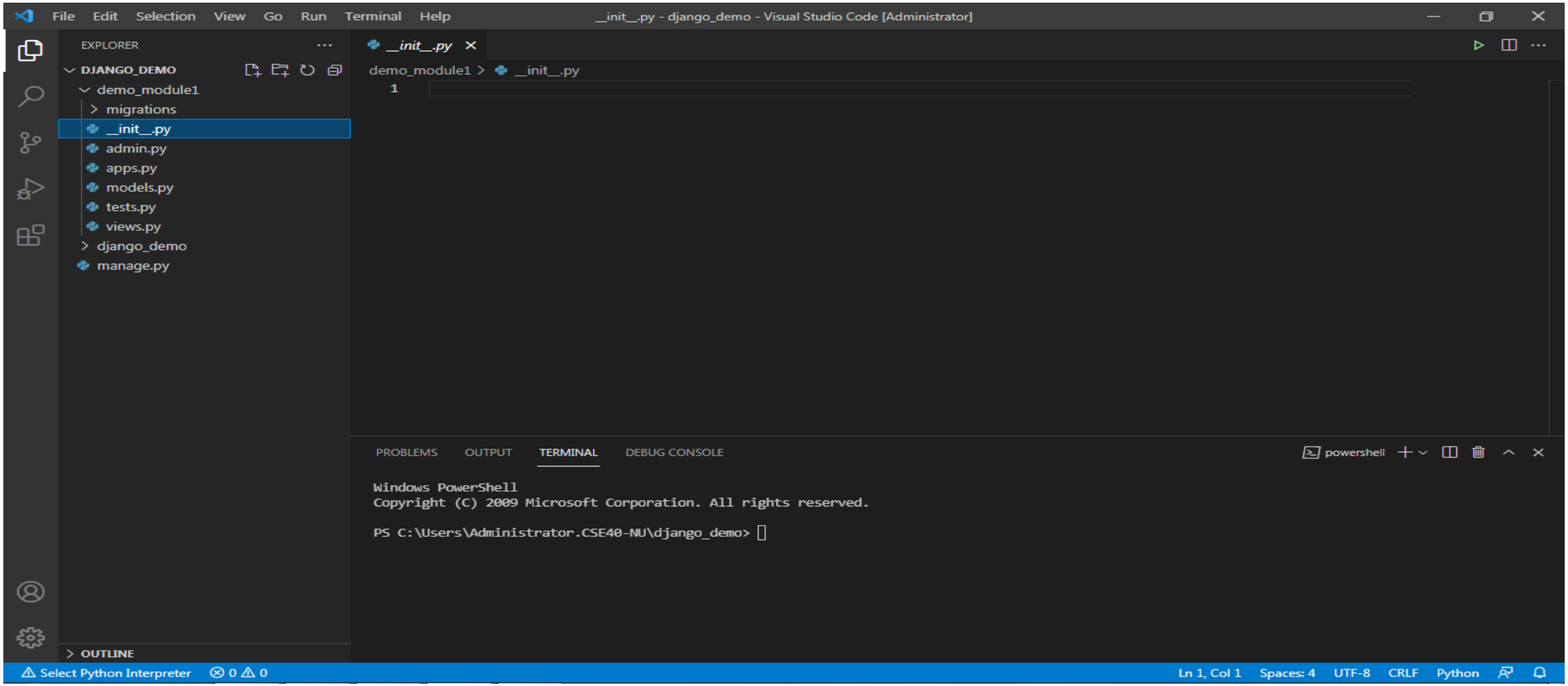
- Django uses the concept of Projects and apps for managing the codes and presents them in a readable format.

- 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**

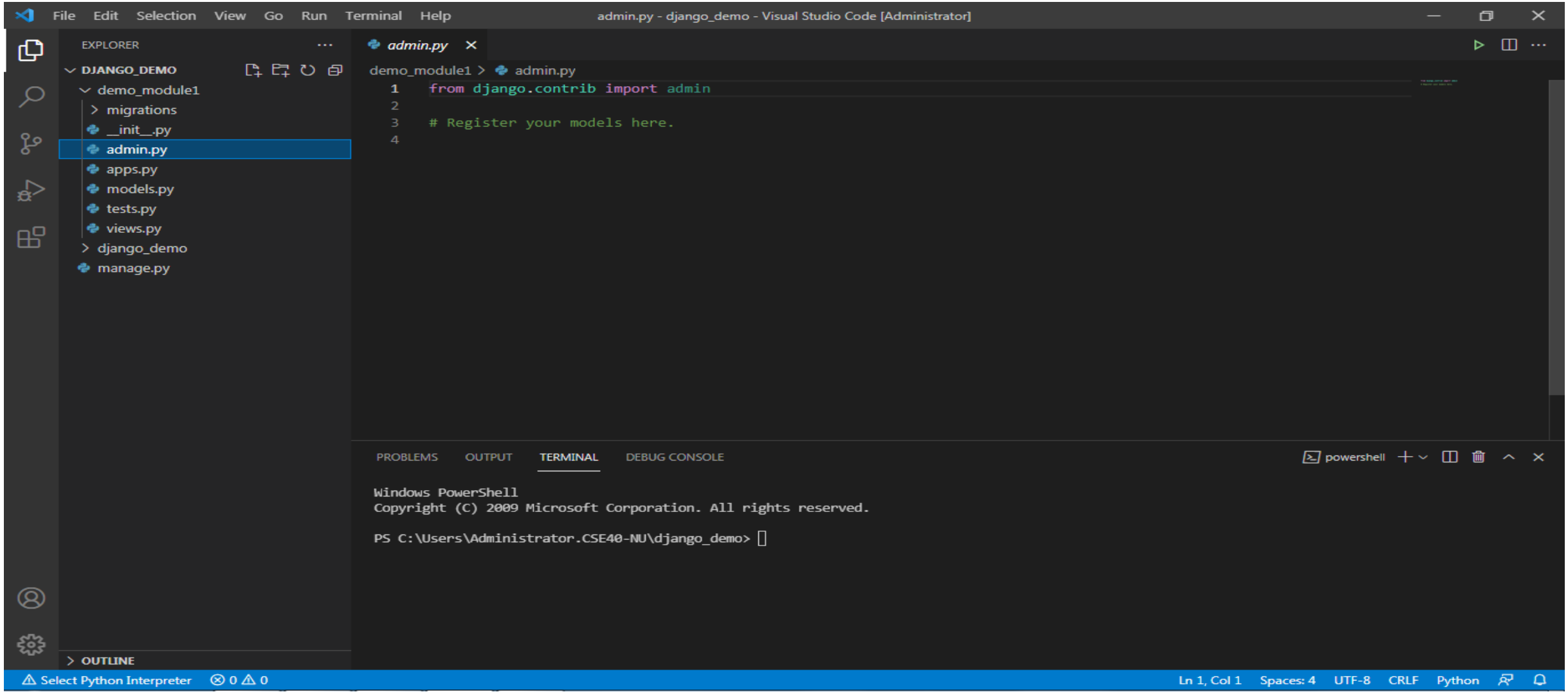
- This file has the same functionality just as in the _init_.py file in the Django project structure.

- 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.**

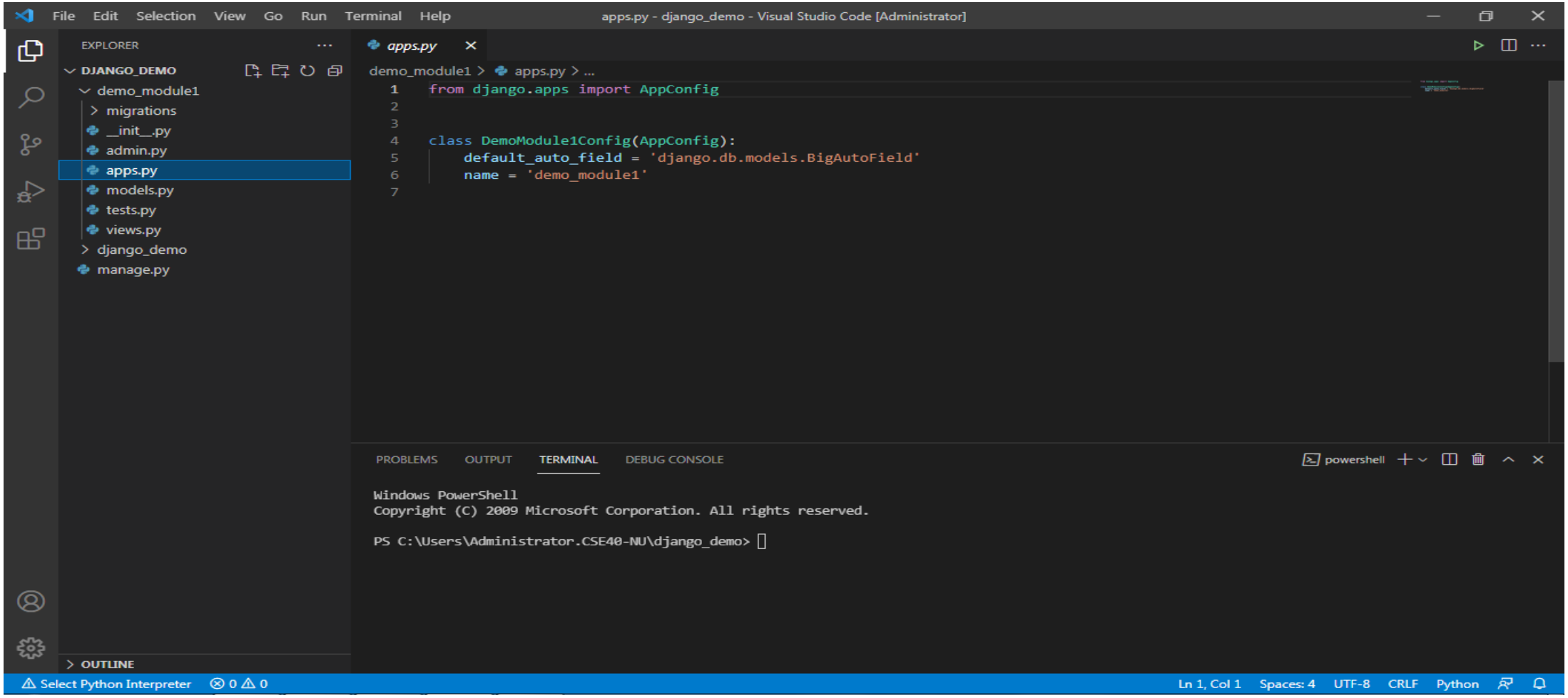
Django App Structure (contd…)

- **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.

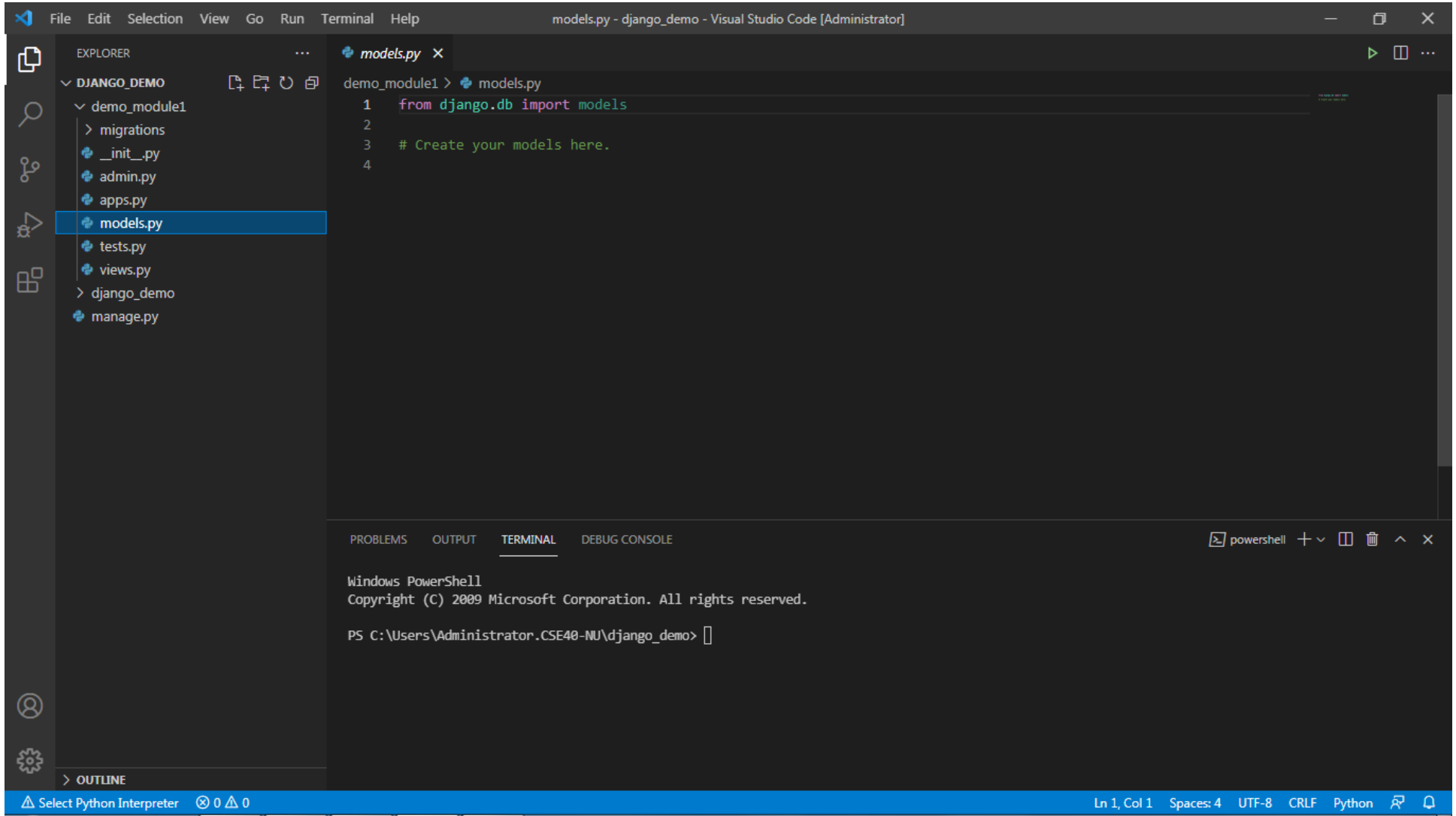
- Users can configure the attributes of their application using the apps.py file.

- 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.

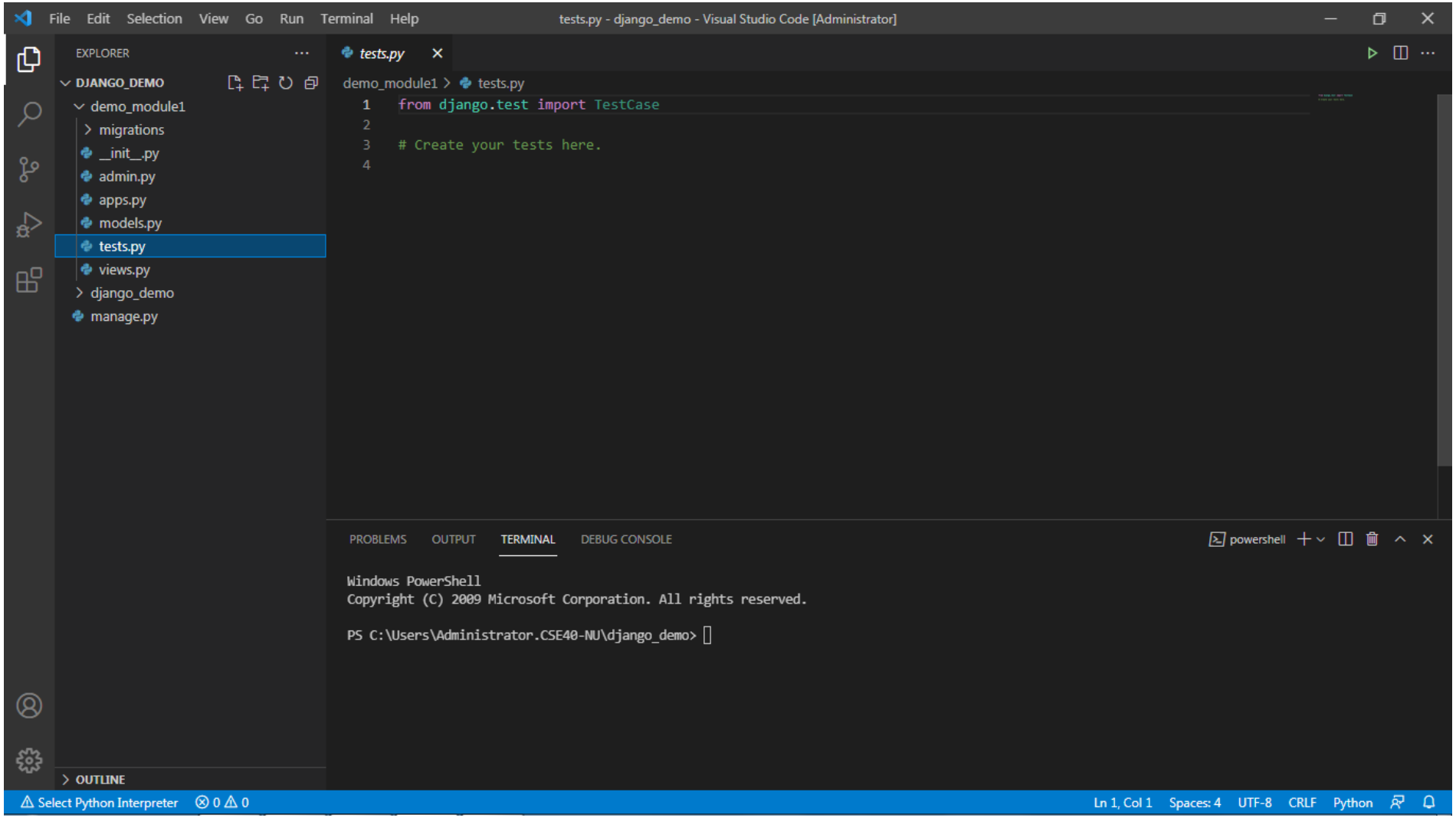
- Models define the structure of the database. It tells about the actual design, relationships between the data sets, and their attribute constraints.

- 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…)

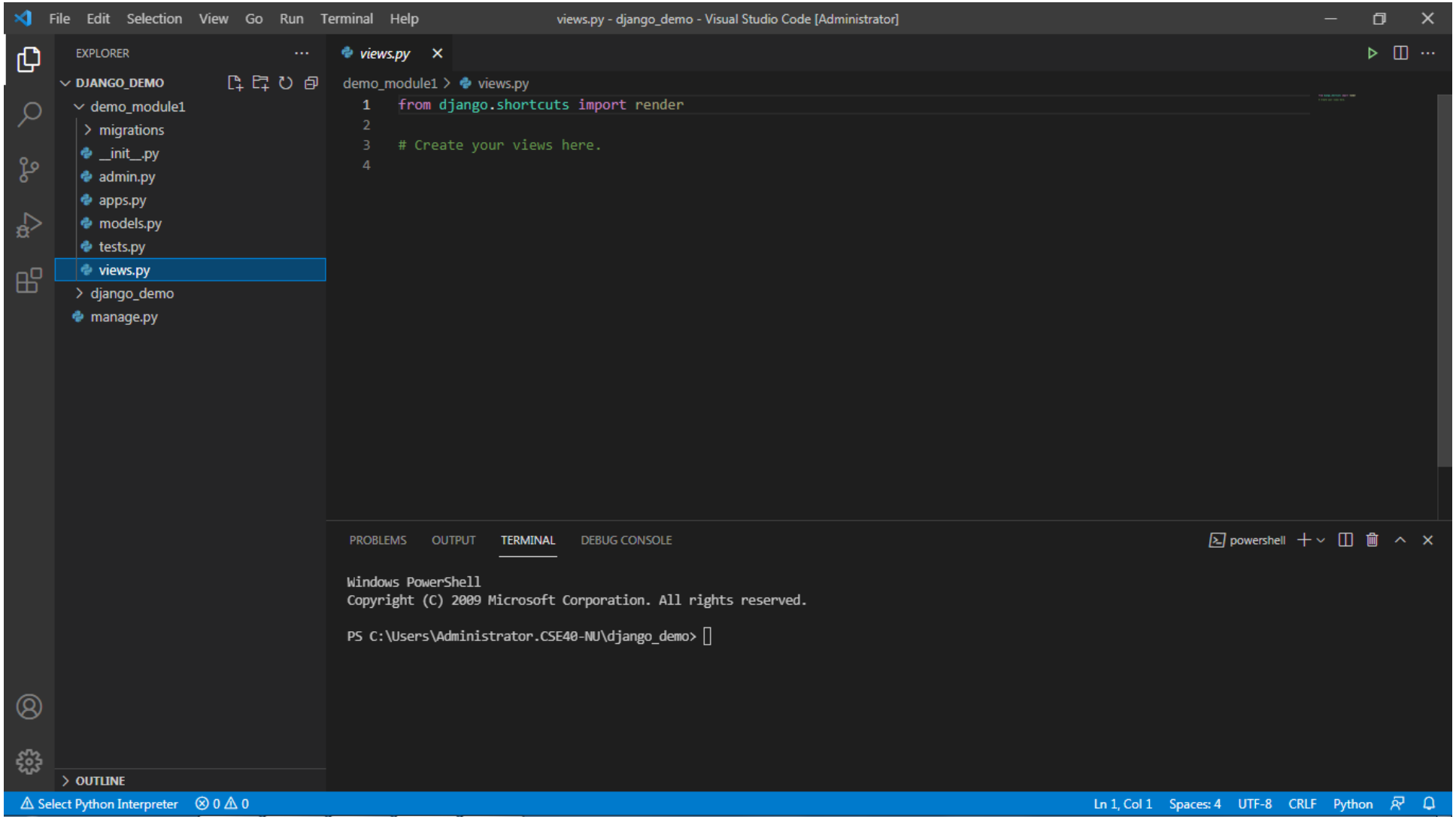
- **tests.py**

- 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.