Competency Based Learning Material (CBLM)

Web Application Development with Python

Level-4

Module: Enabling Django Framework Environment

Code: CBLM- OU-ICT-WADP-01-L4-V1

National Skills Development Authority
Chief Advisor's Office
Government of the People's Republic of Bangladesh

Copyright

National Skills Development Authority
Chief Advisor's Office
Level: 10-11, Biniyog Bhaban,
E-6 / B, Agargaon, Sher-E-Bangla Nagar Dhaka-1207, Bangladesh.
Email: ec@nsda.gov.bd
Website: www.nsda.gov.bd.
National Skills Portal: http:\\skillsportal.gov.bd

This Competency Based Learning Materials (CBLM) on "**Enabling Django Framework Environment**" under the **Web Application Development with Python , Level-4** qualification is developed based on the national competency standard approved by National Skills Development Authority (NSDA)

This document is to be used as a key reference point by the competency-based learning materials developers, teachers/trainers/assessors as a base on which to build instructional activities.

National Skills Development Authority (NSDA) is the owner of this document. Other interested parties must obtain written permission from NSDA for reproduction of information in any manner, in whole or in part, of this Competency Standard, in English or other language.

It serves as the document for providing training consistent with the requirements of industry in order to meet the qualification of individuals who graduated through the established standard via competency-based assessment for a relevant job.

This document has been developed by NSDA in association with industry representatives, academia, related specialist, trainer, and related employee. Public and private institutions may use the information contained in this CBLM for activities benefitting Bangladesh.

Approved by the Authority ……. meeting held on .....

How to use this Competency Based Learning Material (CBLM)

The module contains training materials and activities for you to complete. These activities may be completed as part of structured classroom activities or you may be required you to work at your own pace. These activities will ask you to complete associated learning and practice activities in order to gain knowledge and skills you need to achieve the learning outcomes.

1. Review the **Learning Activity** page to understand the sequence of learning activities you will undergo. This page will serve as your road map towards the achievement of competence.

2. Read the **Information Sheets.** This will give you an understanding of the jobs or tasks you are going to learn how to do. Once you have finished reading the **Information Sheets** complete the questions in the **Self-Check.**

3. **Self-**Checks are found after each **Information Sheet**. **Self-Checks** are designed to help you know how you are progressing. If you are unable to answer the questions in the **Self-Check** you will need to re-read the relevant **Information Sheet**. Once you have completed all the questions check your answers by reading the relevant **Answer Keys** found at the end of this module.

4. Next move on to the **Job Sheets. Job Sheets** provide detailed information about *how to do the job* you are being trained in. Some **Job Sheets** will also have a series of **Activity Sheets**. These sheets have been designed to introduce you to the job step by step. This is where you will apply the new knowledge you gained by reading the Information Sheets. This is your opportunity to practice the job. You may need to practice the job or activity several times before you become competent.

5. Specification **sheets**, specifying the details of the job to be performed will be provided where appropriate.

6. A review of competency is provided on the last page to help remind if all the required assessment criteria have been met. This record is for your own information and guidance and is not an official record of competency

When working though this Module always be aware of your safety and the safety of others in the training room. Should you require assistance or clarification please consult your trainer or facilitator.

When you have satisfactorily completed all the Jobs and/or Activities outlined in this module, an assessment event will be scheduled to assess if you have achieved competency in the specified learning outcomes. You will then be ready to move onto the next Unit of Competency or Module

Table of Contents

Module Content

Unit of Competency	**Enable Django Framework Environment**
Unit Code	**OU-ICT-WADP-01-L4-V1**
Module Title	**Enabling Django Framework Environment**
Module Descriptor	This module covers the knowledge, skills and attitudes required to enabling Django Framework Environment It includes the task of installing python and Django and starting a project.
Nominal Hours	10 Hours
Lerning Outcome	After completing the practice of the module, the trainees will be able to perform the following jobs: 1. Install python and Django 2. Start a project

Assessment Criteria

1. Python is installed
2. Django is installed
3. Database is set up
4. Working environment is explored
5. Project structure is explored
6. The development server is started

Learning Outcome 1: Install python and Django

Assessment Criteria	1. Python is installed 2. Django is installed 3. Database is set up
Conditions and Resources	1. Real or simulated workplace 2. CBLM 3. Handouts 4. Laptop 5. Multimedia Projector 6. Paper, Pen, Pencil, Eraser 7. Internet facilities 8. White board and marker 9. Audio Video Device
Contents	1 Installation process of Python 2 Installation process of Django 3 Database 3.1 PostgreSQL 3.2 SQLite 3.3 MySQL 3.4 NoSQL 4 Without a Database
Activities/job/Task	1. Install python and Django
Training Methods	1. Discussion 2. Presentation 3. Demonstration 4. Guided Practice 5. Individual Practice 6. Project Work 7. Problem Solving 8. Brainstorming
Assessment Methods	Assessment methods may include but not limited to 1. Written Test 2. Demonstration 3. Oral Questioning 4. Portfolio

Learning Experience 1: Install python and Django

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Activities	Recourses/Special Instructions
1. Trainee will ask the instructor about the learning materials	1. Instructor will provide the learning materials 'Install python and Django'
2. Read the Information sheet and complete the Self Checks & Check answer sheets on "Interpret the Project"	2. Read Information sheet 1: Install python and Django 3. Answer Self-check 1: Install python and Django 4. Check your answer with Answer key 1: Install python and Django
3. Read the Job/Task Sheet and Specification Sheet and perform job/Task	5. Job/Task Sheet and Specification Sheet Job Sheet-1.1: Install python in Windows Job Sheet-1.2: Install Django in Windows

Information Sheet 1: Install python and Django

Learning Objective:

After completion of this information sheet, the learners will be able to explain, define and interpret the following contents:

1.1 Installation process of Python
1.2 Installation process of Django
1.3 Setting up Database

1.1. Install Python

1.1.1. What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

1.1.2. What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.

- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

1.1.3. Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

1.1.4. Good to know

- The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
- In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

1.1.5. Python Syntax compared to other programming languages

- Python was designed for readability and has some similarities to the English language with influence from mathematics.
- Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
- Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

1.1.6. Python Install

The installation requires downloading the official Python *.exe* installer and running it on your system. The sections below will explain several options and details during the installation process.

Step 1: Select Python Version

Deciding on a version depends on what you want to do in Python. The two major versions are Python 2 and Python 3. Choosing one over the other might be better depending on your project details. If there are no constraints, choose whichever one you prefer.

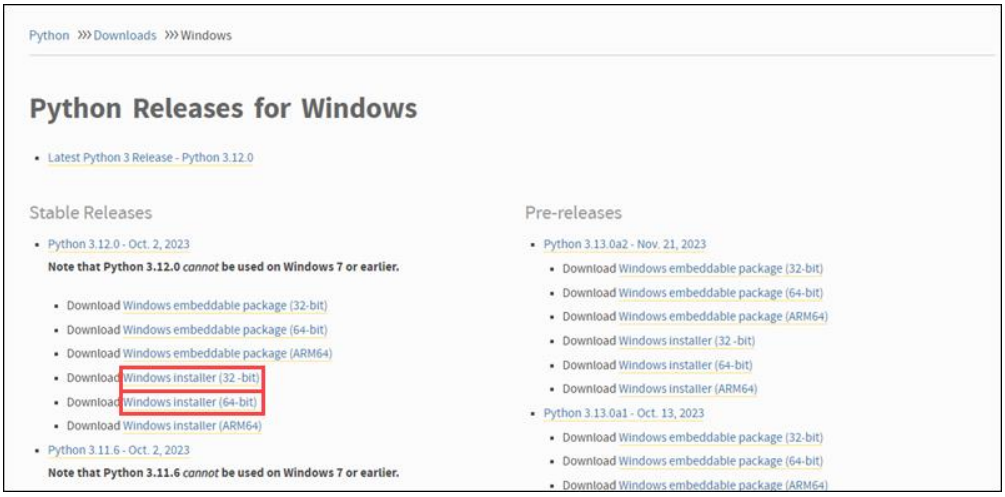
We recommend Python 3, as Python 2 reached its end of life in 2020. Download Python 2 only if you work with legacy scripts and older projects. Also, choose a stable release over the newest since the newest release may have bugs and issues.

Step 2: Download Python Executable Installer

Start by downloading the Python executable installer for Windows:

1. Open a web browser and navigate to the Downloads for Windows section of the official Python website.

2. Locate the desired Python version.



3. Click the link to download the file. Choose either the Windows 32-bit or 64-bit installer.

The download is approximately 25MB.

Step 3: Run Executable Installer

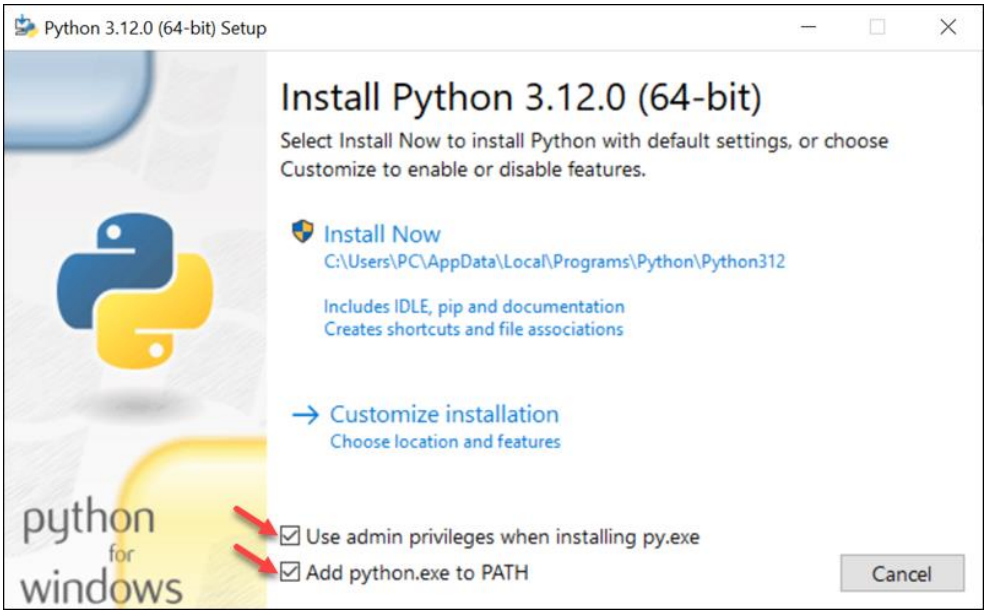
The steps below guide you through the installation process:

1. Run the downloaded **Python Installer**.

2. The installation window shows two checkboxes:

- **Admin privileges**. The parameter controls whether to install Python for the current or all system users. This option allows you to change the installation folder for Python.

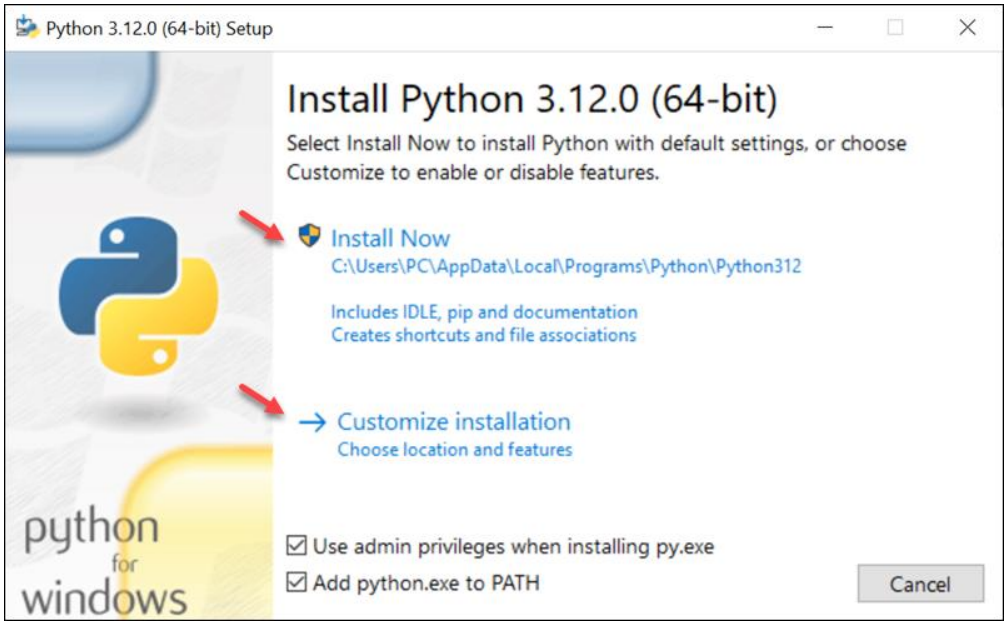
- Add Python to PATH. The second option places the executable in the PATH variable after installation. You can also add Python to the PATH environment variable manually later.



For the most straightforward installation, we recommend ticking both checkboxes.

3. Select the **Install Now** option for the recommended installation (in that case, skip the next two steps).

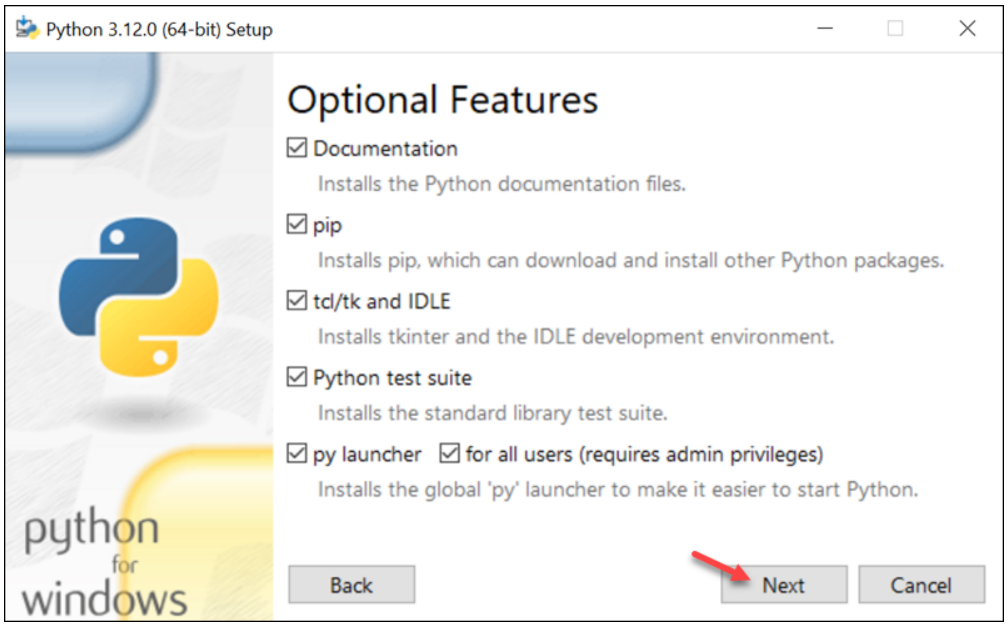
To adjust the default installation options, choose **Customize installation** instead and proceed to the following step.



The default installation installs Python to *C:\Users\[user]\AppData\Local\Programs\Python\Python[version]* for the current user. It includes IDLE (the default Python editor), the PIP package manager, and additional documentation. The installer also creates necessary shortcuts and file associations.

Customizing the installation allows changing these installation options and parameters.

4. Choose the optional installation features. Python works without these features, but adding them improves the program's usability.

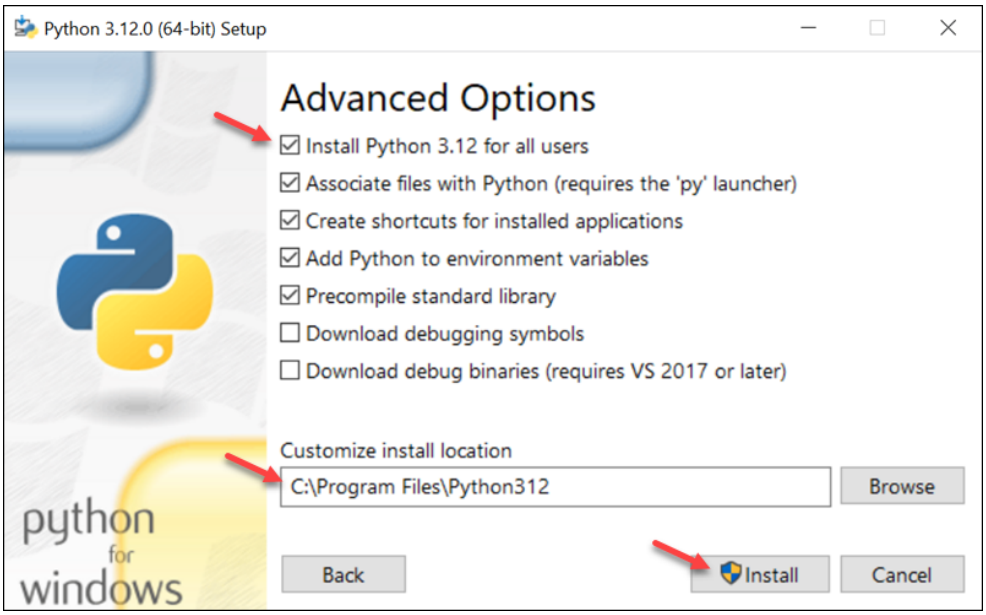


Click **Next** to proceed to the Advanced Options screen.

5. The second part of customizing the installation includes advanced options.

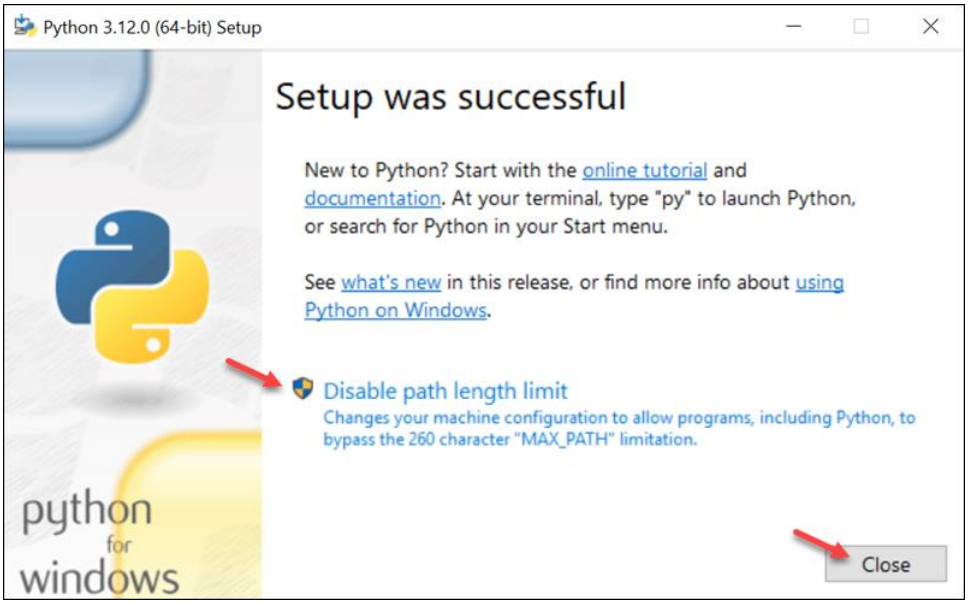
Choose whether to install Python for all users. The option changes the install location to *C:\Program Files\Python[version]*. If selecting the location manually, a common choice is *C:\Python[version]* because it avoids spaces in the path, and all users can access it. Due to administrative rights, both paths may cause issues during package installation.

Other advanced options include creating shortcuts, file associations, and adding Python to PATH.



After picking the appropriate options, click **Install** to start the installation.

6. Select whether to disable the path length limit. Choosing this option will allow Python to bypass the 260-character **MAX_PATH** limit.



The option will not affect any other system settings, and disabling it resolves potential name-length issues. We recommend selecting the option and closing the setup.

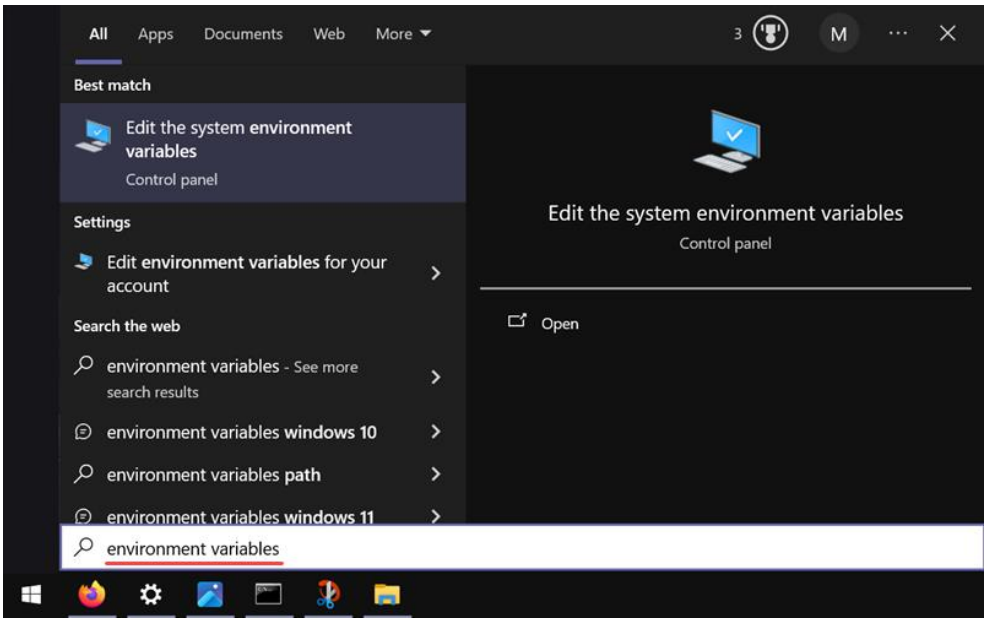
Step 4: Add Python to Path (Optional)

If the Python installer does not include the **Add Python to PATH** checkbox or you have not selected that option, continue in this step. Otherwise, skip to the next step.

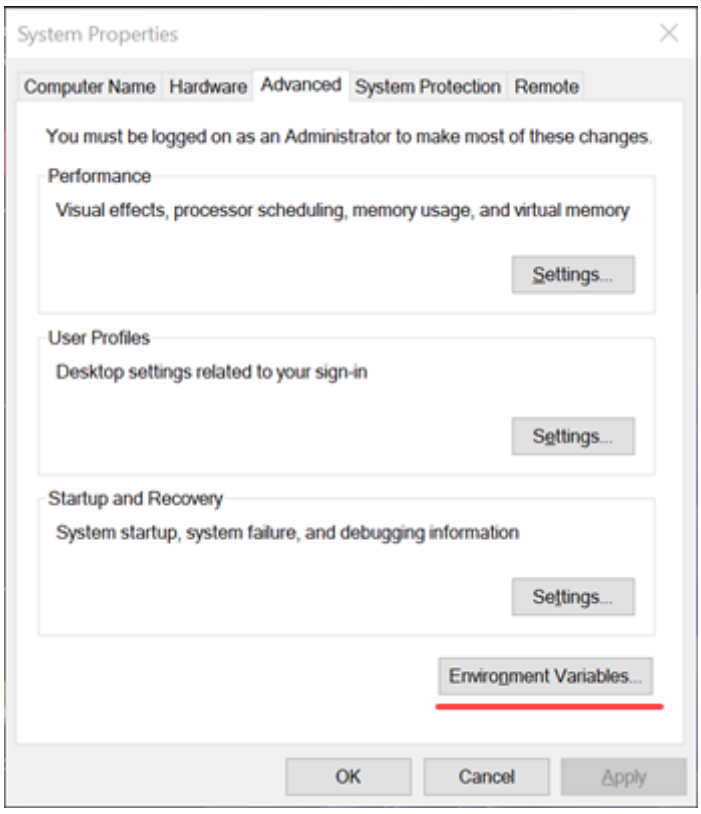
Adding the Python path to the PATH variable alleviates the need to use the full path to access the Python program in the command line. It instructs Windows to review all the folders added to the PATH environment variable and to look for the *python.exe* program in those folders.

To add Python to PATH, do the following:

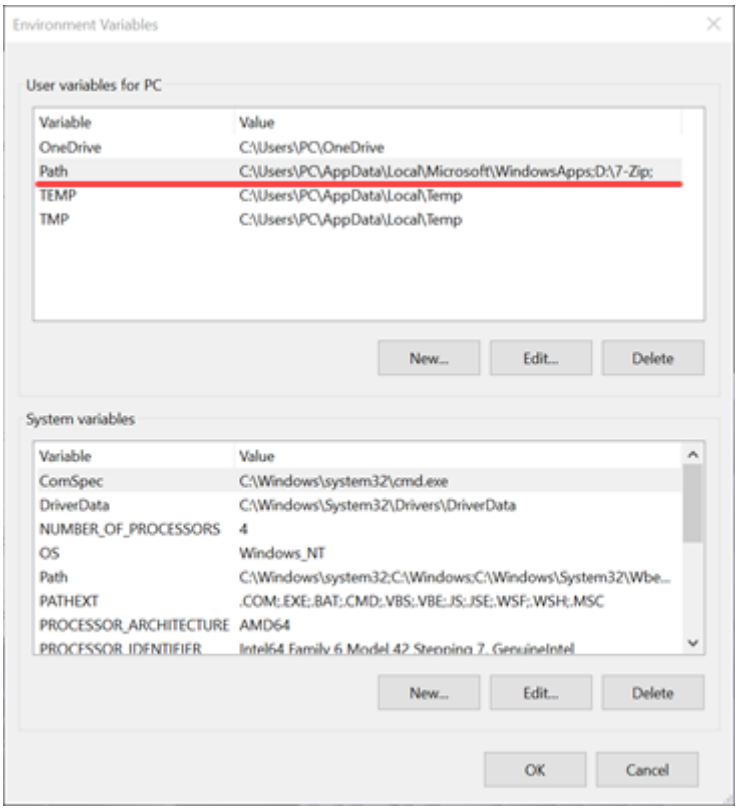
1. In the **Start** menu, search for **Environment Variables** and press **Enter**.



2. Click **Environment Variables** to open the overview screen.

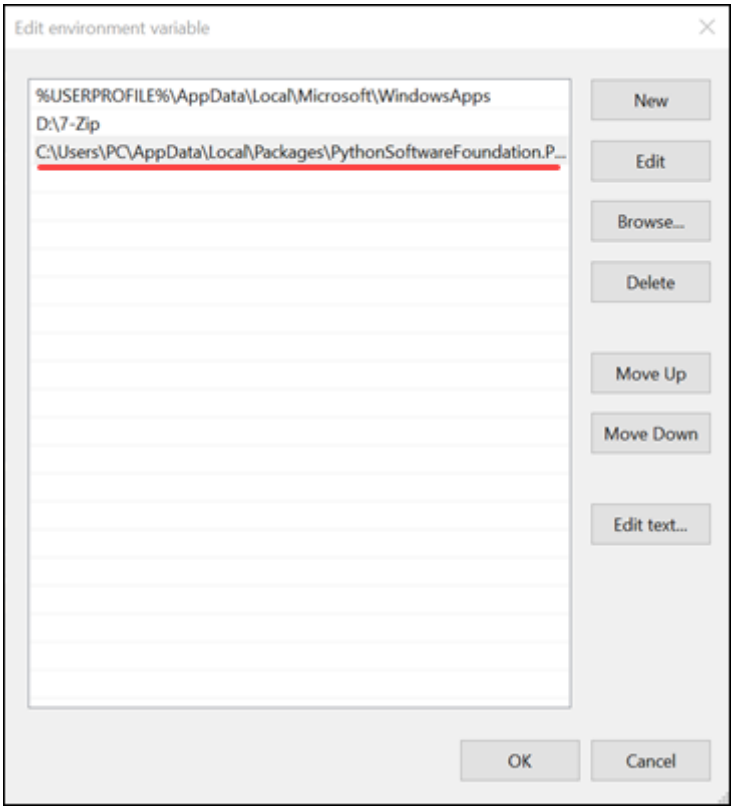


3. Double-click **Path** on the list to edit it.



Alternatively, select the variable and click the **Edit** button.

4. Double-click the first empty field and paste the Python installation folder path.



Alternatively, click the **New** button instead and paste the path.

5. Click **OK** to save the changes. If the command prompt is open, restart it for the following step.

Step 5: Verify Python Was Installed on Windows

The first way to verify that Python was installed successfully is through the command line. Open the command prompt and run the following command:

python --version

```
C:\Users\PC>python --version
Python 3.12.0
```

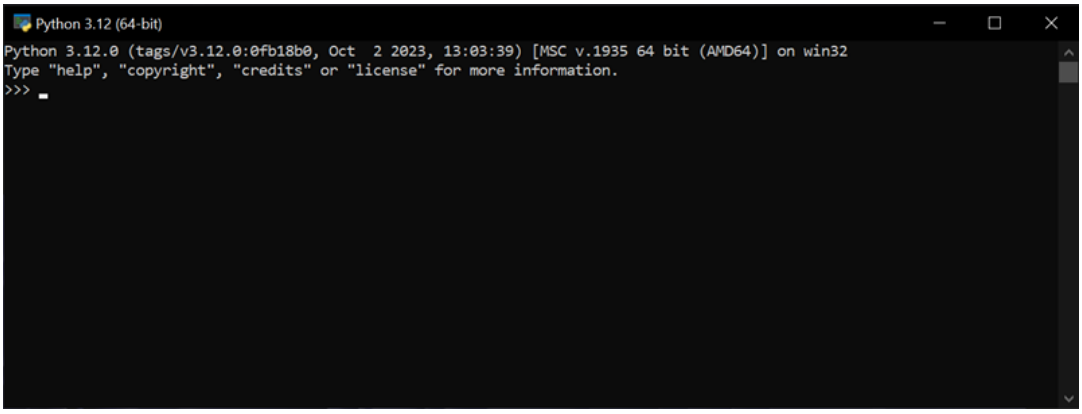
The output shows the installed Python version.

The second way is to use the GUI to verify the Python installation. Follow the steps below to run the Python interpreter or IDLE:

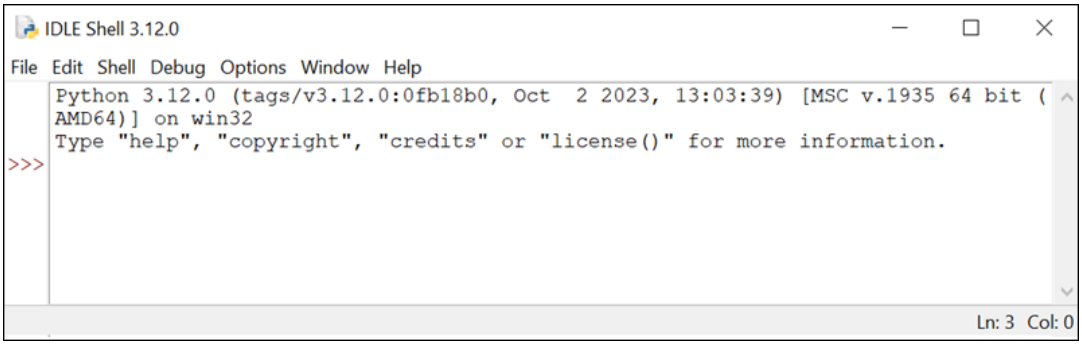
1. Navigate to the directory where Python was installed on the system.

2. Double-click *python.exe* (the Python interpreter) or IDLE.

3. The interpreter opens the command prompt and shows the following window:



Running IDLE opens Python's built-in IDE:



In both cases, the installed Python version shows on the screen, and the editor is ready for use.

Step 6: Verify PIP Was Installed

To verify whether PIP was installed, enter the following command in the command prompt:

pip --version

If it was installed successfully, you should see the PIP version number, the executable path, and the Python version:

PIP has not been installed yet if you get the following output:

'pip' is not recognized as an internal or external command,

Operable program or batch file.

If an older version of Python is installed or the PIP installation option is disabled during installation, PIP will not be available.

1.1.7. Python Quickstart

Python is an interpreted programming language, this means that as a developer you write Python (.py) files in a text editor and then put those files into the python interpreter to be executed.

The way to run a python file is like this on the command line:

```
C:\Users\Your Name>python helloworld.py
```

Where "helloworld.py" is the name of your python file.

Let's write our first Python file, called helloworld.py, which can be done in any text editor.

helloworld.py

```
print("Hello, World!")
```

Simple as that. Save your file. Open your command line, navigate to the directory where you saved your file, and run:

```
C:\Users\Your Name>python helloworld.py
```

The output should read:

```
Hello, World!
```

Congratulations, you have written and executed your first Python program.

1.2. Install Django

1.2.1. What is Django?

Django is a Python framework that makes it easier to create web sites using Python.
Django takes care of the difficult stuff so that you can concentrate on building your web applications.

Django emphasizes reusability of components, also referred to as DRY (Don't Repeat Yourself), and comes with ready-to-use features like login system, database connection and CRUD operations (Create Read Update Delete).
Django is especially helpful for database driven websites.

1.2.2. How does Django Work?

Django follows the MVT design pattern (Model View Template).

- Model The data you want to present, usually data from a database.
- View A request handler that returns the relevant template and content based on the request from the user.
- Template A text file (like an HTML file) containing the layout of the web page, with logic on how to display the data.

1.2.3. Model

The model provides data from the database.

In Django, the data is delivered as an Object Relational Mapping (ORM), which is a technique designed to make it easier to work with databases.
The most common way to extract data from a database is SQL. One problem with SQL is that you have to have a pretty good understanding of the database structure to be able to work with it.

Django, with ORM, makes it easier to communicate with the database, without having to write complex SQL statements.
The models are usually located in a file called models.py.

1.2.4. View

A view is a function or method that takes http requests as arguments, imports the
relevant model(s), and finds out what data to send to the template, and returns the final result.
The views are usually located in a file called views.py.

1.2.5. Template

A template is a file where you describe how the result should be represented.
Templates are often .html files, with HTML code describing the layout of a web page, but it can also be in other file formats to present other results, but we will concentrate on .html files.
Django uses standard HTML to describe the layout, but uses Django tags to add logic:
<h1>My Homepage</h1>

My name is {{ firstname }}.
The templates of an application is located in a folder named templates.

1.2.6. URLs

Django also provides a way to navigate around the different pages in a website.
When a user requests a URL, Django decides which *view* it will send it to.
This is done in a file called urls.py.

When you have installed Django and created your first Django web application, and the browser requests the URL, this is basically what happens

1. Django receives the URL, checks the urls.py file, and calls the view that matches the URL.
2. The view, located in views.py, checks for relevant models.
3. The models are imported from the models.py file.

4. The view then sends the data to a specified template in the template folder.
5. The template contains HTML and Django tags, and with the data it returns finished HTML content back to the browser.

Django can do a lot more than this, but this is basically what you will learn in this tutorial, and are the basic steps in a simple web application made with Django.

1.2.7. Django History

Django was invented by Lawrence Journal-World in 2003, to meet the short deadlines in the newspaper and at the same time meeting the demands of experienced web developers.

Initial release to the public was in July 2005.
Latest version of Django is 4.0.3 (March 2022).

1.2.8. Django Requires Python

To check if your system has Python installed, run this command in the command prompt:

```
python --version
```

If Python is installed, you will get a result with the version number, like this

```
Python 3.9.2
```

If you find that you do not have Python installed on your computer, then you can download it for free from the following website: https://www.python.org/

1.2.9. PIP

To install Django, you must use a package manager like PIP, which is included in Python from version 3.4.

To check if your system has PIP installed, run this command in the command prompt:

```
pip --version
```

If PIP is installed, you will get a result with the version number.

For me, on a windows machine, the result looks like this:

```
pip 20.2.3 from c:\python39\lib\site-packages\pip (python 3.9)
```

If you do not have PIP installed, you can download and install it from this page: https://pypi.org/project/pip/

1.2.10. Install Django

Now, that we have created a virtual environment, we are ready to install Django.

Django is installed using pip, with this command:

Windows:

```
(myworld) C:\Users\Your Name>py -m pip install Django
```

Unix/MacOS:

```
(myworld) ... $ python -m pip install Django
```

Which will give a result that looks like this (at least on my Windows machine):

```
Collecting Django
  Downloading Django-4.0.3-py3-none-any.whl (8.0 MB)
     |████████████████████████████████| 8.0 MB 2.2 MB/s
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.2-py3-none-any.whl (42 kB)
Collecting asgiref<4,>=3.4.1
  Downloading asgiref-3.5.0-py3-none-any.whl (22 kB)
Collecting tzdata; sys_platform == "win32"
  Downloading tzdata-2021.5-py2.py3-none-any.whl (339 kB)
     |████████████████████████████████| 339 kB 6.4 MB/s
Installing collected packages: sqlparse, asgiref, tzdata, Django
Successfully installed Django-4.0.3 asgiref-3.5.0 sqlparse-0.4.2 tzdata-2021.5
WARNING: You are using pip version 20.2.3; however, version 22.3 is available.
You should consider upgrading via the 'C:\Users\Your
Name\myworld\Scripts\python.exe -m pip install --upgrade pip' command.
```

That's it! Now you have installed Django in your new project, running in a virtual environment!

1.2.11. Windows, Mac, or Unix?

You can run this project on either one. There are some small differences, like when writing commands in the command prompt, Windows uses py as the first word in the command line, while Unix and MacOS use python:

Windows:

```
py --version
```

Unix/MacOS:

```
python --version
```

Check Django Version

You can check if Django is installed by asking for its version number like this:

```
(myworld) C:\Users\Your Name>django-admin --version
```

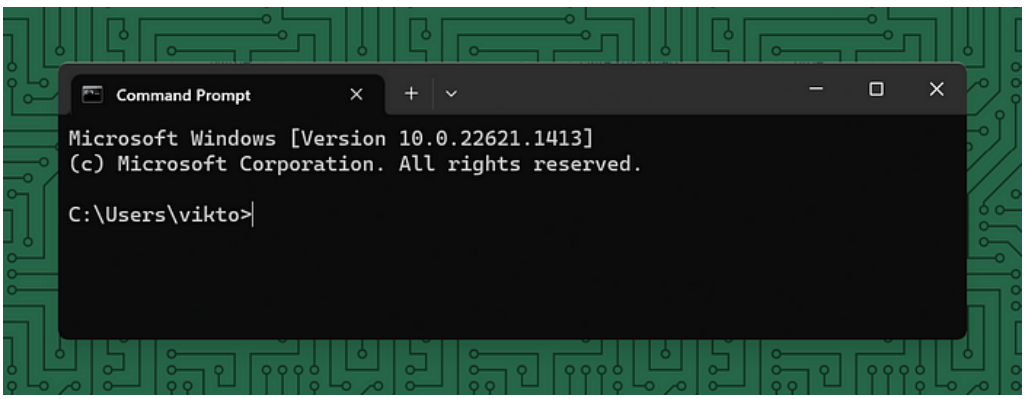
If Django is installed, you will get a result with the version number:

Install Django in Windows

Make sure you have the following prerequisites installed on your Windows machine to follow the instructions:

- Python 3.6 or higher
- Pipenv

We will use the built-in Command Prompt (CMD) for this tutorial. You can search for CMD to open it using Windows search.



Command Prompt

If you don’t have pipenv installed, you can install it by running the following command in your CMD **after Python was installed**:
pip install pipenv

Once these prerequisites are installed, you can install and launch Django.

Step 1: Create a new project directory
Open your CMD and create a new directory for your Django project. You can do this by running the following command:
mkdir myproject
Change your current directory to the newly created directory by running:
cd myproject

Step 2: Install Django using Pipenv
To install Django using pipenv, run the following command:
pipenv install django
This command will create a new virtual environment for your project and install Django and its dependencies within this environment.

Step 3: Launch the pipenv shell
pipenv shell
This command will activate the virtual environment created in the previous step, and you’ll see the name of the environment displayed in your CMD.

Virtual environment name

To exit pipenv virtual environment, type in exit. That will terminate the virtual environment, so you won’t see its name in the CMD anymore.

Step 4: Create a new Django project

The following commands must be executed while your virtual environment is active. Otherwise, they won't work.

To create a new Django project, run the following command:

django-admin startproject myproject .

This command will create a new Django project named "myproject" in the current directory (note the . dot at the end of the command).

This command doesn't have any output. So if you don't see anything after executing it, don't worry. A new directory named "myproject" with Django files will be created. That's how you know it worked.

Step 5: Launch the development server

To launch the development server, navigate to the root directory of your project (where the manage.py file is located) and run the following command:

python manage.py runserver

This command will start the development server, and you'll see output similar to the following:

Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.

Run 'python manage.py migrate' to apply them.

March 19, 2023 - 10:30:52

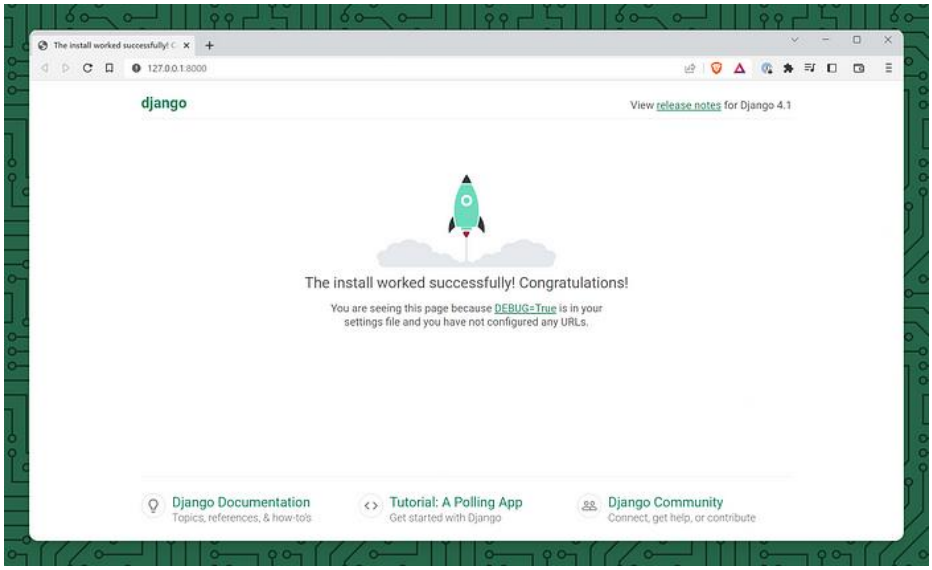
Django version 4.1, using settings 'myproject.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CTRL-BREAK.

This output indicates that the development server is running and listening for incoming connections on port 8000.

Open your browser and go to http://127.0.0.1:8000/ to see Django's page:



Django app's homepage

If you see this page in your browser, Django has been successfully installed and runs using the development server.

1.3. Setup Database

1.3.1. PostgreSQL

PostgreSQL, also sometimes called Postgres, is a powerful and popular open-source relational database management system (RDBMS). Here's a quick rundown of its key features:

- **Free and Open-Source:** Unlike some other major database systems, PostgreSQL is completely free to use and modify. This makes it a popular choice for individual developers and companies that are cost-conscious.
- **Object-Relational:** While it primarily uses SQL for data manipulation, PostgreSQL also incorporates some object-oriented features. This allows for more complex data structures and greater flexibility in how you model your data.
- **ACID Transactions:** PostgreSQL ensures data integrity through ACID compliant transactions. ACID stands for Atomicity, Consistency, Isolation, and Durability, which are essential properties for ensuring reliable data updates.
- **Rich Feature Set:** PostgreSQL offers a wide range of features, including support for complex data types, stored procedures, triggers, and more. This makes it suitable for a variety of demanding applications.

- **Cross-Platform:** You can run PostgreSQL on most major operating systems, including Linux, macOS, and Windows. This gives you flexibility in choosing the platform that best suits your needs.

Django supports PostgreSQL 12 and higher. psycopg 3.1.8+ or psycopg2 2.8.4+ is required, though the latest psycopg 3.1.8+ is recommended.

PostgreSQL connection settings

The Connection Service File allows you to save connection settings for each so-called “service” locally.

So when you have a database called gis on a local PostgreSQL with port 5432 and username/password is docker/docker you can store this as a service called my-local-gis.

```
# Local GIS Database for Testing purposes
[my-local-gis]
host=localhost port=5432 dbname=gis user=docker password=docker
```

This Connection Service File is called pg_service.conf and is by client applications (such as psql or QGIS) generally found directly in the user directory. In Windows it is then found in the user’s application directory postgresql.pg_service.conf. And in Linux it is by default located directly in the user’s directory ~/.pg_service.conf.

But it doesn’t necessarily have to be there. The file can be anywhere on the system (or on a network drive) as long as you set the environment variable PGSERVICEFILE accordingly:

```
export PGSERVICEFILE=/home/dave/connectionfiles/pg_service.conf
```

Once you have done this, the client applications will search there first – and find it.

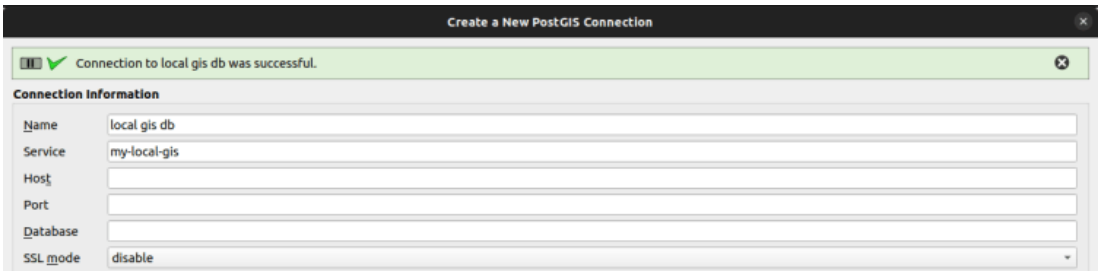
If the above are not set, there is also another environment variable PGSYSCONFDIR which is a folder which is searched for the file pg_service.conf.

Once you have this, the service name can be used in the client application. That means in psql it would look like this:

```
~$ psql service=my-local-gis
psql (14.11 (Ubuntu 14.11-0ubuntu0.22.04.1), server 14.5 (Debian 14.5-1.pgdg110+1))
```

SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

gis=#
And in QGIS like this:



If you then add a layer in QGIS, only the name of the service is written in the project file. Neither the connection parameters nor username/password are saved. In addition to the security aspect, this has various advantages, more on this below.
But you don’t have to pass all of these parameters to a service. If you only pass parts of them (e.g. without the database), then you have to pass them when the connection is called:

$psql "service=my-local-gis dbname=gis"
psql (14.11 (Ubuntu 14.11-0ubuntu0.22.04.1), server 14.5 (Debian 14.5-1.pgdg110+1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

gis=#
You can also override parameters. If you have a database gis configured in the service, but you want to connect the database web, you can specify the service and explicit the database:
$psql "service=my-local-gis dbname=web"
psql (14.11 (Ubuntu 14.11-0ubuntu0.22.04.1), server 14.5 (Debian 14.5-1.pgdg110+1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

web=#
Of course the same applies to QGIS.

And regarding the environment variables mentioned, you can also set a standard service.
export PGSERVICE=my-local-gis
Particularly pleasant in daily work with always the same database.
$ psql
psql (14.11 (Ubuntu 14.11-0ubuntu0.22.04.1), server 14.5 (Debian 14.5-1.pgdg110+1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

gis=#

The PostgreSQL backend passes the content of **OPTIONS** as keyword arguments to the connection constructor, allowing for more advanced control of driver behavior. All available parameters are described in detail in the PostgreSQL documentation.

Optimizing PostgreSQL's configuration

Django needs the following parameters for its database connections

- **client_encoding**: **'UTF8'**,
- **default_transaction_isolation**: **'read committed'** by default, or the value set in the connection options (see below),
- **timezone**:
    - when **USE_TZ** is **True**, **'UTC'** by default, or the **TIME_ZONE** value set for the connection,
    - when **USE_TZ** is **False**, the value of the global **TIME_ZONE** setting.

If these parameters already have the correct values, Django won't set them for every new connection, which improves performance slightly. You can configure them directly in **postgresql.conf** or more conveniently per database user with ALTER ROLE.
Django will work just fine without this optimization, but each new connection will do some additional queries to set these parameters.

Isolation level

Like PostgreSQL itself, Django defaults to the **READ COMMITTED** isolation level. If you need a higher isolation level such as **REPEATABLE READ** or **SERIALIZABLE**, set it in the **OPTIONS** part of your database configuration in **DATABASES**:

```
from django.db.backends.postgresql.psycopg_any import IsolationLevel

DATABASES = {
    # ...
    "OPTIONS": {
        "isolation_level": IsolationLevel.SERIALIZABLE,
    },
}
```

Note

Under higher isolation levels, your application should be prepared to handle exceptions raised on serialization failures. This option is designed for advanced uses.

Role

If you need to use a different role for database connections than the role use to establish the connection, set it in the **OPTIONS** part of your database configuration in **DATABASES**:

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        # ...
        "OPTIONS": {
            "assume_role": "my_application_role",
        },
    },
}
```

Server-side parameters binding

With psycopg 3.1.8+, Django defaults to the client-side binding cursors. If you want to use the server-side binding set it in the **OPTIONS** part of your database configuration in **DATABASES**

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        # ...
        "OPTIONS": {
            "server_side_binding": True,
        },
    },
}
```

This option is ignored with **psycopg2**.

Indexes for varchar and text columns

When specifying **db_index=True** on your model fields, Django typically outputs a single **CREATE INDEX** statement. However, if the database type for the field is either **varchar** or **text** (e.g., used by **CharField**, **FileField**, and **TextField**), then Django will create an additional index that uses an appropriate PostgreSQL operator class for the column. The extra index is necessary to correctly perform lookups that use the **LIKE** operator in their SQL, as is done with the **contains** and **startswith** lookup types.

Migration operation for adding extensions

If you need to add a PostgreSQL extension (like **hstore**, **postgis**, etc.) using a migration, use the **CreateExtension** operation.

Server-side cursors

When using **QuerySet.iterator()**, Django opens a server-side cursor. By default, PostgreSQL assumes that only the first 10% of the results of cursor queries will be fetched. The query planner spends less time planning the query and starts returning results faster, but this could diminish performance if more than 10% of the results are retrieved. PostgreSQL's assumptions on the number of rows retrieved for a cursor query is controlled with the cursor_tuple_fraction option.

Transaction pooling and server-side cursors

Using a connection pooler in transaction pooling mode (e.g. PgBouncer) requires disabling server-side cursors for that connection.

Server-side cursors are local to a connection and remain open at the end of a transaction when **AUTOCOMMIT** is **True**. A subsequent transaction may attempt to fetch more results from a server-side cursor. In transaction pooling mode, there's no guarantee that subsequent transactions will use the same connection. If a different connection is used, an error is raised when the transaction references the server-side cursor, because server-side cursors are only accessible in the connection in which they were created.
One solution is to disable server-side cursors for a connection in **DATABASES** by
setting **DISABLE_SERVER_SIDE_CURSORS** to **True**.

To benefit from server-side cursors in transaction pooling mode, you could set up another connection to the database in order to perform queries that use server-side cursors. This connection needs to either be directly to the database or to a connection pooler in session pooling mode.

Another option is to wrap each **QuerySet** using server-side cursors in an **atomic()** block, because it disables **autocommit** for the duration of the transaction. This way, the server-side cursor will only live for the duration of the transaction.

Manually-specifying values of auto-incrementing primary keys

Django uses PostgreSQL's identity columns to store auto-incrementing primary keys. An identity column is populated with values from a sequence that keeps track of the next available value. Manually assigning a value to an auto-incrementing field doesn't update the field's sequence, which might later cause a conflict. For example

```
>>> from django.contrib.auth.models import User
>>> User.objects.create(username="alice", pk=1)
<User: alice>
>>> # The sequence hasn't been updated; its next value is 1.
>>> User.objects.create(username="bob")
IntegrityError: duplicate key value violates unique constraint
"auth_user_pkey" DETAIL:  Key (id)=(1) already exists.
```

If you need to specify such values, reset the sequence afterward to avoid reusing a value that's already in the table. The **sqlsequencereset** management command generates the SQL statements to do that.

Test database templates

You can use the **TEST['TEMPLATE']** setting to specify a template (e.g. '**template0**') from which to create a test database.

Speeding up test execution with non-durable settings

You can speed up test execution times by configuring PostgreSQL to be non-durable.

Warning
This is dangerous: it will make your database more susceptible to data loss or corruption in the case of a server crash or power loss. Only use this on a development machine where you can easily restore the entire contents of all databases in the cluster.

1.3.2. SQLite

SQLite is a lightweight, embedded relational database management system (RDBMS) that's included with Python. Unlike PostgreSQL, it stores data in a single file, making it convenient for development and small to medium-sized applications. Here's a breakdown of SQLite and how to set it up in Django:

SQLite Advantages for Development

- **Simple Setup:** No separate installation is required; it comes bundled with Python.
- **Lightweight:** Uses minimal resources, making it ideal for development environments.
- **Zero-configuration:** Django automatically creates the database file if it doesn't exist.
- **Portable:** The database file is a single file, making it easy to move between development environments.

Setting up SQLite in Django

1. **Enable SQLite in Django settings.py:**

   - Open your project's settings.py file.
   - Locate the DATABASES dictionary. By default, Django is likely already configured for SQLite.

Python
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': 'mydatabase.db',  # You can choose any name for your database file
    }
}
```
Use code with caution.

content_copy

2. **Create Models (Optional):**
   - Define your data models in a separate app within your Django project.
   - These models represent the structure of your data in the database.
3. **Migrate (Optional):**
   - If you have existing models, run python manage.py makemigrations to create migration files.
   - Then, run python manage.py migrate to apply the migrations and create the database tables (if they don't exist).

Important Note

- While SQLite is excellent for development, it has limitations for large-scale applications due to its single-file nature and potential performance bottlenecks. Consider PostgreSQL or other database systems for production environments.

1.3.3. MySQL

MySQL is another popular open-source relational database management system (RDBMS) that's widely used for various applications. Here's a breakdown of what MySQL is and how to set it up with Django:

What is MySQL?

- **Open-Source and Feature-Rich:** Similar to PostgreSQL, MySQL is free to use and offers a vast range of features, including complex data types, user management, stored procedures, and more.
- **Scalable:** MySQL can handle large datasets and complex queries, making it suitable for medium to large-scale applications.
- **Wide Adoption:** Due to its popularity, MySQL has a large community and extensive documentation available online.

Setting up MySQL in Django

The Database is the essential component of the web application to store and organize the data. Whenever we develop an application/website, we need to choose a suitable database that makes it more interactive.
Django comes with a built-in SQLite database. However, we can use the various databases in Django. Below are the lists of databases that Django supports.

- PostgreSQL
- MariaDB
- MySQL
- Oracle

- SQLite

There are also numbers of database backends provided by third parties. Django middleware allows us to communicate with the database. In this tutorial, we will learn how we can connect the MySQL database to our Django application.

Prerequisites

- MySQL server 5.7+ must be installed
- Python 3.0+ must be installed

We assume that you have already installed the MySQL server on your local computer. If you haven't installed then download it from MySQL official website.

Important Considerations

- **Security:** Ensure you choose strong credentials for your database user and avoid storing sensitive information directly in your settings file. Consider environment variables for a more secure approach.
- **Character Encoding:** Verify that your MySQL database is configured to use UTF-8 encoding to avoid character set issues.

By following these steps, you can successfully set up MySQL as the backend database for your Django project. Remember that while MySQL is powerful, it may require more configuration compared to SQLite. Choose the database system that best suits your project's needs and complexity.

1.3.4. NoSQL

NoSQL (Not Only SQL) refers to a category of databases that differ fundamentally from traditional relational databases like MySQL or PostgreSQL. Here's a breakdown of NoSQL and its setup in Django (with some caveats):

Understanding NoSQL Databases:

- **Non-Relational:** Unlike relational databases that rely on tables with fixed structures, NoSQL databases offer more flexibility in data storage. They use various models like key-value pairs, documents, graphs, or columns.
- **Scalability:** NoSQL databases excel at handling massive datasets and high-volume writes due to their distributed nature.

- **Variety of Use Cases:** NoSQL databases are particularly valuable for applications dealing with unstructured or semi-structured data, real-time data processing, and big data analytics.

Setting Up NoSQL in Django:

While Django itself doesn't natively support NoSQL databases, there are third-party libraries and workarounds to achieve some level of integration:

1. **Third-Party Libraries:**
    - Libraries like `django-nonrel` or `django-mongodb-engine` allow you to connect Django to NoSQL databases like MongoDB. These libraries provide an abstraction layer for interacting with the NoSQL database using a Django-like syntax.

Important Considerations:

- **Limited Support:** Third-party libraries for NoSQL integration with Django might have limitations compared to using a relational database. They may require additional configuration and might not offer the full functionality you'd expect from a native Django ORM.
- **Project Suitability:** Carefully evaluate your project's needs. If your application relies heavily on features like complex joins or data integrity constraints typically handled by relational databases, NoSQL might not be the best choice.

Alternative Approach: Hybrid Solutions

In some cases, you might consider a hybrid approach where you use a relational database for core structured data and a NoSQL database for specific functionalities that benefit from its flexibility, like caching or storing large unstructured documents.

Choosing the Right Database:

The decision between a relational database and NoSQL for your Django project depends on your specific data model and application requirements. If you have a well-defined data schema with complex relationships and need strong data integrity, a relational database like MySQL or PostgreSQL is likely a better fit. If you prioritize scalability, flexibility for unstructured data, or real-time data processing, NoSQL could be a viable option, but consider the limitations of using it with Django.

1.3.5. Without a database

You can technically use Django without a database, but it's important to understand the limitations and trade-offs involved. Here's what you need to know:

What Django Offers (Without a Database):

- **Structure and Organization:** Even without a database, Django provides a framework for organizing your project. You can define models (which represent your data structure) and views (which handle user requests) even if the data isn't persisted in a database.
- **Templating Engine:** Django's powerful templating engine allows you to create dynamic HTML pages. You can use variables and logic within your templates to generate content.
- **URL Routing:** You can define URLs in your Django project and map them to specific views. This allows for a clean and organized way to handle user requests.

What's Missing (Without a Database):

- **Data Persistence:** The biggest limitation is the lack of data persistence. Any data you store or user interactions will be lost once the server restarts.
- **Features Reliant on Database:** Django functionalities like user authentication, sessions, and the admin interface heavily depend on a database backend. These features won't work without a database.
- **Scalability:** For any sizable application, managing data without a database becomes cumbersome and limits scalability.

When Might Using Django Without a Database Be Useful?

There are a few limited scenarios where using Django without a database might be a consideration:

- **Simple Static Websites:** For very basic websites with static content that doesn't require user interaction or data storage, Django can provide some structure and organization.
- **Proof of Concept or Prototypes:** If you're quickly prototyping an application to demonstrate functionality, you might temporarily use Django without a database to showcase basic features before integrating a database later.

Self-Check Sheet 1: Install python and Django

01. Which of the following is NOT a core file found in the root directory of a Django project?
Answer:
a) **init**.py
b) settings.py
c) urls.py
d) database.py

02. Django templates use what syntax to display dynamic data within HTML?
Answer:
a) Curly braces { }
b) Angle brackets < >
c) Double curly braces {{ }}**)**
d) Square brackets [ ]

03. What design pattern does Django follow for structuring web applications?
Answer:
a) MVC (Model View Controller)
b) MVB (Model View Builder)
c) MVT (Model View Template)
d) MVVM (Model-View-ViewModel)

04. Which of the following is a primary benefit of using a virtual environment for Django development?
Answer:
a) Improves website loading speed
b) Isolates project dependencies
c) Enables collaboration with non-programmers
d) Simplifies database configuration

05. In Django, what is the primary purpose of a model?
Answer:
a) Defines the URL structure for the application
b) Represents the data structure and interacts with the database
c) Handles user interaction and logic within a view
d) Contains the HTML templates used for displaying content

Answer Key 1: Install python and Django

01. Which of the following is NOT a core file found in the root directory of a Django project?
Answer: No Correct Answer

02. Django templates use what syntax to display dynamic data within HTML?
Answer: d) Square brackets [ ]

03. What design pattern does Django follow for structuring web applications?
Answer: d) MVVM (Model-View-ViewModel)

04. Which of the following is a primary benefit of using a virtual environment for Django development?
Answer: c) Enables collaboration with non-programmers

05. In Django, what is the primary purpose of a model?
Answer: c) Handles user interaction and logic within a view

Job Sheet-1.1: **Install python in Windows**

UoC Cover

OU-ICT-WADP-01- L4-V1: Enable Django Framework Environment

Working Procedure / Steps

1. Verify the system requirements for the latest version of Python (e.g., Windows 10/11, appropriate CPU and RAM).
2. Ensure administrative rights on the computer.
3. Back up any important data before proceeding.
4. Check for any pending Windows updates and install them if necessary.
5. Open a web browser and navigate to the official Python website
6. Click on the latest version compatible with Windows.
7. Download the Windows executable installer (e.g., python-3.x.x-amd64.exe).
8. Locate the downloaded installer in your downloads folder and double-click it to run.
9. In the installer window, check the box labeled “Add Python to PATH” to make Python accessible from the command line.
10. Click “Install Now” to proceed with the default installation settings.
11. Wait for the installation to complete, then click “Close.”
12. Open Command Prompt by pressing Win + R, typing cmd, and hitting Enter.
13. Type python --version or python3 --version and press Enter to verify that Python is correctly installed and to check the installed version.
14. Optionally, type python and press Enter to enter the Python interactive shell. Type print("Hello, World!") and press Enter to ensure Python is working correctly.

Specification Sheet-1.1: **Install python in Windows**

Project Overview

- **Objective:** To install Python on a Windows system, providing a foundation for web development using Django.

Technical Requirements

- **Operating System:** Windows 10 or 11
- **System Resources:** Sufficient CPU and RAM to meet Python's requirements
- **Python Installer:** The latest stable version of Python for Windows

Other Requirements

- **Administrative Privileges:** User must have administrative rights on the computer.
- **Internet Connection:** A stable internet connection is required to download the Python installer.
- **Backup:** It is recommended to create a backup of important data before proceeding with the installation.
- **Windows Updates:** Ensure the system is up-to-date with the latest Windows updates.

Job Sheet-1.2: Install Django in Windows

UoC Cover

OU-ICT-WADP-01- L4-V1: Enable Django Framework Environment

Working Procedure / Steps

1) Verify that Python is already installed on the Windows system.
2) If not, follow the Python installation procedure.
3) Ensure you have administrative rights on the computer.
4) Backup important data and check for any pending Windows updates.
5) Open Command Prompt by pressing Win + R, typing cmd, and pressing Enter.
6) Ensure pip (Python package installer) is updated to the latest version:
7) Install Django using pip
8) Wait for the installation to complete. pip will download and install Django and its dependencies.
9) Verify Installation:

Specification Sheet-1.2: Install Django in Windows

Technical Requirements

- Python: Python must be installed on the system.
- Pip: The Python package installer (pip) must be installed.
- Django: Django will be installed using pip.

Other Requirements

- Administrative Privileges: User must have administrative rights on the computer.
- Backup: It is recommended to create a backup of important data before proceeding.
- Windows Updates: Ensure the system is up-to-date with the latest Windows updates.

Learning Outcome 2: Start a project

Assessment Criteria	1. Working environment is explored 2. Project structure is explored 3. The development server is started
Conditions and Resources	1. Real or simulated workplace 2. CBLM 3. Handouts 4. Laptop 5. Multimedia Projector 6. Paper, Pen, Pencil, Eraser 7. Internet facilities 8. White board and marker 9. Audio Video Device
Contents	1. Working environment 2. Project structure a. __init__.py b. manage.py c. settings.py d. urls.py 3. Start development server
Activities/job/Task	1. Start Django Development Server
Training Methods	1. Discussion 2. Presentation 3. Demonstration 4. Guided Practice 5. Individual Practice 6. Project Work 7. Problem Solving 8. Brainstorming
Assessment Methods	Assessment methods may include but not limited to 1. Written Test 2. Demonstration 3. Oral Questioning 4. Portfolio

Learning Experience 2: Start a project

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Activities	Recourses/Special Instructions
1. Trainee will ask the instructor about the learning materials	1. Instructor will provide the learning materials 'Start a project'
2. Read the Information sheet and complete the Self Checks & Check answer sheets on "Define Tasks of the Project"	2. Read Information sheet 2: Define Tasks of the Project 3. Answer Self-check 2: Start a project 4. Check your answer with Answer key 2: Start a project
3. Read the Job/Task Sheet and Specification Sheet and perform job/Task	5. Job/Task Sheet and Specification Sheet Task Sheet-2: Start Django Development Server in Windows

Information Sheet 2: Start a project

Learning Objective:

After completion of this information sheet, the learners will be able to explain, define and interpret the following contents:

2.1. Working environment is explored
2.2 Project structure is explored
2.3 The development server is started

2.1. Working environment

Prerequisites:

- **Python:** Ensure you have Python 3.6 or later installed on your system. You can verify this by opening your terminal or command prompt and typing `python --version` (or `python3 --version` on some systems). If not installed, download it from the official Python website https://www.python.org/downloads/.
- **Text Editor or IDE:** Choose a code editor or Integrated Development Environment (IDE) of your preference. Popular options include Visual Studio Code, PyCharm, Sublime Text, or any editor you're comfortable with.

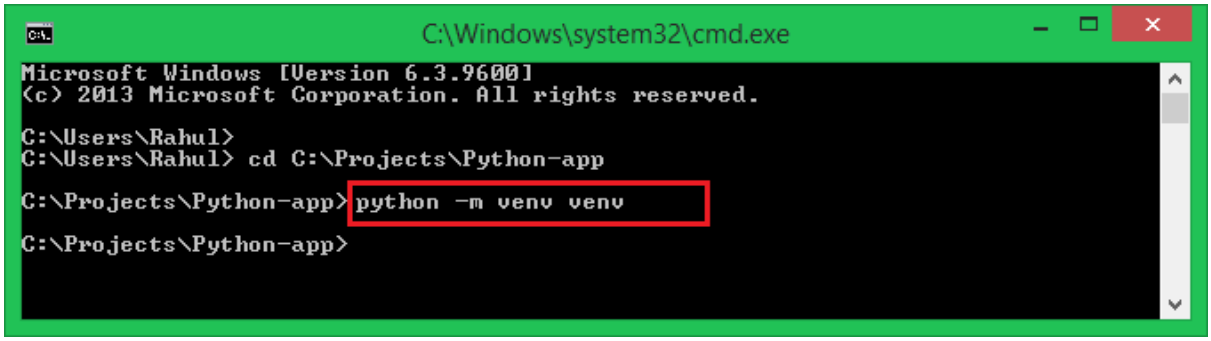
1. **Virtual Environment (Recommended):**

   A Virtual Environment provides an isolated environment for your application. That helps to maintain multiple Python applications on a single machine without any module conflicts. Once we create a activate the virtual environment, all the remaining activities are performed under that environment. It also helps to migrate the application to a new system.

   Create a Virtual Environment in Python

   A Python module **venv** is available by default in Python 3.3 and later versions. To create a virtual environment, cd to your project directory and run the following command to create a new virtual environment.

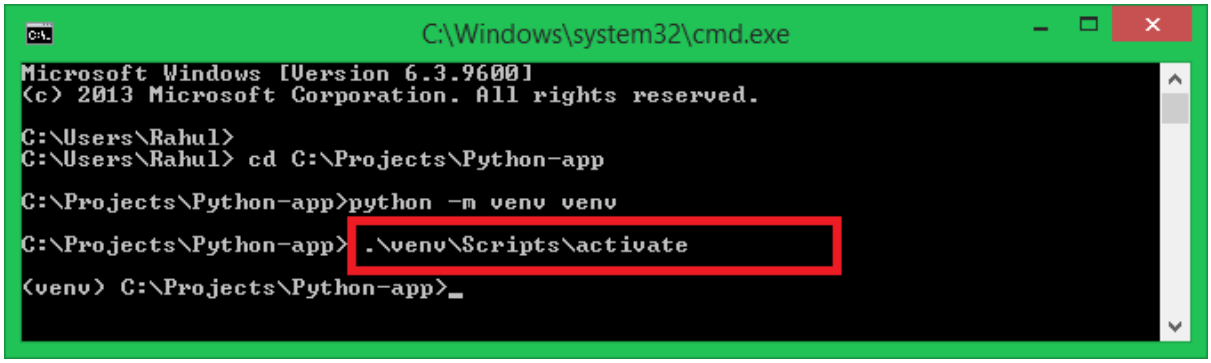
   1. **Create the Environment:** The following commands will create a new virtual environment under c:\Projects\Python-app\venv directory:
   2. cd c:\Projects\Python-app
   3. python3 -m venv venv



Creating a Python Virtual Environment

4. **Activate the Environment:** Now, we have a virtual environment, we need to activate it.
5. .\venv\Scripts\activate

After you activate the environment, the command prompt will be changed to show the virtual environment.



Activate Python Virtual Environment on Windows

Creating the requirements.txt **File (Optional)**

After you activate the virtual environment, you can add packages to it using pip. You can also create a description of your dependencies using pip.

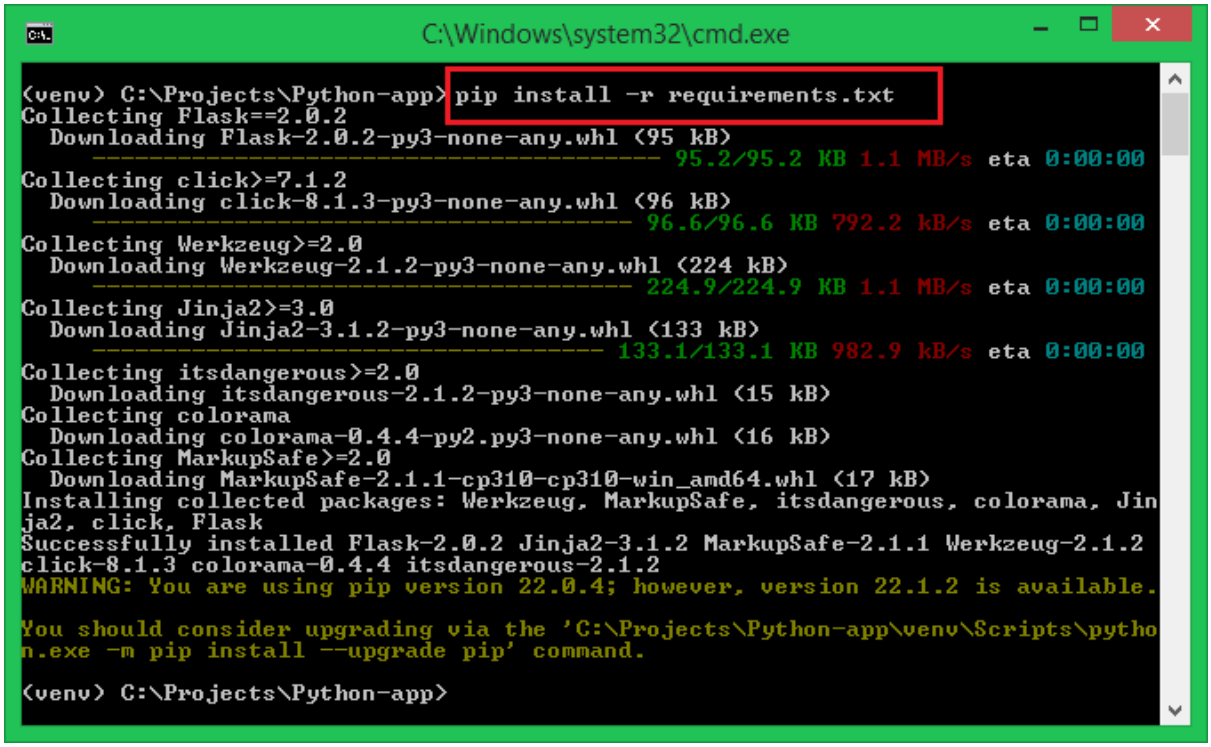
ADVERTISEMENT

Use the following command to create requirements.txt containing all the installed packages on your system.

pip freeze > requirements.txt

This file can be used by the other project collaborators to install or update Python modules on their system virtual environments using the following command.

pip install -r requirements.txt



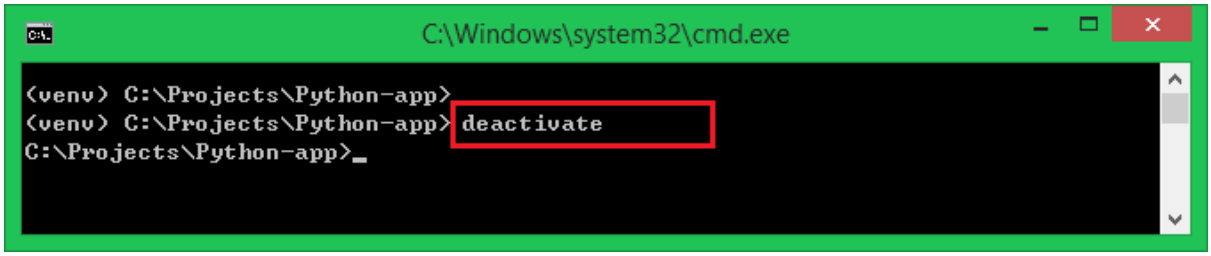
Installing Python modules from requirements.txt

Deactivate the Virtual Environment

You can close the virtual environment and return to normal system settings, by typing the deactivate command:

deactivate

After executing the above command, you'll notice that the command prompt returns to normal.



Deactivate python virtual environment

2. **Install Django:**

The Python ecosystem has a lot of web frameworks. One that has consistently been popular is the Django framework. It's popular for being robust, secure, and allows developers to develop projects fast and meet their deadlines. It is free and open-source, and it works on both Windows and *nix systems.

In this tutorial, you will learn how to install Django on Windows using pip. After that, you will verify the installation, create a project, and start a Django development server.

Prerequisites

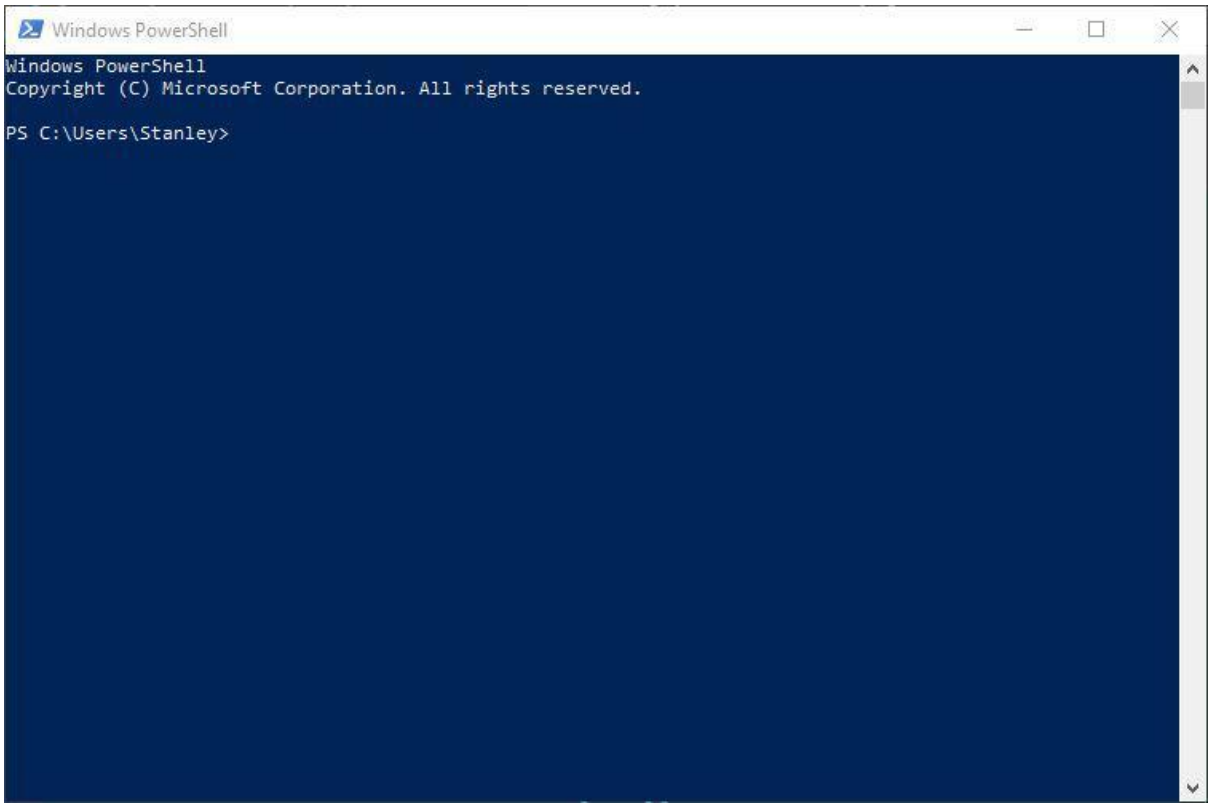
Before you install Django, you must make sure that Python is installed on your system.

The commands that you will run in this tutorial were tested on the **PowerShell** shell, but most commands should work on **Windows Command Prompt(CMD)** with a few exceptions. For a smooth experience, I would suggest you use PowerShell for this tutorial.

Step 1 — Opening PowerShell

First, you need to open PowerShell on your computer. You can do that by searching for PowerShell in the Windows search box or you can open the **Run** dialog box by holding the **Windows logo key** and **R**(WIN+R). Once the dialog is open, type powershell, and then click **OK**.

You should now have the PowerShell window opened.



Now that you have opened PowerShell on your computer, you'll verify the installation of Python in the next section.

Step 2 - Verifying Python Installation

Before you install Django, first, you need to make sure that you installed Python on your system.

To do that, type the following command in PowerShell prompt to verify the installation:

1. python -V

-V option logs the Python version installed on your system.

After running the command, you should see output like this:

PS C:\Users\Username> python -V

Python 3.9.7

At the time of writing, it is Python 3.9.7. You might have a different version from mine, and that's fine. As long as you see the Python version logged, Python is installed on your system.

Now that you've confirmed Python is installed on your system, you will upgrade pip.

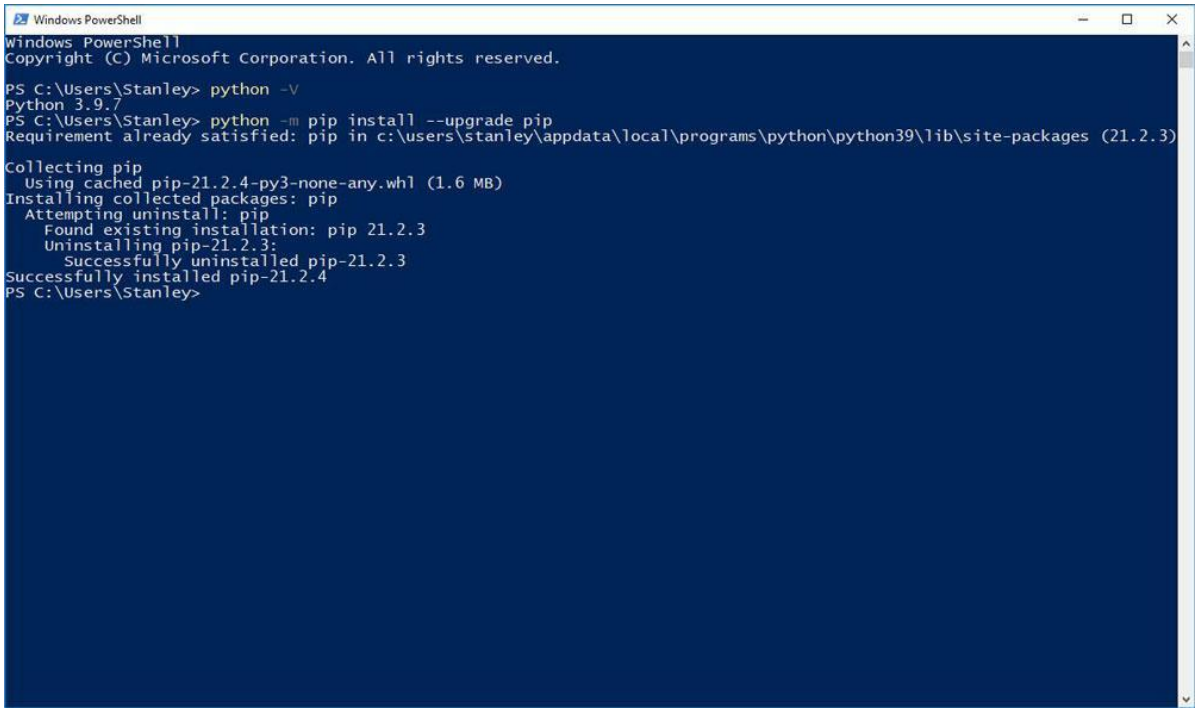
Step 3 - Upgrading Pip

Python comes with pip by default. But most of the time, it comes with an old version. it's always a good practice to upgrade pip to the latest version.

Enter the following command to upgrade pip on your system:

1. python -m pip install --upgrade pip
2.

You'll get output identical to the following screenshot showing you that the upgrade was a success:



Now you've upgraded pip, you'll create the project directory where you'll install Django.

Step 4 - Creating a Project Directory

In this section, you will create a directory that will contain your Django application. We will name it django_project since this tutorial is a demo. But in a real project, you can give the directory a suitable name, such as forum, blog, etc.

Change into your Desktop directory with the cd command:

1. cd Desktop
2.

Create the directory using the mkdir command:

1. mkdir django_project
2.

Move into the django_project directory using the cd command:

1. cd django_project
2.

Your prompt should now show you that you're in the django_project directory as shown in the following output:

1. PS C:\Users\Stanley\Desktop\django_project>

Now that you've created the working directory for your project, you'll create a virtual environment where you'll install Django.

Step 5 - Creating the Virtual Environment

In this step, you'll create a virtual environment for your project. A virtual environment is an isolated environment in Python where you can install the project dependencies without affecting other Python projects. This lets you create different projects that use different versions of Django.

If you don't use a virtual environment, your projects in your system will use the same Django version installed globally. This might look like a good thing until the latest version of Django comes out with breaking changes causing your projects to fail altogether.

To create a virtual environment, type the following command and wait for a few seconds:

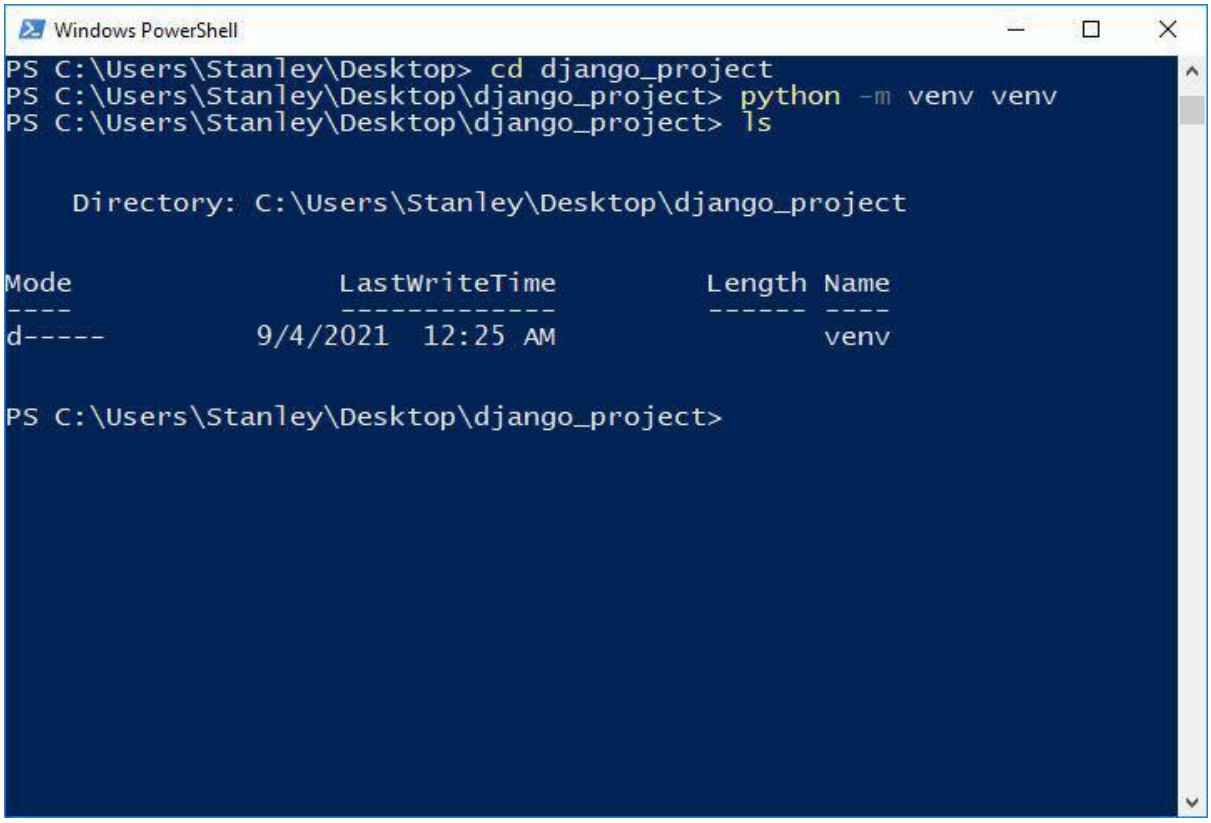
1. python -m venv venv

The command will create a directory called venv inside your project directory.

Next, confirm the venv directory has been created by listing the directory contents using the ls command:

1. ls

You should see the directory venv in the output as shown in the following screenshot:



Now you've created the virtual environment directory, you'll activate the environment.

Step 6 - Activating the Virtual Environment

In this section, you'll activate the virtual environment in your directory.

Run the following command to activate the virtual environment:

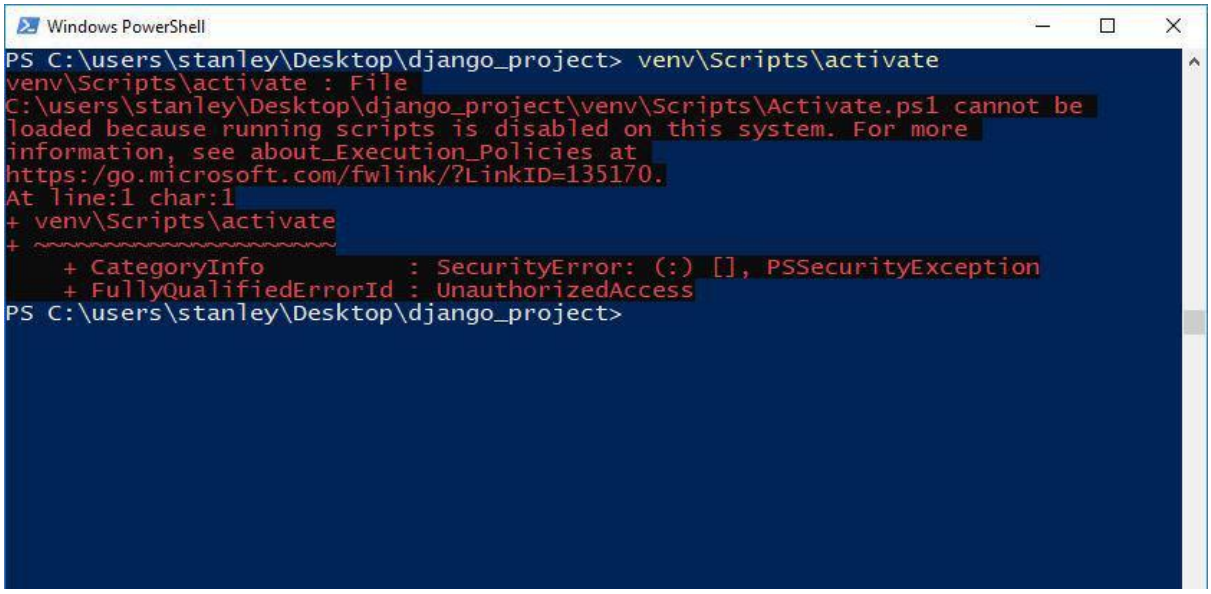
1. venv\Scripts\activate
2.

After you run the command, you will see

a (venv) at the beginning of the prompt. This shows that the virtual environment is activated:

(venv) PS C:\Users\Stanley\Desktop\django_project>

If you run into the error shown in the following screenshot on PowerShell when activating the virtual environment, for the sake of brevity, I described the reason and the solution in another post.



Now that you've activated the virtual environment for your project, the moment you've been waiting for is here. It's time to install Django!

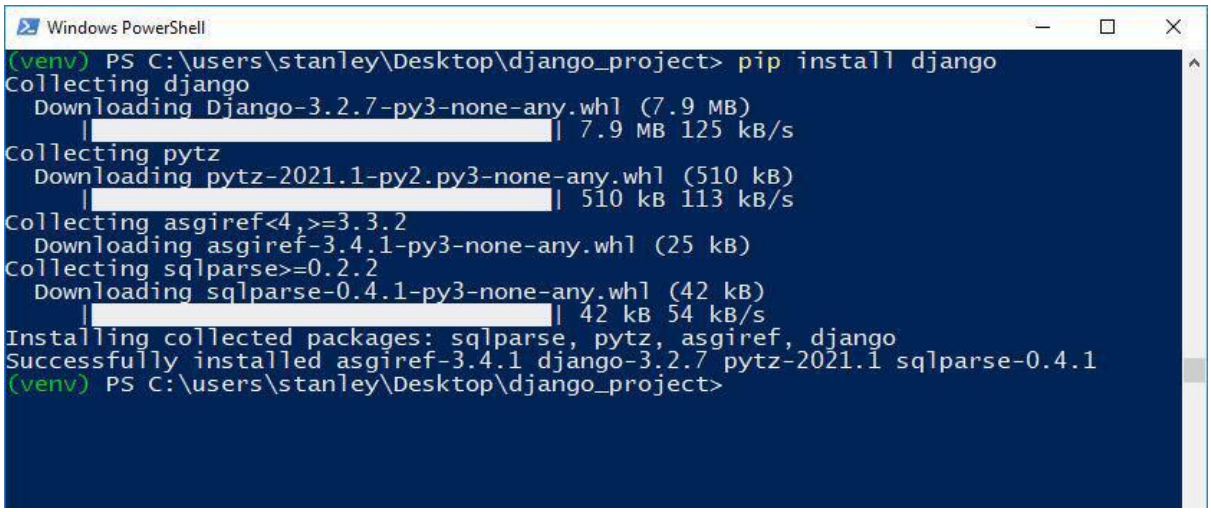
Step 7 - Installing Django

In this section, you will install Django on your system using pip.

Run the following command to install Django using pip install:

1. pip install django

The command will install the latest version of Django. You should see Django being downloaded as shown in the following screenshot:



If you want to install a different Django version, you can specify the version as follows:

1. pip install django==3.1

Once the installation finishes, you need to verify that Django has been installed. To do that, type the following command:

1. django-admin --version

You will get output showing you the Django version installed on your system:

(venv) PS C:\users\stanley\Desktop\django_project> django-admin --version

At the time of writing, the latest Django version is 3.2.7, and that's why my output shows that.

You've now installed Django on your system, great job! You'll begin to create a Django project.

2.2. Project structure

a. init.py

- **Purpose:** As mentioned earlier, an empty __init__.py file within a directory tells Python that the directory is a Python package. This allows you to import modules and sub-packages from that directory in your project or other Python applications.
- **Content:** While typically empty, you can add comments or initialization code within this file if necessary. However, avoid placing application logic or functionalities here, as that's what your app's __init__.py files are for (explained later).

b. manage.py

- **Purpose:** The manage.py script is your command-line interface for interacting with your Django project. It provides various commands to manage different aspects of your project's lifecycle.
- **Common Commands:** Here are some frequently used manage.py commands:
    - runserver: Starts the Django development server for testing purposes.
    - makemigrations: Creates migration files to track database schema changes.
    - migrate: Applies database migrations to update the database schema based on your models.
    - createsuperuser: Creates a superuser account for administrative access to the Django admin interface.
    - collectstatic: Collects static files (CSS, JavaScript, images) used by your application and places them in a designated location for serving.
    - There are many other commands available for specific tasks like running tests, creating app templates, and more. You can explore the full list using python manage.py help.

c. settings.py

- **Purpose:** The settings.py file is the heart of your Django project's configuration. It defines various settings that determine how your project behaves.
- **Key Settings Categories:**
    - **Database Settings:** Specify the database connection details, including database engine (e.g., django.db.backends.sqlite3), database name, username, password, and host.
    - **Secret Keys:** Include cryptographic keys used for various security purposes like session management and password hashing (keep these secret!).
    - **Installed Apps:** List all the Django apps that are part of your project, including both built-in Django apps and any custom apps you've developed.
    - **Template Settings:** Define the directories where your Django templates are located.
    - **Static Files Settings:** Specify the location where static files (CSS, JavaScript, images) are collected and the URL prefix used to serve them.
    - **Authentication Settings:** Configure authentication backends and define user models if applicable.
    - **Email Settings:** Configure email sending mechanisms if your project needs to send emails.
    - **Logging Settings:** Define how logging messages are handled within your project.
    - **Many Other Options:** Django offers a vast array of configuration options within settings.py to customize various aspects of your project's behavior. Refer to the official Django documentation https://docs.djangoproject.com/en/5.0/ref/settings/ for detailed explanations of all available settings.

Best Practices for settings.py

- **Organize Settings:** Consider using a logical structure for your settings, grouping related configurations together for better readability.
- **Environment-Specific Settings:** For production environments, it's recommended to keep sensitive information like database credentials and secret keys outside of your version control system (VCS) like Git. You can achieve this by creating separate settings files for development, testing, and production, and using environment variables to manage those settings.
- **Follow the Django Style Guide:** Adhere to the recommended coding style for Django projects to maintain consistency and readability. You can find the style guide here [invalid URL removed].

d. urls.py

- **Purpose:** The urls.py file in your project's root directory defines the central URL routing for your entire project. It acts like a traffic dispatcher, directing incoming user requests to the appropriate view functions within your application based on the URL pattern that matches the request.
- **Structure:**
    - This file typically imports the django.urls module and uses the path function (or url function in older Django versions) to define URL patterns.
    - Each URL pattern consists of a regular expression that matches a specific URL format and a view function that handles the request when the pattern matches.

App Directories

- Within your Django project's root directory, you'll likely create separate directories for each application within your project. These app directories house the specific code and functionalities of your application.
- **Common App Structure:** A standard app directory might contain the following files and subdirectories:
    - __init__.py: As mentioned earlier, an empty file marking the directory as a Python package.
    - models.py: Defines the data models for your application using Django's ORM (Object Relational Mapper). These models represent the structure of your data and how it relates to your database tables.
    - views.py: Contains Python functions called views that handle incoming user requests. These views typically interact with your models, templates, and other parts of your application logic to generate a response.
    - urls.py: Defines URL routing specifically for your application. It maps URLs relevant to your app's functionality to the corresponding view functions within views.py.
    - templates (subdirectory): Holds the HTML templates used to render dynamic content for your application's views. These templates can include variables and logic to display data or user input dynamically.
    - static (subdirectory): (Optional) Contains static files (CSS, JavaScript, images) specific to your application.
    - admin.py (Optional): If your application requires an admin interface for managing data, you can define the admin configuration in this file.

Additional Files

- wsgi.py: This file (often located in the root directory) serves as the entry point forWSGI (Web Server Gateway Interface) servers like uWSGI or Gunicorn. These servers are typically used in production environments to handle web requests more efficiently than Django's built-in development server.

Best Practices for Project Structure

- **Modular Design:** Organize your project into well-defined applications with clear separation of concerns. This promotes maintainability and reusability of code.
- **Meaningful Naming:** Use descriptive names for your files, directories, models, views, and URLs to enhance readability and understanding of your project structure.
- **Code Comments:** Include clear and concise comments within your code to explain the purpose of different functionalities and logic.

By following these guidelines, you can establish a clean and maintainable project structure that facilitates efficient development and collaboration in your Django projects.

2.3. Start Development Server

To start the development server in your Django project, follow these steps:

1. **Navigate to your project directory**

   Open your terminal or command prompt and use the `cd` command to change directory into the root directory of your Django project. For example, if your project is named `myblog`, you would type:

   Bash

   ```
   cd myblog
   ```

2. **Activate your virtual environment (if applicable)**

   If you're using a virtual environment to isolate your project dependencies, make sure it's activated before running the development server. The activation process typically involves running a script specific to your virtual environment manager. Refer to the documentation for your chosen virtual environment tool for specific activation instructions.

3. **Run the development server**

   Once you're in your project's root directory with the virtual environment activated (if applicable), execute the following command:

Bash

```
python manage.py runserver
```

This will start the Django development server, typically running on your local machine at `http://127.0.0.1:8000/`.

4. **Open the server in your web browser**

   After running the command, open a web browser and navigate to the URL `http://127.0.0.1:8000/`. You should see the Django welcome page, indicating that the development server is running successfully.

Additional Notes

- The development server is primarily intended for development and testing purposes. It's not recommended for use in production environments due to performance limitations and security considerations.
- By default, the development server listens on port 8000. If this port is already in use, you can specify a different port number when running the command:

  Bash

  ```
  python manage.py runserver <port_number>
  ```

  Replace `<port_number>` with the desired port number (e.g., `python manage.py runserver 8080`).

Self-Check Sheet 2: Start a project

1. Which of the following is NOT a benefit of using a virtual environment for Django development?

a) Isolates project dependencies
b) Improves website loading speed
c) Simplifies database configuration
d) Enables collaboration with non-programmers

2. The manage.py script in a Django project is used for:

a) Defining project configuration (settings)
b) Writing application logic (views)
c) Interacting with the project through various commands
d) Creating and managing HTML templates

3. The primary purpose of the init.py file within a Django project directory is:

a) To store frequently used functions
b) To mark the directory as a Python package
c) To define global variables for the project
d) To configure database connection details

4. In Django's URL routing (urls.py), the path function is used to:

a) Define the overall layout of your web pages
b) Connect URL patterns to specific model classes
c) Specify database queries for retrieving data
d) Manage user authentication and authorization

5. Which of the following files within a Django app directory typically defines the data structure and interacts with the database?

a) views.py
b) settings.py
c) urls.py
d) models.py

Answer Key 2: Start a project

1. Which of the following is NOT a benefit of using a virtual environment for Django development?

Answer: a) Isolates project dependencies

2. The manage.py script in a Django project is used for:

Answer: c) Interacting with the project through various commands (Correct)

3. The primary purpose of the init.py file within a Django project directory is:

Answer: b) To mark the directory as a Python package (Correct)

4. In Django's URL routing (urls.py), the path function is used to:

Answer: b) Connect URL patterns to specific model classes (Correct)

5. Which of the following files within a Django app directory typically defines the data structure and interacts with the database?

Answer: d) models.py (Correct)

Task Sheet-2: Start Django Development Server in Windows

UoC Cover

OU-ICT-WADP-01- L4-V1: Enable Django Framework Environment

Working Procedure / Steps

1. **Create a New Django Project:**
    - Choose a directory where you want to create your Django project.
    - Open a command prompt or terminal window in that directory.
    - Use the command to create a new project:
2. **Start the Development Server:**
    - Navigate to the project directory:
    - Start the development server:
    - The development server will start running on http://127.0.0.1:8000/. Open your web browser and visit this URL to see your Django project.

Review of Competency

Below is yourself assessment rating for module “Enabling Django Framework Environment”

Assessment of performance Criteria	Yes	No
Python is installed	☐	☐
Django is installed	☐	☐
Database is set up	☐	☐
Working environment is explored	☐	☐
Project structure is explored	☐	☐
The development server is started	☐	☐

I now feel ready to undertake my formal competency assessment.

Signed:

Date:

Development of CBLM

The Competency based Learning Material (CBLM) of ‘“Enabling Django Framework Environment’ (Occupation: Web Application Development with Python , Level-4) for National Skills Certificate is developed by NSDA with the assistance of SAMAHAR Consultants Ltd.in the month of June, 2024 under the contract number of package SD-9C dated 15th January 2024.

SL No.	Name and Address	Designation	Contact Number
1	Khan Mohammad Mahmud Hasan	Writer	Cell: 01714087897 Email: kmmhasan@gmail.com
2	A K M Mashuqur Rahman Mazumder	Editor	Cell: 01676323576 Email : mashuq.odelltech@odell.com.bd
3	Khan Mohammad Mahmud Hasan	Co-Ordinator	Cell: 01714087897 Email: kmmhasan@gmail.com
4	Md. Saif Uddin	Reviewer	Cell:01723004419 Email: engrbd.saif@gmail.com