my\_project/

├── venv/ # Virtual environment directory

├── app/ # Application code (e.g., main program)

│ ├── \_\_init\_\_.py # Marks this directory as a package

│ ├── models.py # Database models

│ ├── views.py # Views or routes (Flask, Django)

│ └── utils.py # Helper functions

├── requirements.txt # List of installed packages (dependencies)

├── run.py # Entry point of your application

└── README.md # Project documentation

# Using python:

1. Activate the Virtual Environment 🡪 source venv/bin/activate || .\venv\Scripts\activate (cmd) || .\venv\Scripts\Activate.ps1 (Powershell)
2. Install the Dependencies 🡪 pip install <>
3. Generate the requirements.txt File 🡪 file that lists all the installed dependencies.

pip freeze > requirements.txt 🡪 This will take a snapshot of all the installed Python packages and their versions in your virtual environment.

* In curr dir

1. Check the requirements.txt File 🡪 containing a list of installed dependencies.
2. Installing the dependency with req.txt 🡪 pip install -r requirements.txt

# Installing packages

1. pip install <package\_name>
2. pip install <package\_name>==<version>
3. pip install <package1> <package2> <package3> 🡪 multiple
4. pip install -r requirements.txt 🡪 replicating an environment
5. pip install git+https://github.com/user/repo.git 🡪 installing a github repo
6. pip install /path/to/package.tar.gz 🡪 Install a package from a local directory

# Listing Installed Packages

1. pip list
2. pip list --outdated
3. pip show <package\_name>
4. pip freeze > requirements.txt 🡪 Freezing Installed Packages
5. pip freeze --local > requirements.txt 🡪
6. pip install --upgrade <package\_name> 🡪 Upgrade a specific package to the latest version

# Upgrading Packages

1. pip install --upgrade <package\_name>

# Uninstall multiple packages:

1. pip uninstall <package1> <package2>

# Other Useful Commands:

1. pip –version 🡪 Check the current version of pip
2. pip config list 🡪 Check pip configuration
3. pip search <package\_name> 🡪 Search for a package on PyPI
4. pip check

Once you've **installed** a **⇨ package** using **pip**, you can import and use it in your project

**Importing** refers to bringing external code into your script so you can use its functionality.

1. **Module**: A single file containing Python code, which may define **functions, classes, and variables**

Example: math.py, os.py, requests.py

* It is just a single .py file.
* You can import specific functions, classes, or variables from it, or you can import the entire file.
* You import it by the filename (without the .py extension)
* import math\_utils

1. **Package**: A collection of modules in a directory, which usually contains an **\_\_init\_\_.py file** to indicate it's a package.

Example: numpy, pandas, flask

* It is a directory that contains multiple .py files (modules).
* It must have an \_\_init\_\_.py file, which can be empty or contain initialization code.
* A package can contain sub-packages.
* You import it by the package name (directory name) and then access specific modules within it.
* import my\_package.math\_utils

import pandas as pd 🡪 Importing a **package**

import math 🡪 Importing an entire **module**

from math import sqrt 🡪 Importing specific functions or variables **from a module**

import numpy as np 🡪 Renaming **a module** during import

# How to know????

1. pip show flask
2. cd /path/to/your/venv/lib/python3.9/site-packages/flask/
3. ls /path/to/your/venv/lib/python3.9/site-packages/flask/