Introduction to Coding World using Python

The videos introduce the fundamentals of programming with a focus on Python. The instructor highlights the tools used in the course, particularly VS Code and diagramming tools like Eraser and Excalidraw, to aid visual learning.

It explains programming as giving instructions to a computer, using an analogy of making chai (tea) to illustrate the process. The video breaks down programming into three components: gathering data, checking conditions, and executing steps.

The instructor also addresses misconceptions about coding, emphasizing that while starting with simple code can be easy, mastering it requires time and effort. The lecture concludes with the promise of translating the steps from the chai analogy into code in the next video, making programming seem more approachable.

Why Python ?

Python is widely used in robotics for several key reasons:

- **Simplicity and Readability:** Python's clear, intuitive syntax makes it easy to learn and write code quickly. This speeds up development and prototyping, crucial in a fast-evolving field like robotics.
- **Rich Ecosystem of Libraries:** Python boasts a vast collection of specialized libraries for tasks common in robotics, such as:
    - **Machine Learning/AI:** TensorFlow, PyTorch, enabling robots to learn and make intelligent decisions.
    - **Computer Vision:** OpenCV for processing visual data from cameras.
    - **Numerical Computing:** NumPy, SciPy for complex mathematical operations needed for robot kinematics and control.
    - **Robotics-Specific Libraries:** PyRobot, PyKDL, Pybotics, and compatibility with the **Robot Operating System (ROS)**, which is heavily used in robotics research and development.
- **Rapid Prototyping:** Its interpreted nature allows for quick testing and iteration of code, enabling developers to refine robot behaviors and algorithms efficiently.
- **Integration Capabilities:** Python can easily integrate with other languages like C++, allowing developers to use Python for high-level logic while leveraging faster, lower-level languages for performance-critical components.
- **Cross-Platform Compatibility:** Python runs on various operating systems, making it flexible for different robot hardware and development environments.

Then we installed python and wrote the following code to check version

```
import sys
print(sys.version)
```

Virtual Environments

"Getting everything on a Virtual Environment" means creating an isolated space for your Python project. Imagine it as a self-contained box where your project's Python interpreter and all its specific libraries (dependencies) live, separate from other Python projects on your computer.

Why Virtual Environments are Necessary:

1. **Dependency Management:** Different projects often need different versions of the same library. Without virtual environments, installing a new version for one project might break another. Virtual environments prevent these conflicts.
2. **Cleanliness:** They keep your global Python installation clean and free from project-specific clutter.
3. **Reproducibility:** You can easily share your project's exact dependencies with others, ensuring they can run your code without issues.

Simple Command Line Codes:

Let's say you have a file named requirements.txt with a list of libraries your project needs (e.g., numpy==1.26.4, pandas==2.2.2).

1. **Create a virtual environment:**

Bash

```
python -m venv my_project_env
```

(Replace my_project_env with your desired environment name.)

2. **Activate the virtual environment:**
    - **On Windows:**

Bash

```
my_project_env\Scripts\activate
```

    - **On macOS/Linux:**

Bash

```
source my_project_env/bin/activate
```

3. You'll see the environment's name in your terminal prompt (e.g., (my_project_env) your_username@your_computer).

4. **Install from requirements.txt:**

Bash

```
pip install -r requirements.txt
```

This command tells pip (Python's package installer) to install all the libraries listed in requirements.txt specifically into your active virtual environment.

5. Deactivate the virtual environment:

When you're done working on the project:

Bash

```
Deactivate
```

What to give to end user if he wills to use our Python Application ?

Giving someone just your code and requirements.txt isn't enough for them to run your app easily. They'd need Python, a virtual environment setup, and to know how to run your script.

To make it simple for them, you usually give them one of these:

---

1. A Standalone Executable

This is the easiest for non-developers. Tools like **PyInstaller** bundle your app, Python, and all its parts into one file (like an .exe on Windows). They just click and run!

---

2. A Web Application

If your app runs in a browser, you host it on a server. Users then just visit a website URL. No installation for them at all.

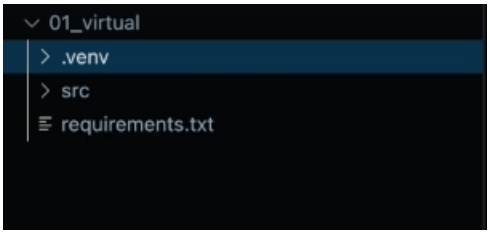
---

3. A Docker Container

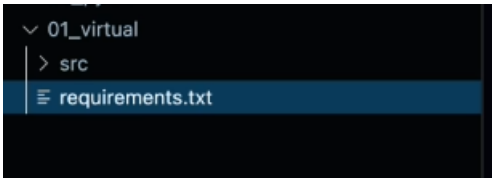
For more technical users, you can package your app into a **Docker** container. This ensures your app runs exactly the same everywhere, but the user needs Docker installed.

For most non-technical users, a **standalone executable** is the best choice.

We actually then delete the venv and ship the remaining content of the folder. Like in the images shown.

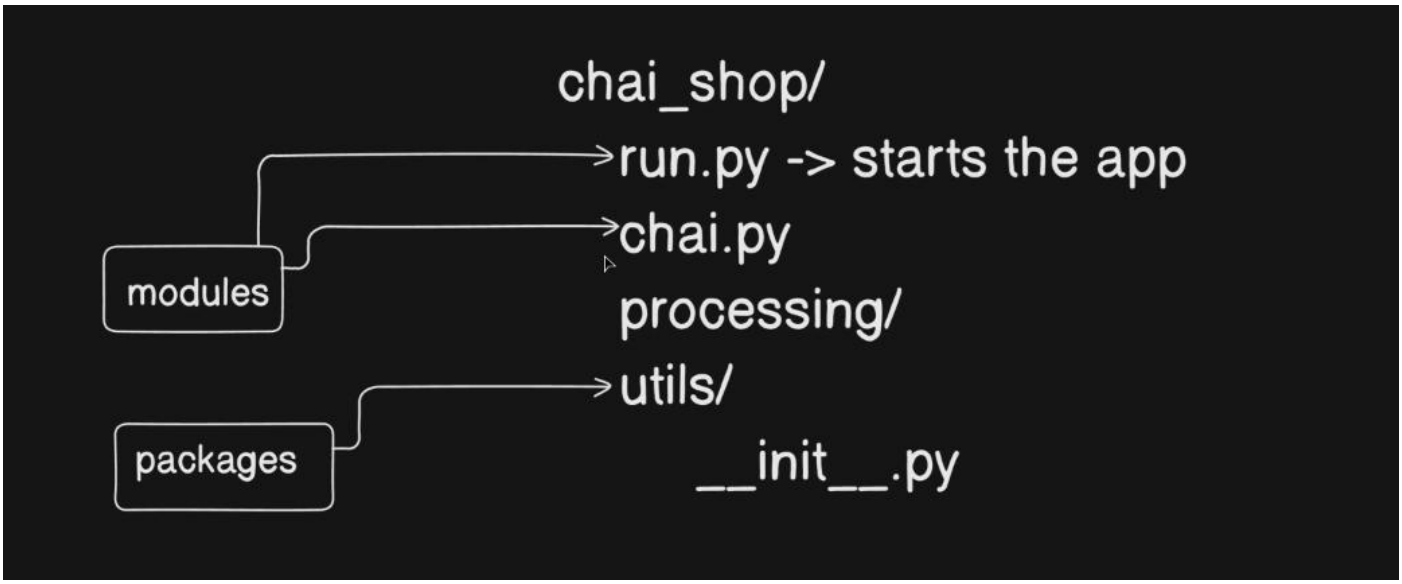


Folder structure of 01_virtual before deleting Venv.



Folder structure of 01_virtual after deleting Venv. Only requirements.txt and src containing the code exists.

Preferred Python Project Structure



PEP8 and Zen of Python

PEP 8 and the Zen of Python, emphasizes the importance of writing clean and maintainable code. PEP 8 serves as a style guideline, recommending practices like using four spaces for indentation, meaningful naming for functions and classes, and the use of code formatters. The Zen of Python outlines principles for writing Pythonic code, prioritizing simplicity and clarity. The speaker encourages learners to adopt these guidelines to create code that is simple, readable, and understandable by others.