🐍 Python Development Guide 🐍 (for Linux Users)

Practical Tools and Best Practices for Beginners

🐍 0. What is Python?

Python is a high-level, general-purpose programming language known for its **simplicity, readability**, and **versatility**. It's used in everything from **web development** and **automation** to **data science**, **machine learning**, **scripting**, and **software prototyping**.

- **Readable syntax**: Code looks like plain English, which makes it beginner-friendly.
- **Dynamically typed**: You don't have to declare variable types.
- **Interpreted**: Code runs line-by-line, which is great for testing and debugging.
- **Extensive standard library**: Comes with tons of useful built-in modules.
- **Massive community & ecosystem**: Thousands of third-party packages via PyPI.

Whether you're automating a boring task or building a complex web app, Python gets the job done with flexible code and a clear structure.

Python is used professionally in areas like:

- 🧠 **Data Science & Machine Learning** – Libraries like `pandas`, `scikit-learn`, and `TensorFlow` make Python a top choice for AI and analytics.
- 🌐 **Web Development** – Frameworks like `Flask` and `Django` are used to build scalable, production-grade web apps.
- 🤖 **Automation & Scripting** – Ideal for writing scripts to automate tasks, manage systems, or glue software together.
- 🧬 **Scientific Computing** – Used by NASA, biotech firms, and researchers for simulations, analysis, and prototyping.
- 🔐 **Cybersecurity**, **Finance**, **Game Development**, and more — Python shows up everywhere.

Python's simplicity makes it accessible for beginners and great for getting started with computer programming, but it's also powerful enough to build serious, scalable, and complex systems, making it a favorite for professionals.

📜 1. IDEs (Integrated Development Environments)

An **IDE** is a software application that helps you write code more efficiently. It combines multiple tools into one interface: a code editor, debugger, terminal, file explorer, and more.

Think of an IDE as your workshop. It gives you everything you need to build software in one place.

🔧 Common IDEs for Python:

- **VS Code**: Lightweight, customizable, and supports many programming languages via extensions.
- **Jupyter Notebook**: Popular for data science and experimentation.

✅ Why use an IDE?

- **Syntax highlighting:** Automatically colors your code so you can read it more easily.
- **Auto-complete:** Helps you write code faster by suggesting completions.
- **Error checking (linting):** Highlights potential mistakes as you type.
- **Debugging tools:** Let you pause your code mid-run to examine variables.
- **Integrated terminal and version control (Git):** No need to leave your IDE to run commands or commit code.

> 💡**Beginner advice:** Start with **VS Code** with the official Python extension if you're unsure. It has everything you need and plenty of tutorials. **Jupyter Notebook** is also a good option.
>
> It is also worth noting that you don't necessarily **NEED** an IDE to write Python code, you can write Python code in any plain text editor like nano or vim, which come pre-Installed on most Linux distros. or on your terminal. But IDEs generally offer a better user experience.

✅ Terminal / Shell:

```
python3
```

This opens the **interactive Python prompt**. Good for testing small code snippets.

🐍 2. Interpreters

A **Python interpreter** is the program that reads and runs your Python code. On Linux, this is usually the `python3` executable, which often comes pre-installed for system tasks.

Never remove or overwrite this pre-installed Python interpreter. Removing or overwriting system Python can **break core tools** (like apt, add-apt-repository, or graphical utilities).

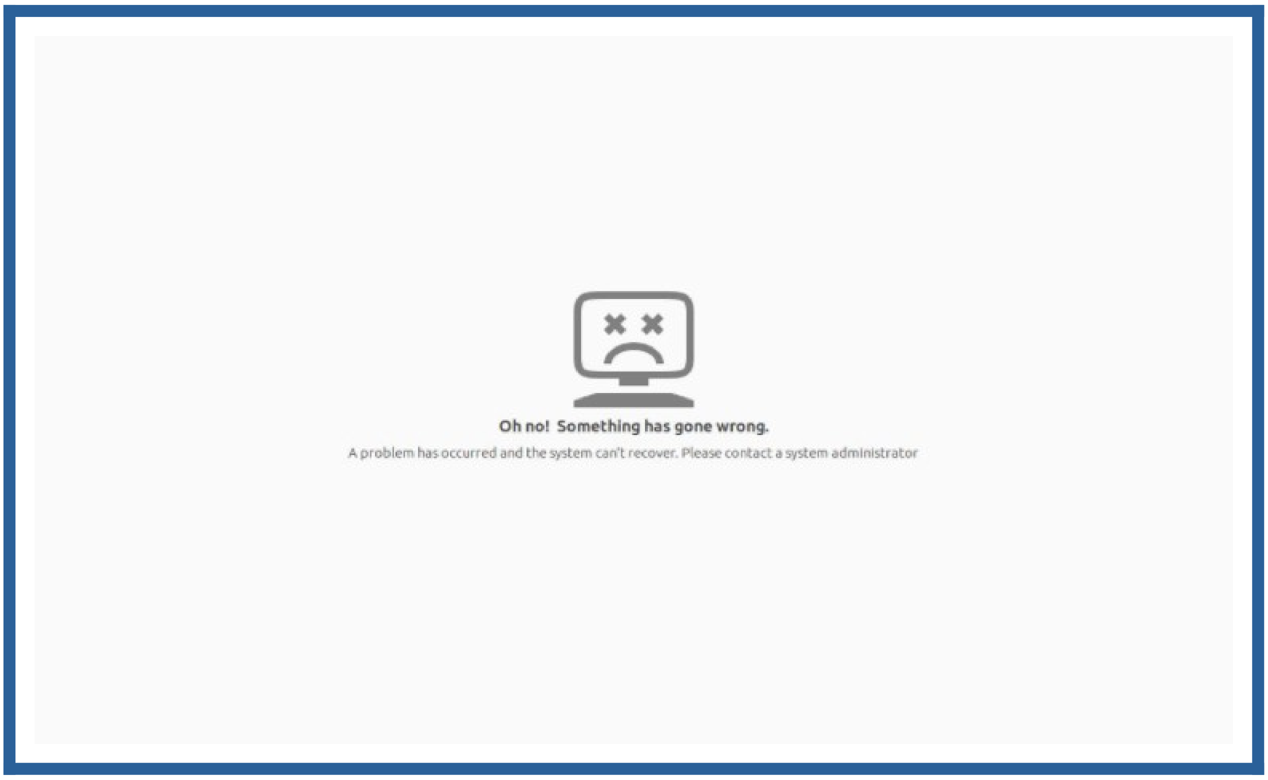
As a Python developer, you'll still need to install and manage python packages. Instead of using the system Python directly, the best practice is to create a **virtual environment**. This gives you an isolated space to work in, where you can install and manage packages without affecting the system.

🔄 Check if Python is installed:

```
python3 --version
which python3
```

The first command shows the installed version. The second shows the path to the interpreter that is currently being used.

> 🚨 Important: Linux often comes with Python pre-installed for system tasks. Never remove or overwrite it!
>
> Or else you might kill your OS:



Oh no! Something has gone wrong.

A problem has occurred and the system can't recover. Please contact a system administrator

💽 3. Virtual Environments (venv)

A **virtual environment** is a self-contained folder that has its own Python interpreter and libraries.

Why this matters:

- Prevents conflicts between projects that need different versions of the same library.
- Keeps your system Python clean and untouched.
- Lets each project manage its own dependencies independently.

⚒️ How to create and use one:

1. **Install venv (if not already):**

```
sudo apt install python3-venv
```

2. **Create the environment (inside a project folder):**

```
cd your/project/folder/path
python3 -m venv venv
```

(This creates a `venv/` folder with its own Python and pip.)

3. **Activate the environment:**

```
source venv/bin/activate
```

Now your shell prompt will show `(venv)` indicating you're inside the virtual environment.

4. **Deactivate when done (after you are done working on your project):**

```
deactivate
```

```
💡 Best practice: Use one virtual environment per project.
```

📃 4. `requirements.txt`

`requirements.txt` is a plain text file listing all the Python libraries your project uses, including their versions.

This lets you:

- Reproduce the same environment later.
- Easily share your project setup with others.
- Deploy the project on another machine.

✅ Create it:

```
pip freeze > requirements.txt
```

This saves all currently installed packages in the virtual environment.

📥 Install from it:

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

This installs all packages listed in the file.

> ℹ️ Tip: Always regenerate this file after changing dependencies.

🆚 5. `apt` vs `pip` on Linux

On Linux, you have two main tools for installing software relevant to Python development:

Tool	Purpose	Scope
`apt`	Installs system-level packages	For the whole OS
`pip`	Installs Python libraries	For Python (usually inside a virtual environment)

▶️ `apt`:

- The **Advanced Package Tool** for system-wide software.
- Installs Debian/Ubuntu system packages: not just Python, but editors, compilers, etc.
- Installs to locations shared by the whole OS.

▶️ `pip`:

- Python's package manager.
- Installs **Python-only** packages from the Python Package Index (PyPI).
- Installs packages **within your virtual environment** (or globally if not careful).

> 💡 Best practice:
>
> Use `apt` for system software **ONLY**; Always use `pip` (inside an activated venv) to install python libraries for your project.

🔄 6. Recommended Linux Workflow for Python Projects

```
# Step 1: Create a project folder (any folder will do)

# Step 2: Create the virtual environment (inside of the project folder)
```

```
cd your/project/folder/path
python3 -m venv venv
```

```
# Step 3: Activate the virtual environment
```

```
source venv/bin/activate
```

```
# Step 4: Install needed packages
```

```
pip install requests
```

```
# Step 5: Save dependencies
```

```
pip freeze > requirements.txt
```

```
# Step 6: Create python files to write your code (inside of your project folder)
```

```
python app.py
```

```
# Step 7: Exit the environment
```

```
deactivate
```

```
This keeps everything clean, reproducible, and manageable. Your system’s python
interpreter stays intact, your projects are isolated, and collaboration becomes
easier.
```
