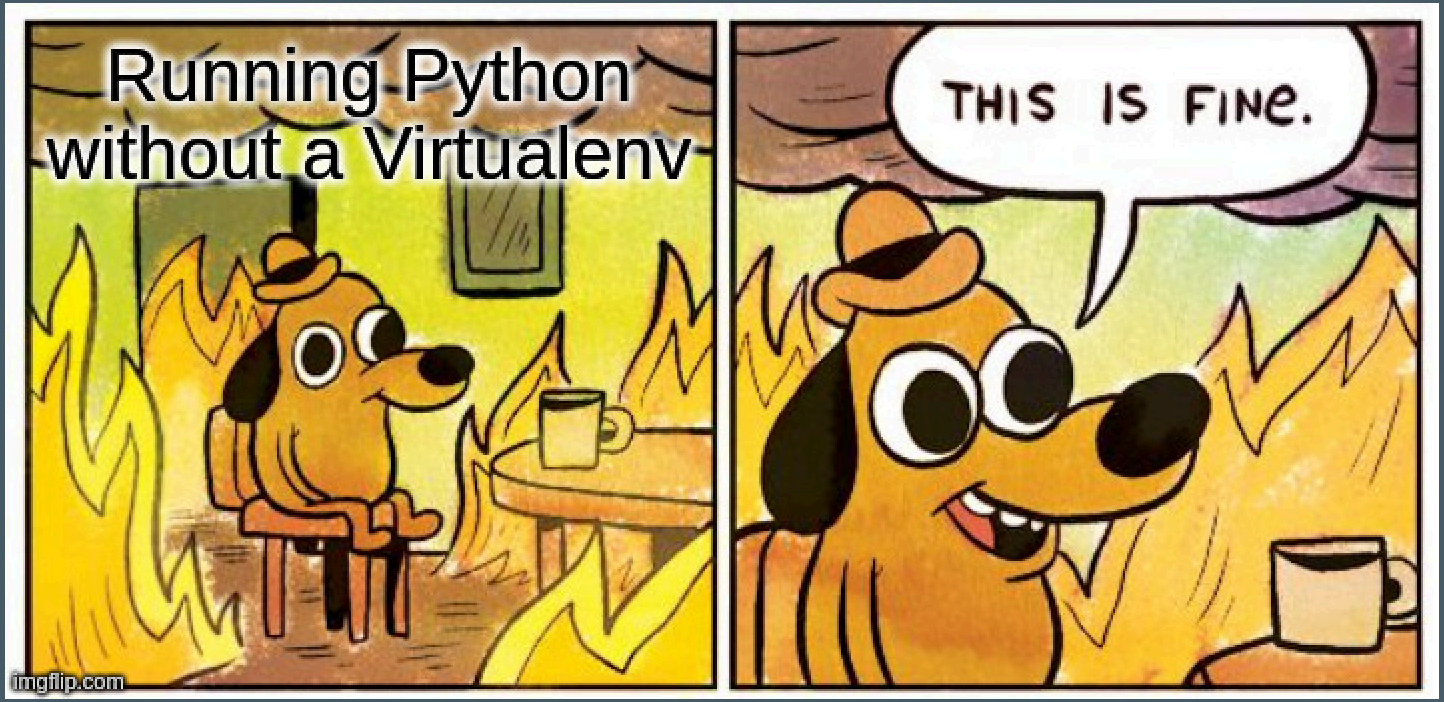
🐍 Python Project Setup

Virtual Environments, Modules, Packages & Linters

Why is this funny?



Virtual Environments

- Isolate dependencies per project
- Avoid conflicts between packages
- Standard tool: `venv`

```
python3 -m venv .venv
source .venv/bin/activate  # macOS/Linux
.venv\Scripts\activate    # Windows
```

Virtual Environments (continued)

- Deactivate with:

```
deactivate
```

- Add `.venv` to `.gitignore`
- Store dependencies:

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

pyproject.toml

- A standardized configuration file for Python projects
- Introduced by PEP 518
- Used by tools like `poetry`, `black`, `pytest`, `isort`
- Replaces tool-specific config files like `setup.cfg`, `tox.ini`

Example:

```toml
[tool.poetry]
name = "my_project"
version = "0.1.0"
description = "A sample Python project"
authors = ["Your Name <you@example.com>"]

[tool.black]
line-length = 88

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"
```

Alternative: `uv`

- A faster drop-in replacement for pip, venv, and pip-tools
- Built in Rust, very fast dependency resolution

```
uv venv
source .venv/bin/activate
uv pip install numpy pandas
uv pip freeze > requirements.txt
```

Poetry: Project & Dependency Manager

- Handles virtual environments and packaging
- Uses `pyproject.toml` for configuration

```
poetry new my_project
cd my_project
poetry add requests
poetry run python script.py
```

- Use `poetry install` to install all dependencies
- Recommended for packaged projects

Modules and Packages

- **Module** = one `.py` file
- **Package** = folder with `__init__.py`
- Allows organization and reuse of code

```python
# my_module.py
def greet(name):
    return f"Hello, {name}"
```

```python
# usage
from my_module import greet
greet("Alice")
```

Project Structure (Minimal)

```
my_project/
├── src/
│   └── my_package/
│       ├── __init__.py
│       └── core.py
├── tests/
│   └── test_core.py
├── requirements.txt
└── README.md
```

- Source code lives in `src/`
- Tests in `tests/`

Importing from `src/`

To make the package importable:

```
export PYTHONPATH=src     # macOS/Linux
set PYTHONPATH=src        # Windows
```

Or use editable installs:

```
pip install -e src/
```

Finally, the **best** option. Put this in your `pyproject.toml`:

```
[tool.pytest.ini_options]
pythonpath = "src"
```

Running Tests with `pytest`

- Popular test runner for Python
- Automatically finds files like `test_*.py`
- Simple syntax, powerful features

```
pip install pytest
pytest          # Run all tests
pytest tests/   # Run tests in tests/ directory
```

Example test:

```python
# tests/test_core.py
def test_add():
    from my_package.core import add
    assert add(2, 3) == 5
```

Linters

- Help keep your code clean and consistent
- Popular tools:
    - `flake8`
    - `pylint`
    - `black` (auto-formatter)

```
flake8 src/
pylint src/
black src/
```

Summary

- Use `venv` (or `uv` or `poetry`) to manage dependencies
- Structure your project with `src/`, `tests/`, and `requirements.txt`
- Organize code using modules and packages
- Use linters and formatters to write clean code
- Use `pytest` to test your code automatically