Python CLI Interpreter and -m tools

START HERE

`python3 <script.py> [args]`
Run a script

`python3 -m venv .venv`
Create a virtualenv

`python3`
Start the REPL

`python3 -m pip install <pkg>`
Install a package

`python3 -c "<code>"`
Run a one-liner

`python3 -i <script.py>`
Run, then inspect

`python3 -m <module>`
Run a library module

`python3 -m http.server`
Serve this folder

python3 <file> script, -m or -c → `compile` → **Bytecode** cached in __pycache__ → `run` → **Interpreter** sys.path finds imports → `exit` → **Exit code** 0 = success

CONCEPTS

python3 -m <module> runs a module from the import path, so tools like venv, pip and pytest always match the interpreter you called.

On Debian and Ubuntu use python3; plain python may not exist. Inside an activated venv, python works.

Options before the script go to Python; everything after the script name goes to the script in sys.argv.

RUN CODE

`python3 <script.py> <args>`
Run a script

`python3 -c "<code>"`
Run code from a string

`python3 -m <module> <args>`
Run a module as a script

`python3 -V`
Print the version

INTERACTIVE

`python3`
Start the REPL

`python3 -i <script.py>`
Run script, then stay in the REPL

`_`
Last result in the REPL

`help(<obj>)`
Show docs for an object

`dir(<obj>)`
List attributes of an object

`Ctrl+D`
Leave the REPL (Ctrl+Z Enter on Windows)

`PYTHONSTARTUP=<file>`
Run a file at each REPL start

INTERACTIVE (cont.)

`F2`
History browser in the REPL (3.13+)

`F3`
Paste mode in the REPL (3.13+)

VIRTUAL ENVS AND PACKAGES

`python3 -m venv .venv`
Create a virtual environment

`source .venv/bin/activate`
Activate it (Linux, macOS)

`.venv\Scripts\activate`
Activate it (Windows)

`deactivate`
Leave the environment

`.venv/bin/python <script.py>`
Use a venv without activating

`python3 -m venv --clear <dir>`
Empty and recreate an environment

`python3 -m pip install <pkg>`
Install a package

`python3 -m pip install -r requirements.txt`
Install from a list

`python3 -m pip install -e .`
Editable install of this project

DEV SERVERS AND TOOLS

`python3 -m http.server`
Serve current dir on port 8000

`python3 -m http.server <port>`
Serve on another port

`python3 -m http.server -b 127.0.0.1`
Listen on localhost only

`python3 -m json.tool <file>`
Validate and pretty-print JSON

TEST AND DEBUG

`python3 -m pdb <script.py>`
Debug a script from the first line

`python3 -m pdb -c continue <script.py>`
Debug, stopping on a crash

`breakpoint()`
Drop into the debugger from code

`n / s / c / q`
pdb: next, step, continue, quit

`l / ll / w`
pdb: list code, whole function, stack

`python3 -m unittest`
Discover and run unittest tests

`python3 -X faulthandler <script.py>`
Dump traceback on a crash

`python3 -X dev <script.py>`
Development mode: extra checks

MEASURE

`python3 -m timeit "<stmt>"`
Time a small statement

`python3 -m timeit -s "<setup>" "<stmt>"`
Time with a setup step

`python3 -m timeit -n <n> -r <r> "<stmt>"`
Set loops and repeats

`python3 -m cProfile -s cumtime <script.py>`
Profile, sorted by cumulative time

`python3 -X importtime <script.py>`
Show time spent on each import

FLAGS AND BEHAVIOUR

`python3 -u <script.py>`
Unbuffered stdout and stderr

FLAGS AND BEHAVIOUR (cont.)

`python3 -O <script.py>`
Strip assert statements

`python3 -I <script.py>`
Isolated mode: no env vars, user site or cwd

`python3 --help-env`
List environment variables

ENVIRONMENT VARIABLES

`PYTHONPATH=<dir>[:<dir>]`
Extra module search directories

`PYTHONUNBUFFERED=1`
Same as -u, good for logs and containers

Where imports are found

Order	Source
1	Script dir (or cwd with -c / -m)
2	PYTHONPATH entries
3	Standard library
4	site-packages

RECIPES

Fresh project environment

```
python3 -m venv .venv
source .venv/bin/activate
python -m pip install -r
requirements.txt
python -m pytest
```

Share a folder on the LAN

```
cd ~/Public
python3 -m http.server 8080
# then open http://<your-
ip>:8080
```

Pretty-print an API response

```
curl -s <url> | python3 -m
json.tool
# keep non-ASCII text readable
curl -s <url> | python3 -m
json.tool --no-ensure-ascii
```

Debug, profile, time

```
python3 -m pdb -c continue
app.py
python3 -m cProfile -s cumtime
app.py | head -20
python3 -m timeit -s "import
json" "json.dumps({'a': 1})"
```