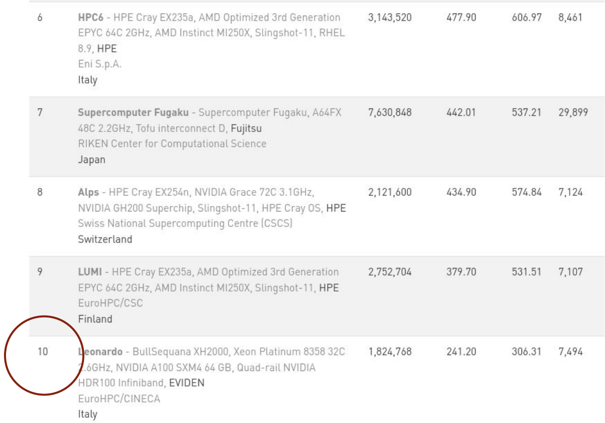
# Welcome to CINECA!

CINECA

Bologna - February 2026

# What is CINECA



|  |  |  |  |  |  |
|---|---|---|---|---|---|
| 6 | **HPC6** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE<br>Eni S.p.A.<br>Italy | 3,143,520 | 477.90 | 606.97 | 8,461 |
| 7 | **Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu<br>RIKEN Center for Computational Science<br>Japan | 7,630,848 | 442.01 | 537.21 | 29,899 |
| 8 | **Alps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cray OS, HPE<br>Swiss National Supercomputing Centre (CSCS)<br>Switzerland | 2,121,600 | 434.90 | 574.84 | 7,124 |
| 9 | **LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE<br>EuroHPC/CSC<br>Finland | 2,752,704 | 379.70 | 531.51 | 7,107 |
| 10 | **Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN<br>EuroHPC/CINECA<br>Italy | 1,824,768 | 241.20 | 306.31 | 7,494 |

# Introduction to Python

## The very first steps

Leonardo Salicari[✦] and Marco Celoria[♣]

✦ l.salicari@cineca.it ♣ m.celoria@cineca.it

Bologna - February 2026

# How to read these slides

- □ = Reference, source (if not present, source can be embedded in text or images)
- 🔍 = Advanced topics
- `whoami` = terminal (Powershell/bash/etc) commands
- `print("Hello World!")` = Python statements
- `print("Ciao!")` = File with Python statements

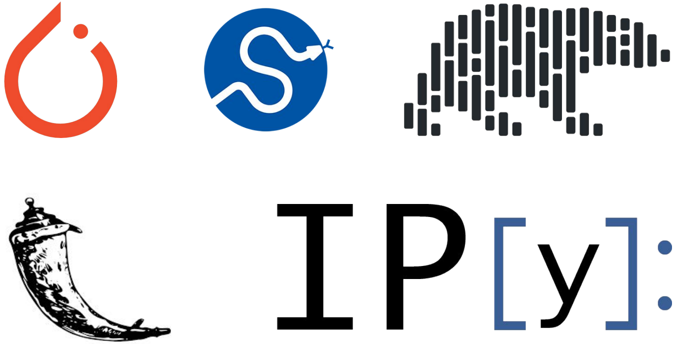
# The 5 Ws

- **What** Python is an interpreted high-level programming language for general-purpose programming
- **When**
  - First release in 1991
  - Python 2 released in 2000 (now EOL)
  - **Python 3** released in 2008
- **Who** Firstly developed by Guido van Rossum, now managed by the Python Software Foundation (Steering Council and through PEPs) 🔍

```
import this
# The Zen of Python, by Tim Peters
#
# Beautiful is better than ugly.
# Explicit is better than implicit.
# ...
print("Hello World!")
# Hello World!
```

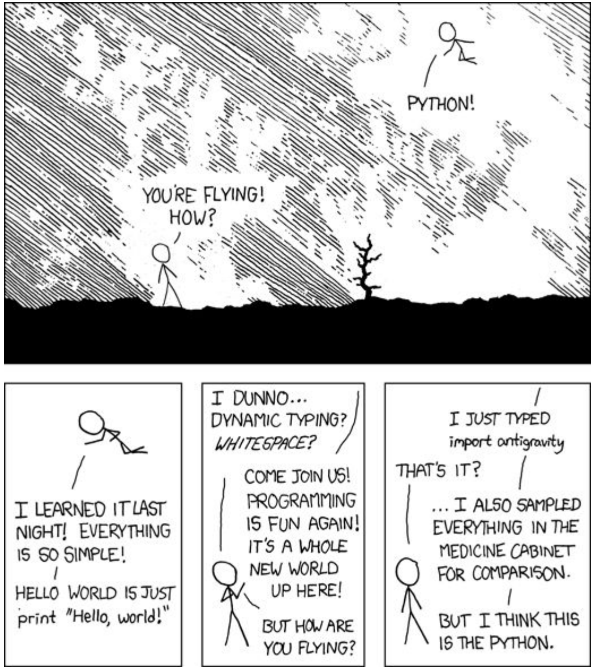
# The 5 Ws

- **Where**
  - Scientific computing
  - Machine learning
  - Rapid prototyping
  - Databases
  - Front/Back-end
  - Control language/Scripting



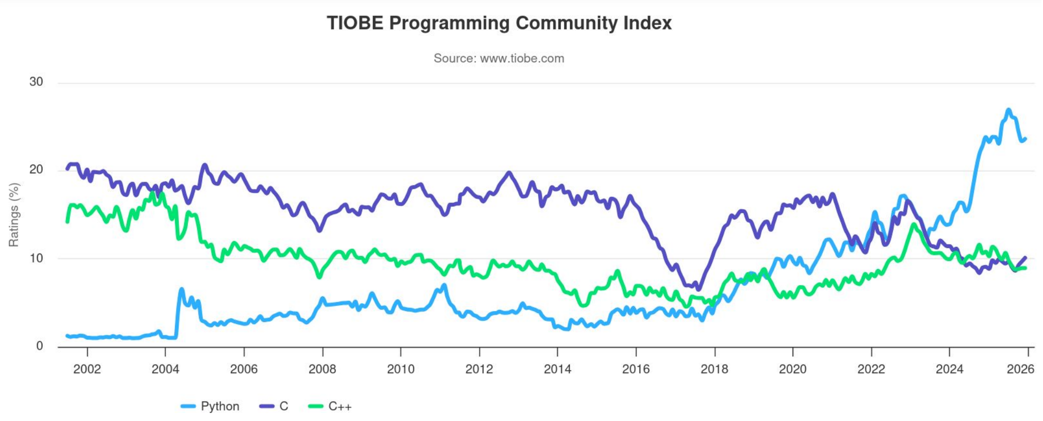
# The 5 Ws

- **Why**
  - **Excellent readability** (therefore, reusability and maintainability)
  - Developer productivity: easy to learn and use
  - Rich third-party libraries
  - Portable across OSs and architectures
  - Great integration with other languages


PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

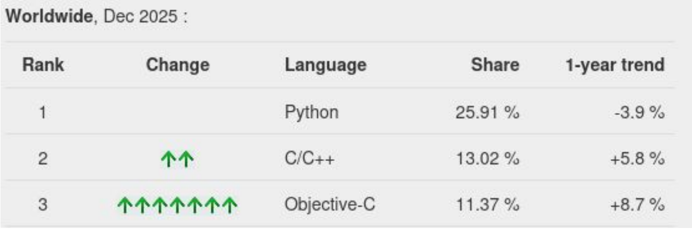
# The Language

## Diffusion



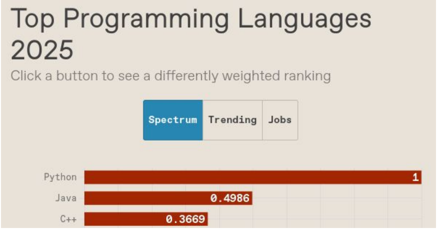
TIOBE Programming Community Index

Source: www.tiobe.com



**Worldwide**, Dec 2025 :

| Rank | Change | Language | Share | 1-year trend |
|---|---|---|---|---|
| 1 | | Python | 25.91 % | -3.9 % |
| 2 | ↑↑ | C/C++ | 13.02 % | +5.8 % |
| 3 | ↑↑↑↑↑↑↑ | Objective-C | 11.37 % | +8.7 % |

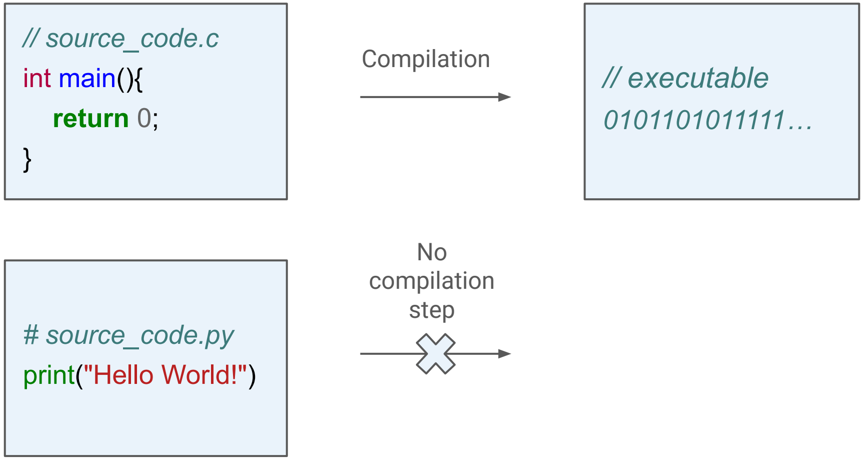


Top Programming Languages 2025

Click a button to see a differently weighted ranking

# The Language

## A dynamic one



Static lang (C-like)

```c
// source_code.c
int main(){
    return 0;
}
```

Compilation

```
// executable
0101101011111…
```

Correctness controlled at *compile-time*

Dynamic lang

```python
# source_code.py
print("Hello World!")
```

No compilation step

Correctness controlled at *run-time*

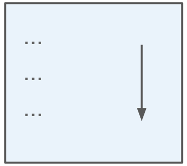
🔍 *almost*, see how python runs code

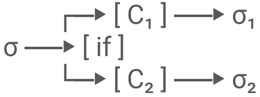
Python is a **dynamic** language: flexibility over strict control

# The Language

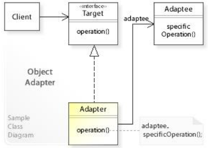
## Features

- (Strong) dynamic **typing** pi = 3.14159 *# variable initialization*
- **Multi-paradigm**: imperative, procedural, functional, object-oriented, structural (modular)





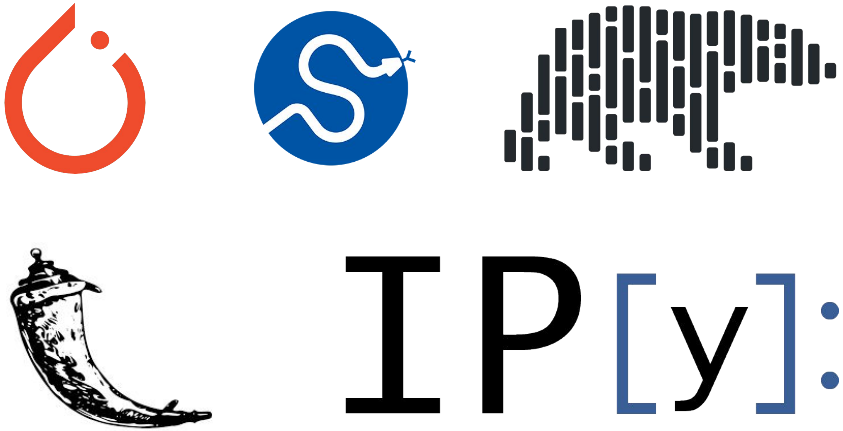
```
x = lambda a : a + 10
print(x(5))
```



- Automatic memory management: the **garbage collector**
- Built-in data structures
- Rich **standard library** (os, pathlib, asyncio, argparse, math etc) □ The Python Standard Library

# The Language

## Pros: its ecosystem

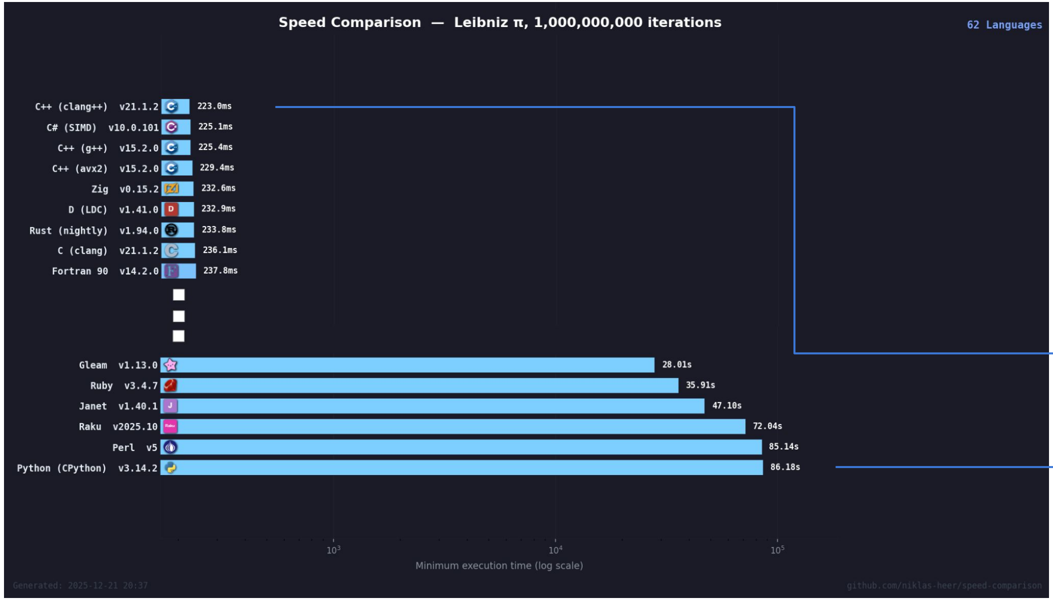


🔍 Awesome Python

🔍 CINECA’s course on scientific libraries

# The Language

## The downside



Test computing:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1},$$

C++: 0.223 s

CPython: 86.18 s

□ GitHub - niklas-heer/speed-comparison, lower is better

A lot of third-party integrations to Python are meant to alleviate this

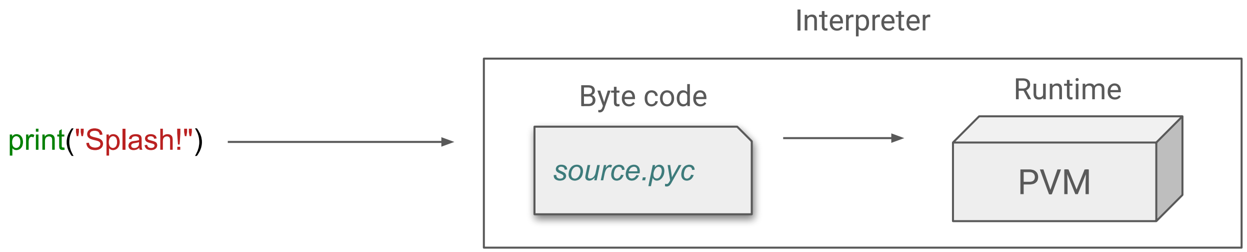
# How Python runs programs

## The interpreter

Python is freely downloadable at download Python (most probably you already have it or is installed in your working system) and accessible through the PowerShell/Terminal:

```
python
```

**The interpreter**: the program that parse, translate to machine code and execute Python statements



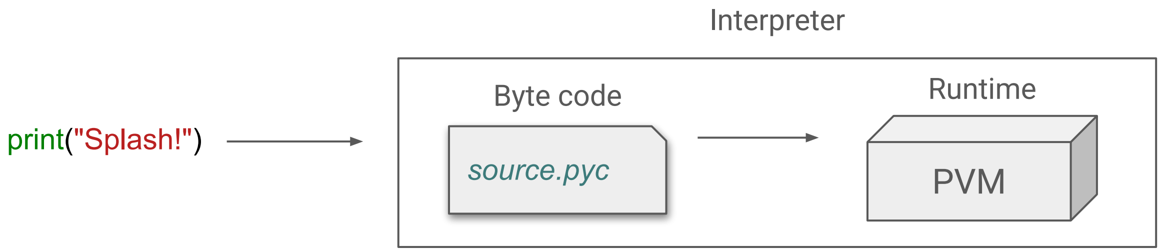
- Architecture independent
- 🔍 saved in __pycache__ for imported modules

Python Virtual Machine: what executes the Byte code

□ Learning Python 5th Ed., M. Lutz (2013)

# How Python runs programs

## The interpreter



🔍 There are multiple implementations, the reference one is CPython. Other implementations are PyPy, Codon (ahead-of-time compiler), µPython etc

# Install Python

Follow instructions from python.org and install the standard interpreter

**Linux/Unix**:

Install it through your package manager/store

```
sudo apt/dnf/pacman install python
```

**MacOS**:

Download it and use the installer or a package managers as homebrew

**Windows**:

Download it and use the installer or through the Microsoft Store

**Test**

Open the terminal (PowerShell/bash/etc) and test:

```
python --version
python -m pip --help
python -m venv --help
```

# How You run programs

## Language shell or REPL

```
$ python
Python 3.13.5 (main, Jun 25 2025, 18:55:22) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

# How You run programs

## .py files

two_plus_three.py

```python
# this is a comment. it will not be read by the interpreter
print("the result of 2+3 is:")
90+2000 # this line is executed, but never printed
print(2+3) # compute 2+3 and print it
```

```
$ python two_plus_three.py
the result of 2+3 is:
5
```

No compilation required, directly calling the interpreter on the text file containing Python statements

🔍 IDEs are great tools to write, run, debug etc. Python codes (e.g. VSCode, PyCharm)

# How You run programs

## .py files

two_plus_three.py

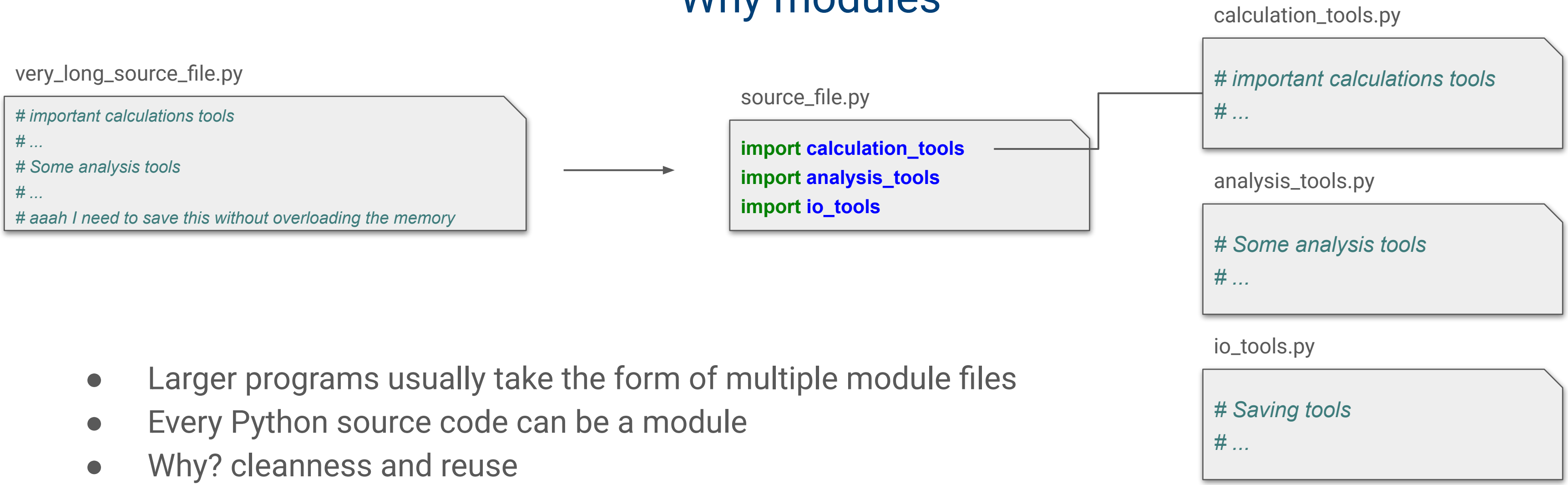
```
# this is a comment. it will not be read by the interpreter
print("the result of 2+3 is:")
90+2000 # this line is executed, but never printed
print(2+3) # compute 2+3 and print it
```

**Writing .py is the most common way to work when writing Python code!**

In this course, we will use mainly another tool to demonstrate and run Python statements that will be detailed later. However, examples with .py will be discussed too.

# Modules and Environments

## Why modules



very_long_source_file.py

```
# important calculations tools
# ...
# Some analysis tools
# ...
# aaah I need to save this without overloading the memory
```

source_file.py

```
import calculation_tools
import analysis_tools
import io_tools
```

calculation_tools.py

```
# important calculations tools
# ...
```

analysis_tools.py

```
# Some analysis tools
# ...
```

io_tools.py

```
# Saving tools
# ...
```

- Larger programs usually take the form of multiple module files
- Every Python source code can be a module
- Why? cleanness and reuse

🔍 Imports are expensive, try to `import this` twice in a REPL session (you'll notice that imports are done only once per Python process!)

CINECA

# Modules and Environments

## What are modules and how to use them

- A module is a *namespace* with a set of names called attributes (variable names, functions)
- A collection of modules is called **library** or a **package**

```
$ python
Python 3.13.5 (main, Jun 25 2025, 18:55:22) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import math
>>> math.sqrt(625)
25.0
>>> from datetime import datetime
>>> print(datetime.now().strftime("%Y-%m-%d"))
2026-01-08
>>> help(math) # open documentation
>>> dir(math) # list of accessible variables/function
```

□ The import system 🔍 More info about modules will be explored in the following lessons

# Modules and Environments

## Third-party packages

An user can download and import libraries **made by other programmers**

Registries

- *PyPi*: handled by the Python Packaging Authority
- *Anaconda* (🔍 and the other channels)

How to get packages:

- pip: the standard tool installed along with Python
- 🔍conda: dev by Anaconda (🔍 alternatives: mamba/micromamba)
- 🔍 uv: alternative to conda written from scratch and focused on speed

# Modules and Environments

## Environments

It can be a mess: where do I install libraries? what if I need a different version with respect to the one I have installed system-wise?

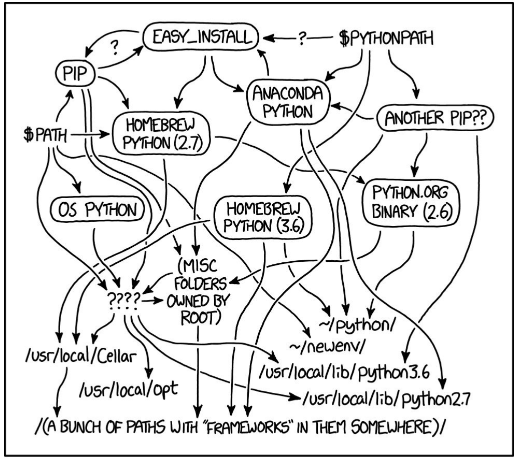
Solution: **Environments**

| 📁 project 1 | 📁 project 2 |
|---|---|
| ● package_A_v1.0 | ● package_A_v2.0 |

Separated spaces where to install different libraries or versions of these

Tools to create and manage them:

- venv: embedded with Python standard library
- 🔍conda/mamba: can manage also Python version, non-python libraries etc.
- 🔍 uv: handles venv through its venv interface



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

# Create a venv

Create a simple venv for this course (suggested to use venv)

🔍 If you are interested to test *uv* or *conda/mamba* locally, please ask to a teacher for advices

Open a terminal (for Windows: use WSL or Powershell) to create the environment with venv and activate it :

```
$ python -m venv .venv
$ source .venv/bin/activate
(.venv) $
```

To exit from the virtual environment:

```
(.venv) $ deactivate
$
```

---

Alternatively, using conda (if already installed):

```
$ conda create --name pythoncourse_env
$ conda activate pythoncourse_env
(pythoncourse_env) $
$ conda deactivate
```

# How You run programs

## Interactive Python: IPython

```
$ python
Python 3.13.5 (main, Jun 25 2025, 18:55:22) [GCC 14.2.0] on linux
>>>
```

```
$ ipython
Python 3.13.5 (main, Jun 25 2025, 18:55:22) [GCC 14.2.0] on linux
IPython 9.9.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]:
```

IP[y]:

Project born in 2001

Features

- introspection (e.g. extract doc from modules)
- rich media
- syntax highlight
- tab auto-completion
- history
- “magic” functions

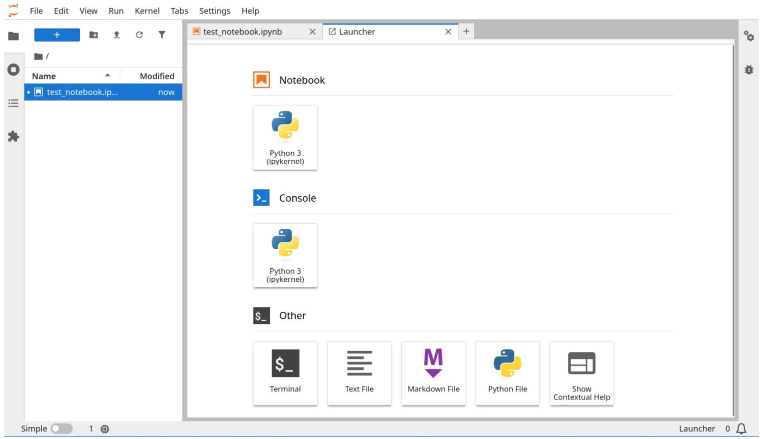
# How You run programs

## Notebooks and Jupyter

- Born in 2014 as spinoff project from IPython
- HTML UI with “kernel” back-end supported by IPython
- handy for instructors and to play around

Terminology

- *kernel*: back-end that execute the code (we will use it for Python, but other languages are available)
- *notebook*: file made of blocks (input/output) that can contain code or other (e.g. markdown)



```
$ python -m pip install jupyter
$ jupyter lab
```

# Install required packages for the course

How to install Jupiter and other dependencies within the previously created venv

Open a terminal and navigate into the directory in which the environment has been created. Activate it:

```
$ source .venv/bin/activate
(.venv) $
```

Install using pip the required packages:

```
(.venv) $ python -m pip install jupyter numpy matplotlib
pandas scipy pytest
```

---

If you used conda, do the following:

```
(pythoncourse_env) $ conda install jupyter numpy
matplotlib pandas scipy pytest
```

---

Test the installation with:

```
(.venv) $ jupyter lab
```

# What else?

## Other introductory topics

- **IDEs**: some are built to be used with Python (e.g. PyCharm) others allows for a plugin ecosystem that helps the user in writing, testing and deploying Python code (VSCode, Vim, NeoVim etc.).
- **Linting and formatting**: there are softwares that read and format your Python code following a precise set of rules. This helps readability. A rapidly popular option is ruff. Such softwares are generally recommended when programming in a production environment.