# I Semester 2025/2026: Programming for chemistry

(in Python!)

Davide Ceresoli (davide.ceresoli@cnr.it)

Instituto di Scienze e Tecnologie Chimiche "G. Natta" (CNR-SCITEC)

September 16, 2025

# Outline

# Introduction

- Dr. Davide Ceresoli, Senior Researcher at CNR-SCITEC
- office phone: 02-503-14276
- email: davide.ceresoli@cnr.it
- background: *laurea* in Materials Science, PhD in Physics
- activity: ab-initio DFT calculations, molecular dynamics, high-pressure, thermoelectrics, code development (Quantum-Espresso)

# Course organization

- 6 CFU, 48 hours (24 lectures two hours each)
- Computer room 310, settore didattico via Celoria:
  - tuesdays 08:30-10:30
  - fridays 08:30-10:30
- MyAriel course website (announcements, slides, ...):
  `https://myariel.unimi.it/course/view.php?id=9408`
- Mirror course website (calendar, slides, ...):
  `https://dceresoli.github.io/2025-Programming`

# Course objectives

- No previous knowledge of programming is assumed!
- By the end of the course, you will:
  - Understand fundamental concepts of programming imperative languages
  - Design algorithms to solve simple problems
  - Learn the Python programming language
  - Solve some chemistry-related problems
  - Have fun programming, maybe a computer game...

# Textbook and reference material

- There are plenty of online free Python books, tutorials and resources
- P. Wentworth *et al.*, *How to Think Like a Computer Scientist in Python 3*, available free at:
  `https://openbookproject.net/thinkcs/python/english3e/`
- The official Python 3 documentation:
  `https://docs.python.org/3/`

# Programming tools: Python + Jupyter

- We'll use the interactive **Jupyter** notebook most of the time. Here are the tools you need to install:
- Windows: Anaconda *(too much user friendly!)*
- Windows: WSL with Debian/Ubuntu
  `sudo apt install python3 jupyter numpy scipy matplotlib`
- Linux and VSCode *(the only Microsoft-product that works!)*
- Mac OS: I have no experience!
- ...

- In the cloud: no installation, only need a browser, you get CPU (and GPU) for free
  1. Google Colab `https://colab.research.google.com/`
  2. Kaggle `https://www.kaggle.com/`
  3. Binder `https://mybinder.org/v2/gh/dceresoli/2025-Programming/HEAD`
  4. Microsoft Azure Notebooks, Datalore, ...

# Policies and grading

- Lectures: will be interactive, with questions and interactive problem solving
- Attendance is recommended
- **Please, do not use AI assistants such as: ChatGPT, Claude, Gemini!**
- There will be "free programming practice" days to catch up, exercise, propose problems
- Final exam: oral, couple of general questions, coding 3–4 notebook cells

Questions?

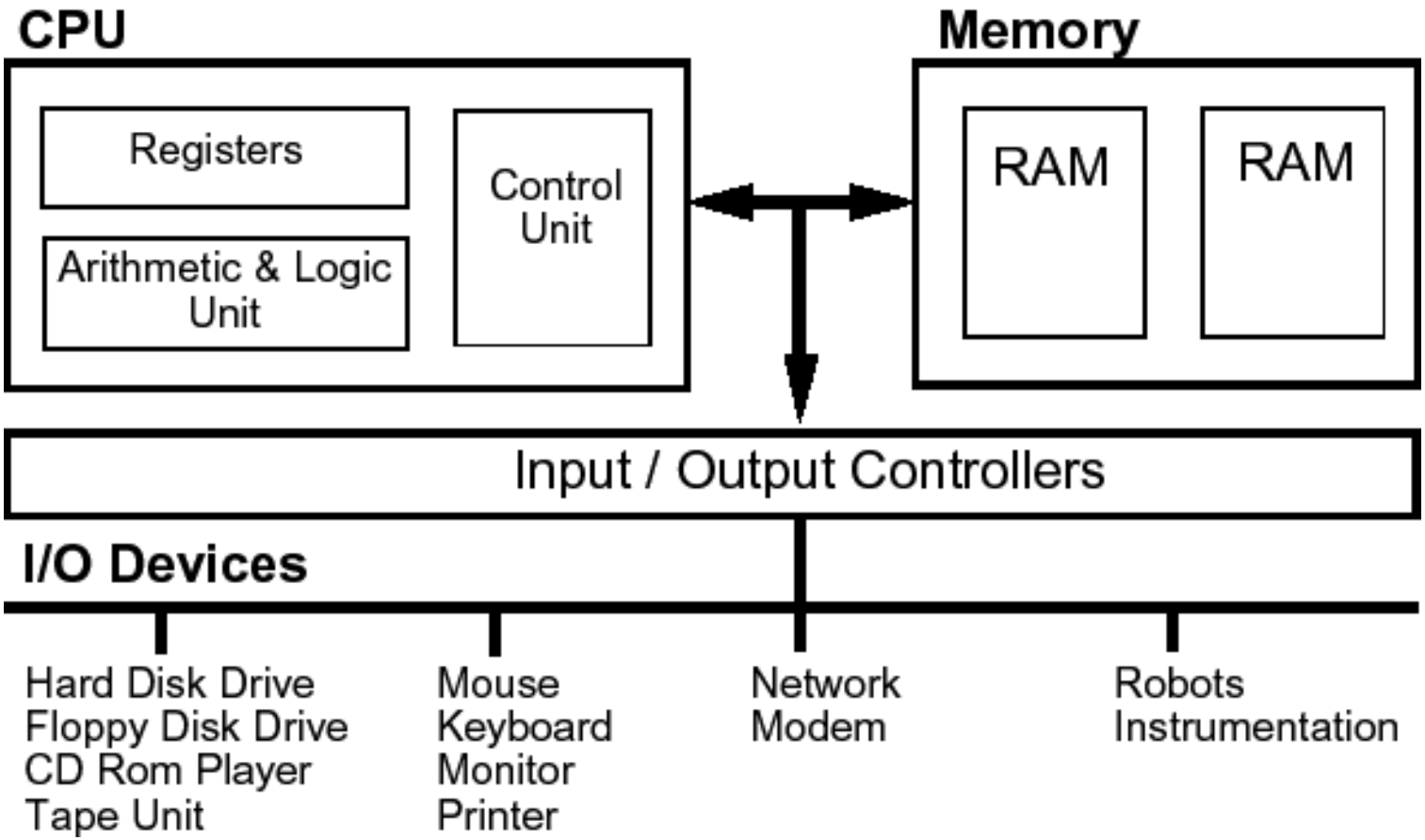
# Modern computer architecture



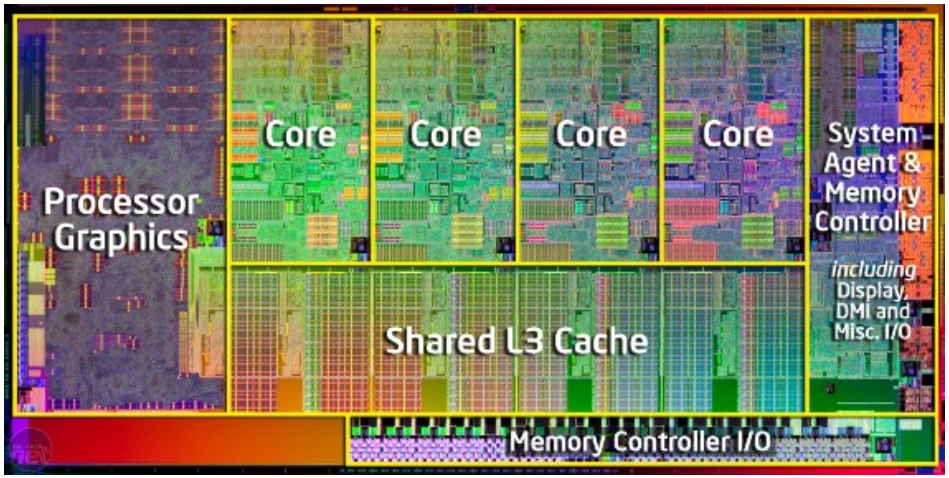

## Take-home message

Computer don't solve problems (yet), people do!

# Modern computer architecture



# The CPU




- The CPU reads the both code and data from RAM
- The code is executed one instruction at the time, data is moved to/from memory
- Conditionally, the CPU jumps to a different location in the code
- The operating system (OS) takes care of:
  - scheduling execution of $N$ processes on $m$ CPU cores
  - move data to/from disk, network and RAM
  - interact with the hardware and with the user

## The CPU understands only low-level *Assembly* instructions

- This are very simple instructions such as:
  - move bytes from RAM to CPU registers and back
  - do arithmetic operations on CPU registers
  - advance to the next instruction, or jump to an other location
  - each CPU has a different set of instructions!

```
Disassembly of section .text:

0000000000000000 <distance>:
   0:   55                          push   %rbp
   1:   48 89 e5                    mov    %rsp,%rbp
   4:   48 83 ec 20                 sub    $0x20,%rsp
   8:   f2 0f 11 45 f8              movsd
%xmm0,-0x8(%rbp)
   d:   f2 0f 11 4d f0              movsd
%xmm1,-0x10(%rbp)
  12:   f2 0f 11 55 e8              movsd
%xmm2,-0x18(%rbp)
  17:   f2 0f 10 45 f8              movsd
-0x8(%rbp),%xmm0
  1c:   66 0f 28 c8                 movapd %xmm0,%xmm1
  20:   f2 0f 59 4d f8              mulsd
-0x8(%rbp),%xmm1
  25:   f2 0f 10 45 f0              movsd
-0x10(%rbp),%xmm0
  2a:   f2 0f 59 45 f0              mulsd
-0x10(%rbp),%xmm0
  2f:   f2 0f 58 c8                 addsd  %xmm0,%xmm1
  33:   f2 0f 10 45 e8              movsd
-0x18(%rbp),%xmm0
  38:   f2 0f 59 45 e8              mulsd
-0x18(%rbp),%xmm0
  3d:   f2 0f 58 c1                 addsd  %xmm1,%xmm0
  41:   e8 00 00 00 00              callq
```

For us, it is easier to program in higher-level languages

The previous block of bytes and assembly instructions, is equivalent to:

```
#include <math.h>

double distance(double x, double y, double z)
{
    return sqrt(x*x + y*y + z*z);
}
```

**This code runs on almost every CPU**

This is why we need to learn programming languages!

## Compiled languages

- Source code is analyzed entirely and checked for errors
- Then, the source code is *compiled* (aka translated) into assembly instructions
- Possibility to perform deep code optimizations
- Finally, multiple code units are *linked* together and with *libraries* of functions
- The results is the *executable*, that can be run by the operating system *at the maximum performance*
- Compiled languages: C, C++, Fortran, Pascal, Go, Rust, ...

## Interpreted languages

- Source code is processed line-by-line and checked for errors
- The interpreter *runs* the source code, also interactively
- Possibility to debug and inspect the code while writing
- Usually very easy and fun to program with
- Difficult to optimize, performance much lower than compiled languages
- Interpreted languages: Python, Basic, Forth, Javascript, PHP, Lisp, Octave, ...

## In the middle: just-in-time compiled languages

- Source code is processed line-by-line and checked for errors, also interactively
- The code is compiled *on-the-fly* to CPU instructions
- Performance is between compiled and interpreted languages
- Price to pay is longer startup time and need of support run-time libraries
- JIT languages: Java, Javascript, Julia, ...

# Programming language popularity

TIOBE
the software quality company

About us | Knowledge | News | Coding Standards | TIOBE Index | Contact

Products | Quality Models | Markets | Schedule a demo

when starting to build a new software system. The definition of the TIOBE index can be found here.

| Aug 2025 | Aug 2024 | Change | Programming Language | Ratings | Change |
|---|---|---|---|---|---|
| 1 | 1 | | Python | 26.14% | +8.10% |
| 2 | 2 | | C++ | 9.18% | -0.86% |
| 3 | 3 | | C | 9.03% | -0.15% |
| 4 | 4 | | Java | 8.59% | -0.58% |
| 5 | 5 | | C# | 5.52% | -0.87% |
| 6 | 6 | | JavaScript | 3.15% | -0.76% |
| 7 | 8 | ↑ | Visual Basic | 2.33% | +0.15% |
| 8 | 9 | ↑ | Go | 2.11% | +0.08% |
| 9 | 25 | ⇈ | Perl | 2.08% | +1.17% |
| 10 | 12 | ↑ | Delphi/Object Pascal | 1.82% | +0.19% |
| 11 | 10 | ↓ | Fortran | 1.75% | -0.03% |
| 12 | 7 | ⇊ | SQL | 1.72% | -0.49% |
| 13 | 30 | ⇈ | Ada | 1.52% | +0.91% |
| 14 | 19 | ⇈ | R | 1.37% | +0.26% |
| 15 | 13 | ↓ | PHP | 1.27% | -0.19% |
| 16 | 11 | ⇊ | MATLAB | 1.19% | -0.53% |

# Why Python?

## Python is slow!

- Can be 100x times slower than C/C++/Fortran

## Python is fast!

- If you use C/C++/Fortran libraries

Julia is emerging as a fast and powerful language for scientific programming

# The Python programming language

- Conceived by Guido van Rossum in the '90s (Python 1)
- Python 2 was popular until few years ago, not used anymore
- We'll be using at least Python 3.11 (shipped with Linux Debian 12)
- Python 2 and Python 3 are incompatible, the code must be converted
- *I will point out major differences between Python and other programming languages*

# How does Python look like?

```
#!/usr/bin/env python
# Anderson model in 2d

import numpy as np
import numpy.random as random
import scipy.sparse as sparse
from scipy.sparse.linalg import eigs, eigsh
import matplotlib.pyplot as plt

# number of sites
Nx, Ny = 100, 100

# hopping and randomness
t = 1.0
r = 0.1

# setup hamiltonian
N = Nx*Ny
H = sparse.lil_matrix((N,N))
H.setdiag(r*random.randn(N))

for ix in range(Nx):
    for iy in range(Ny):
        i = ix*Ny + iy
        H[i,(i+1) % N] = t
        H[i,i-1] = t
        H[i,(i+Ny) % N] = t
        H[i,i-Ny] = t
...
```

## Pros

- Imperative, structured (make use of "functions"), modules
- Plenty of existing libraries and modules
- More than one way to write an algorithm
- Easy to learn!

## Cons

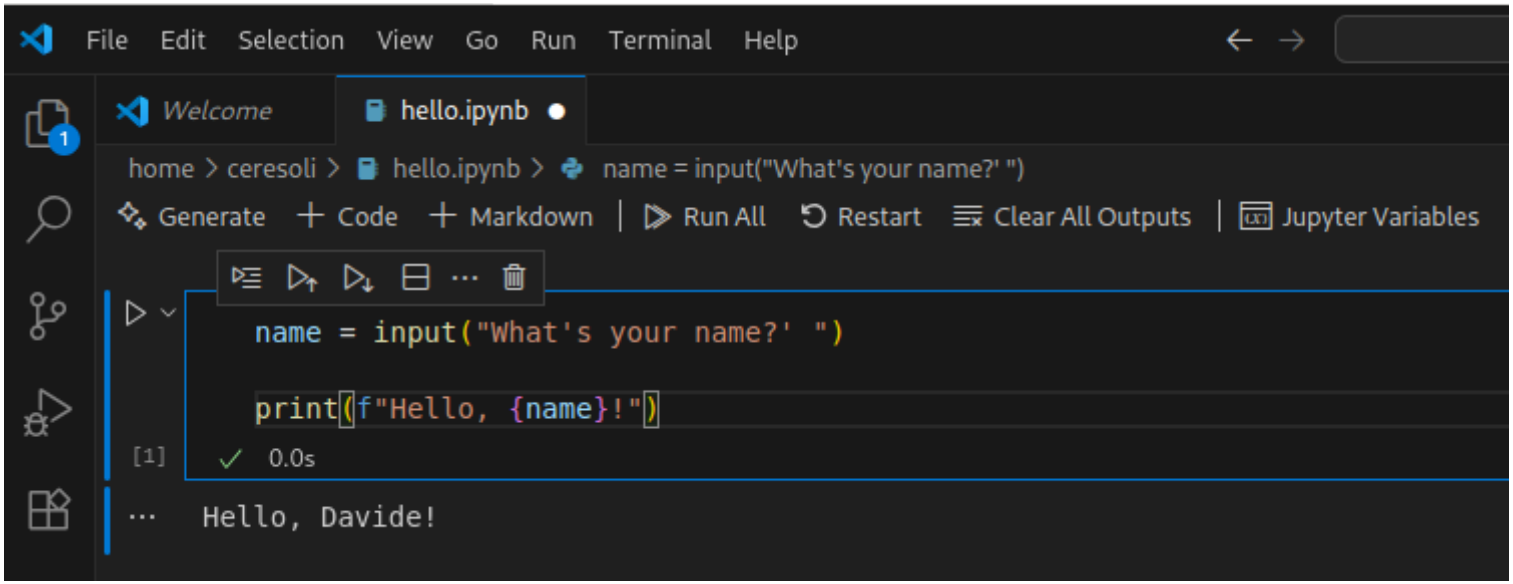
- Easy to mess up with data types
- Not CPU and memory efficient
- Easy to make mistakes!

# Our first Python program (Linux shell)

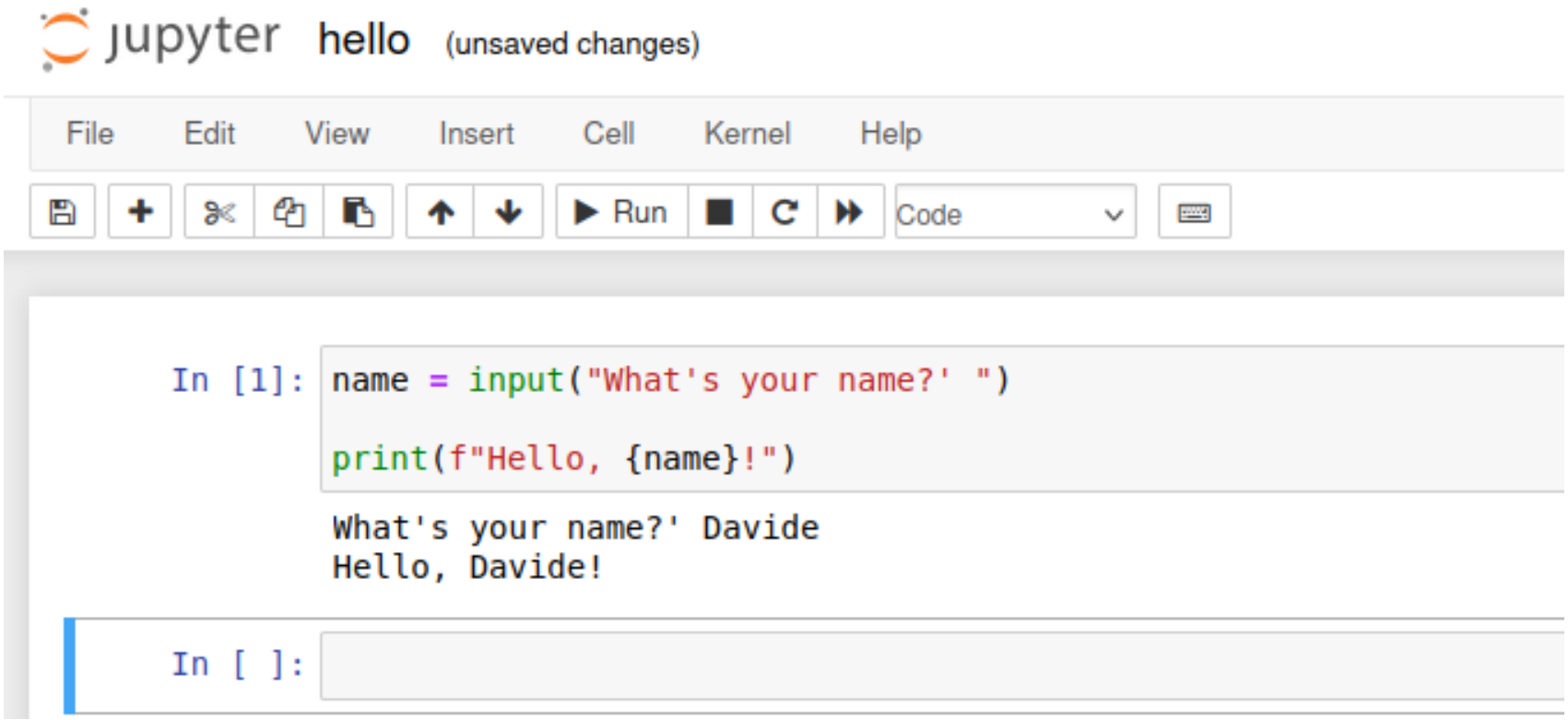
```
ceresoli@bluestar:~$ cat hello.py
#!/usr/bin/env python

name = input("what's your name? ")
print(f"Hello, {name}")
ceresoli@bluestar:~$ chmod 0755 hello.py
ceresoli@bluestar:~$ ./hello.py
what's your name? Davide
Hello, Davide
ceresoli@bluestar:~$
```

# Our first Python program (VScode)



# Our first Python program (Jupyter)



# Time to install and setup Python!

Next step: download lecture #1 notebook from MyAriel or from `https://github.com/dceresoli/2025-Programming`.

# Recap

- Low-level vs high-level languages
- Compilers vs interpreters
- The first program in Python
- Interactive coding with Jupyter notebooks

**Questions?**