

Spring 2025: Programming for chemistry (in Python!)

Davide Ceresoli (davide.ceresoli@cnr.it)

Istituto di Scienze e Tecnologie Chimiche “G. Natta” (CNR-SCITEC)

August 29, 2025

- 1 Course logistics
- 2 The way to program
- 3 The Python programming language
- 4 Recap

- 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)

- 6 CFU, 48 hours (24 lectures two hours each)
- lectures (settore didattico via Celoria):
 - tuesdays 08:30-10:30, room ???
 - fridays 08:30-10:30, room ???
- course website:
<https://dceresoli.github.io/2025-Programming>



- No previous knowledge of programming is assumed!

- No previous knowledge of programming is assumed!
- By the end of the course, you will:

- 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...

- 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/>

- We'll use the interactive Jupyter notebook most of the time. Here are the tools you need to install:

- 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!*)

- 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!*)

- 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 idea!

- 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 idea!
- In the cloud: Google Colab
<https://colab.research.google.com/>

- Lectures: can be interactive, with questions and interactive problem solving

- Lectures: can be interactive, with questions and interactive problem solving
- Attendance is recommended

- Lectures: can be interactive, with questions and interactive problem solving
- Attendance is recommended
- **You are not allowed to use AI assistants such as: ChatGPT, Claude, Gemini!**

- Lectures: can be interactive, with questions and interactive problem solving
- Attendance is recommended
- **You are not allowed to use AI assistants such as: ChatGPT, Claude, Gemini!**
- There will be two “free programming practice” days to catch up, exercise, propose problems

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

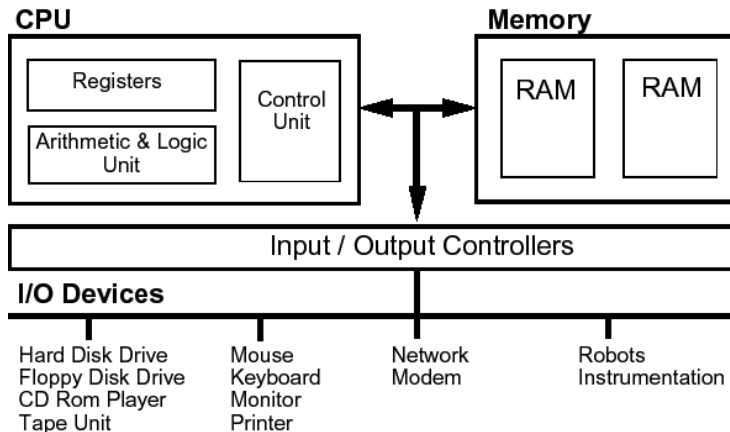
Questions?

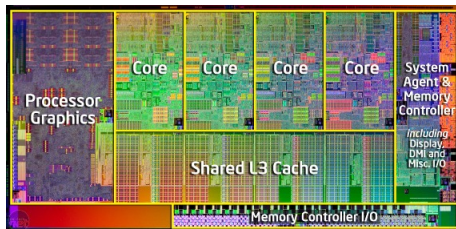


Copyright © Randy Glasbergen. www.glasbergen.com

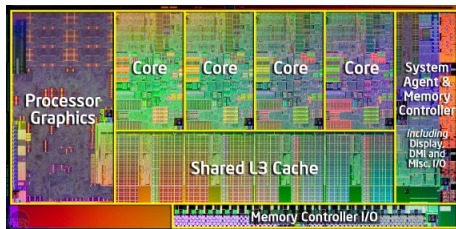
Take-home message

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





- 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 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   0
```



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);
}
```

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

Compiled languages

- Source code is analyzed entirely and checked for errors
- Then, the source code is *compiled* (aka translated) into assembly instructions

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

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

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

- 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

Interpreted languages

- Source code is processed line-by-line and checked for errors
- The interpreter *runs* the source code, also interactively

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

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

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

- 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

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

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

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

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, Wasm, ...

In the middle: just-in-time compiled languages

- Source code is processed line-by-line and checked for errors, also interactively

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

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

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

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, Wasm, ...

















Programming language popularity



[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%



Python is slow!

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

Why Python?

Python is slow!

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

Python is fast!

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

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

- 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!

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!

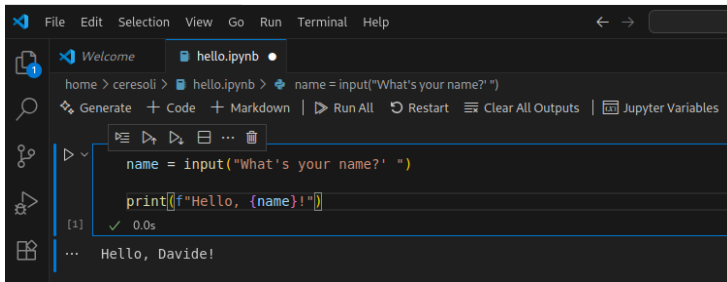
Cons

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

Our first Python program (Linux shell)

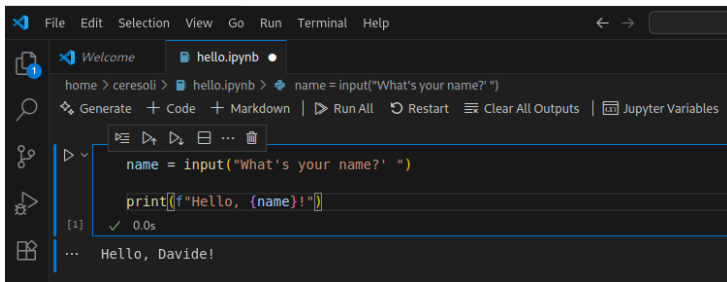
```
ceresoli@bluestar:~$ cat hello.py
#!/usr/bin/env python

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:~$
```



The screenshot shows the Jupyter Notebook interface within a VS Code editor. The file 'hello.ipynb' is open, and the code from the previous terminal window is pasted into the notebook cells. The first cell contains the Python code: `name = input("What's your name? ")` and `print(f"Hello, {name}!")`. The second cell shows the output: `[1] ✓ 0.0s` and `... Hello, Davide!`. The interface includes a menu bar (File, Edit, Selection, View, Go, Run, Terminal, Help), a toolbar with icons for file operations and code execution, and a sidebar with icons for Explorer, Search, Source Control, and Run and Debug.

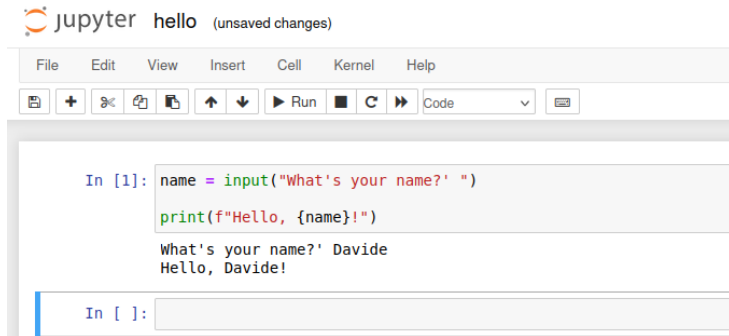
Our first Python program (VSCode)



The screenshot shows the Visual Studio Code interface with a Jupyter notebook named 'hello.ipynb' open. The notebook contains two code cells. The first cell has the code `name = input("What's your name? ")`. The second cell has the code `print(f"Hello, {name}!")`. The output of the second cell is 'Hello, Davide!'. The interface includes a menu bar (File, Edit, Selection, View, Go, Run, Terminal, Help), a sidebar with icons for Explorer, Search, Source Control, Run and Debug, and Extensions, and a toolbar with buttons for Generate, Code, Markdown, Run All, Restart, Clear All Outputs, and Jupyter Variables.

```
home > ceresoli > hello.ipynb > name = input("What's your name? ")  
Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variables  
name = input("What's your name? ")  
print(f"Hello, {name}!")  
[1] ✓ 0.0s  
... Hello, Davide!
```

Our first Python program (Jupyter)



The screenshot shows a Jupyter Notebook window titled "hello (unsaved changes)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". Below the menu is a toolbar with icons for saving, adding cells, undo, redo, copy, paste, and running code. The main area contains a code cell with the following Python code:

```
In [1]: name = input("What's your name?' ")  
        print(f"Hello, {name}!")
```

The output of the code is displayed below the code cell:

```
What's your name?' Davide  
Hello, Davide!
```

Below the output, there is an empty code cell with the prompt "In []:".

Time to install and setup Python!

Next step: download lecture #1 notebook from
<https://github.com/dceresoli/2025-Programming>.

- Low-level vs high-level languages

- Low-level vs high-level languages
- Compilers vs interpreters

- Low-level vs high-level languages
- Compilers vs interpreters
- The first program in Python

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

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

Questions?