032CM - 2025

PROGRAMMING FOR COMPUTATIONAL CHEMISTRY

Introduction to Python

Gianluca Levi

gianluca.levi@units.it, giale@hi.is

Office: Building C11, 3rd floor, Room 329

Fall 2025

Why Python?

Free, **open-source**, available for Linux/macOS/Windows

Easy to read and understand → focus on the scientific computing

Interpreted language (**scripting**)

- Code is compiled automatically the **interpreter**, on the fly
- **Run immediately** without manual compilation step
- Ideal for data manipulation/visualization and exploratory work

Huge community & resources → tutorials, examples everywhere

Why Python?

Modular language with vast collection of free, open-source libraries

Essential **scientific packages**

Why Python?

Modular language with vast collection of free, open-source libraries

Essential **scientific packages**

- **NumPy**
  - Fundamental package for **numerical and scientific computation**
  - N-dimensional **arrays** and vectorized math

Why Python?

Modular language with vast collection of free, open-source libraries

Essential **scientific packages**

- **NumPy**
  - Fundamental package for **numerical and scientific computation**
  - N-dimensional **arrays** and vectorized math
- **SciPy**
  - **Scientific algorithms** for integration, optimization, linear algebra, etc.

Why Python?

Modular language with vast collection of free, open-source libraries

Essential **scientific packages**

- **NumPy**
  - Fundamental package for **numerical and scientific computation**
  - N-dimensional **arrays** and vectorized math
- **SciPy**
  - **Scientific algorithms** for integration, optimization, linear algebra, etc.
- **Matplotlib**
  - Standard library for **data visualization**, **plots**, and publication-quality figures

Why Python?

Modular language with vast collection of free, open-source libraries

Essential **scientific packages**

- **NumPy**
  - Fundamental package for **numerical and scientific computation**
  - N-dimensional **arrays** and vectorized math
- **SciPy**
  - **Scientific algorithms** for integration, optimization, linear algebra, etc.
- **Matplotlib**
  - Standard library for **data visualization**, **plots**, and publication-quality figures
- **Jupyter**
  - Platform for **interactive computing**
  - Interactive **notebooks** mixing code and text
  - Promotes **reproducible and collaborative research**

Why not Python

Can be **slower** than compiled languages (Fortran, C, C++)

- Implement **performance-critical operations in Fortran/C/C++** and call them from Python
- NumPy and SciPy performance-critical libraries written in **Fortran/C/C++**

Reproducible environments and **shipping standalone executables can be tricky** across OS/architectures

Our tool for interactive coding with Python: JupyterLab

Web-based **Integrated Development Environment (IDE)** for interactive computing

Jupyter Notebook

- Interactive document that **combines live code, output** (e.g. result of calculations, plots, tables), **and formatted text**

Interactive computational narrative

Learn more about Jupyter

https://coderefinery.github.io/jupyter/interface/

Your first JupyterLab session

To start a JupyterLab session

1. Connect to the server through the **Bitvise SSH client**
2. In the terminal, type:

   >> jupyter lab

3. Copy http://dscfalpha7.units.it:<port>/lab?reset and paste to your browser address bar (<port> is a number shown by Jupyter)
4. If prompted for a password, use your **server login password**

Learn more about Markdown
https://commonmark.org/help/

Your first JupyterLab session

To end a JupyterLab session

1. **Save your work** in all open notebooks
2. Either:
   - Go to the terminal where JupyterLab is running and press Ctrl+C
     - When asked Shutdown this Jupyter server? [y/N], type y and press Enter
   - Click File → Shut Down
     - Confirm the shutdown
3. Close the browser tab and your SSH terminal

Note!

We all share the **same environment and HOME** Do **not** edit, move, or delete files that aren't yours. Work **only inside your own folder**.
IMPORTANT: When you finish, **close your JupyterLab browser tab** to prevent others from accessing your session and to free resources.

Why Jupyter?

Test, run, explore quickly

- Perfect for data analysis and plotting

Computational narrative

- Combine **code + text (Markdown) + figures** in one shareable document

Reproducible & shareable

- Notebooks easy to publish on repositories

Submit assignments to your teacher/supervisor as one document (can check code)

- Code in notebooks **can be published**
- LIGO released analysis of data related to **Gravitational Waves** as notebooks (https://gwosc.org/tutorials/)

Why not Jupyter

Notebooks aren't "real programs"

- **Hard to structure as modules**, or write unit tests

Cells can run in any order

- Results may be **non-reproducible**

Notebooks are not named by default and tend to collect a a lot of stuff

- **Keep your notebooks tidy** and organized

Not ideal for production runs

- Long runs, batch jobs, work better as **scripts/program packages**

Data objects

Programs manipulate **data objects**

Data objects have a **type**, which defined the type of operations we can perform with them

- 37 → number → can add, subtract, multiply, divide
- 'Time' → character (string of characters) → can concatenate, extract substring, but not divide by a number

Data objects

Programs manipulate **data objects**

Data objects have a **type**, which defined the type of operations we can perform with them

- 37 → number → can add, subtract, multiply, divide
- 'Time' → character (string of characters) → can concatenate, extract substring, but not divide by a number

Scalar (cannot be divided into parts)

- **Numbers** (e.g. 3, 7.46)
- **Logical values** (True, False)

Non-scalar (have internal structures one can access)

- **Sequence of characters** (e.g. 'Time')
- **Lists** (e.g. [1, 2, 'H'])
- **Dictionaries** {"H2O": 18.015, "N2": 28.014}

Types of scalar objects

Built-in (intrinsic) scalar types

int

- Whole numbers (..., -2, -1, 0, 1, 2, ...)

float

- Real (floating-point) numbers (e.g. 3.141593, 2.998e8)
- Approximate real values, stored in floating-point form (**double precision**)

complex

- Complex numbers (e.g. 3.0 - 2.0j)
- Pair of real and imaginary parts

bool

- Logical (Boolean) values (True, False)

NoneType

- Special type with a single value None
- Used to represent a "no value" or a missing value

You can check the data type of any object using type()

Type conversion (casting) and arithmetic operations

Convert between integers and real numbers

float(i) → converts integer i to real

int(x) → truncates real x to integer

round(x) → rounds real x to nearest integer

Basic operators

+ → addition

- → subtraction

* → multiplication

/ → division

// → floor (integer) division

% → Remainder (modulus)

** → exponentiation

Using libraries and modules

A **library** is a **collection of functions** and tools that can be reused

- You can **import** a library to access its functions

```
import numpy
numpy.sqrt(16)
```

- Import with a shortcut name (same result)

```
import numpy as np
np.sqrt(16)
```

Using libraries and modules

A **library** is a **collection of functions** and tools that can be reused

- You can **import** a library to access its functions

```
import numpy

numpy.sqrt(16)
```

- Import with a shortcut name (same result)

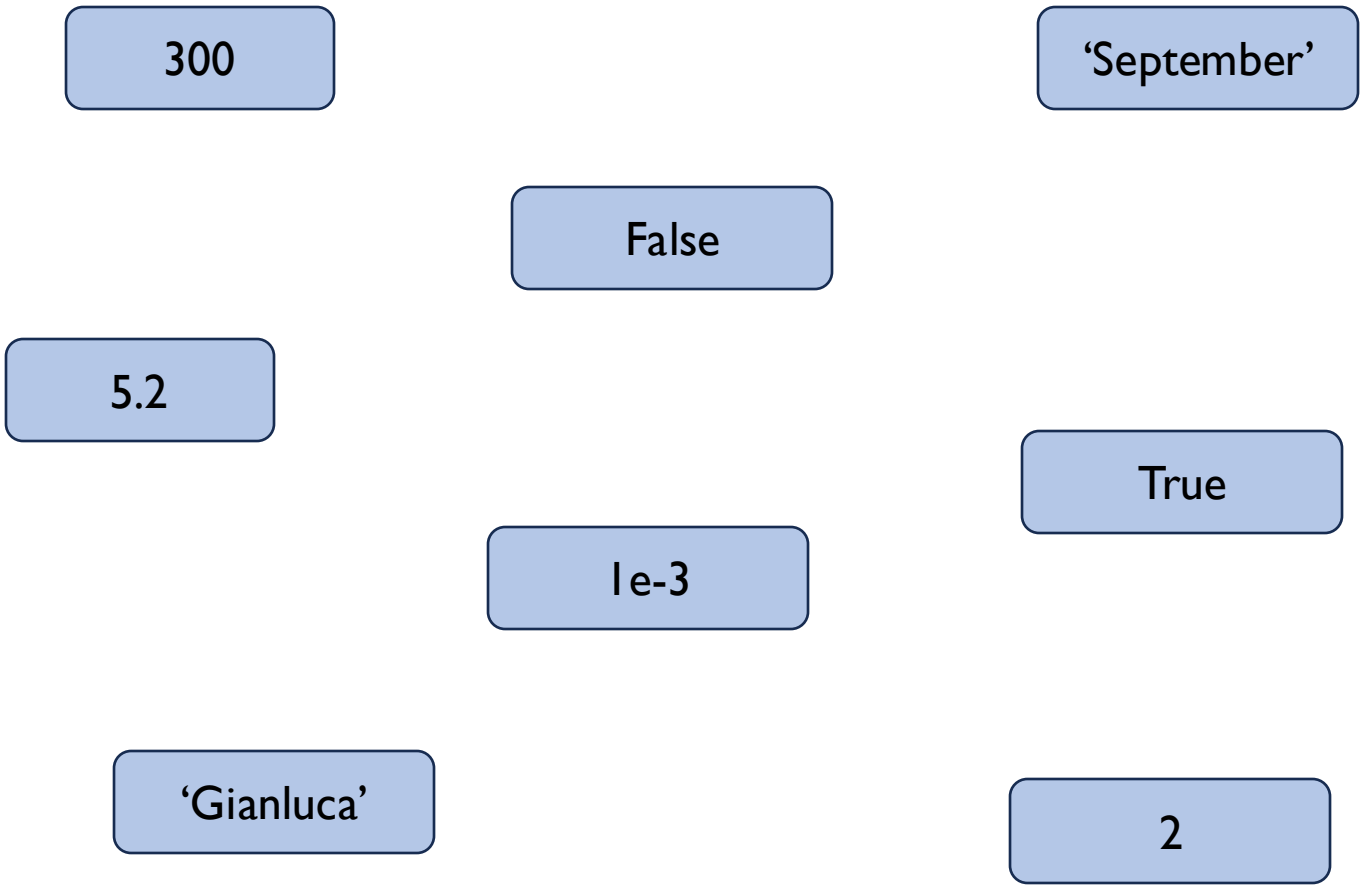
```
import numpy as np

np.sqrt(16)
```

Note!

Place **all import statements at the top** of your Python script or notebook

- Makes dependencies clear and avoids confusion about where functions come from
- In Jupyter notebooks, keep a **dedicated "Imports" cell** near the top so you can re-run it easily

Many data objects, what to do with them?



Many data objects, what to do with them?

Variables assignment

temp = 300

month = 'September'

stop = False

a = 5.2

flag = True

eps = 1e-3

name = 'Gianluca'

const = 2

Variable assignment in Python

Equal sign (=) is an **assignment operator**, not a test for equality

- The expression on the **right-hand side** is evaluated
- The resulting value is **stored** in the variable on the **left-hand side**

```
variable_name = expression
```

i = i + 1 → take the current value of i, add 1, and store the result back in i

Variable assignment

Variable assignment in Python

Equal sign (=) is an **assignment operator**, not a test for equality

- The expression on the **right-hand side** is evaluated
- The resulting value is **stored** in the variable on the **left-hand side**

```
variable_name = expression
```

i = i + 1 → take the current value of i, add 1, and store the result back in i

No declaration needed (**dynamic typing**)

Type determined at runtime from the assigned value

- Can reassign a variable to a value of **any type**
- Makes coding faster and flexible, but also more **error-prone**

Types of non-scalar objects (Collections)

Strings (str)

- **Sequence of characters**
  - Defined using **single** or **double quotes**
  - Can be indexed and sliced
    "abc"[1] → 'b'

Lists (list)

- **Ordered collection** of objects that can be of **any type** (even mixed)
  - Defined by comma-separated values in **square brackets**
  - **Mutable** → contents can be changed (add, remove, reorder)
  - Useful when **order matters**
    [1, 2, "H"]

Dictionaries (dict)

- **Mappings** of key → value pairs
  - Defined using **curly braces** {} with keys and values separated by colons
  - Ideal for **labeled data**
  - {"H2O": 18.015, "N2": 28.014}

Loops and if conditions

Iterative / counting loop

```
for object in sequence:
    statements
```

Conditional / while loop

```
while condition:
    statements
```

Conditional structure

```
if condition1:
    statements1
elif condition2:
    statements2
else:
    statements3
```

Relational operators

== (equal to)

!= (nonequal to)

> (greater than)

>= (greater or equal)

< (less than)

<= (less of equal)

Note!
Indentation in Python matters, it defines code blocks. There are no end do or endif keywords!

Assignment 3

Problem 1

Write a Python code in a `Jupyter notebook` to compute the value of the following two-dimensional functions for $x = 3.0$ and $y = 4.0$. *Note* that variables are case-sensitive, and lowercase variable names are generally preferred.
Hint: Import the `numpy` library in your `Jupyter notebook`. You will need to use `numpy` functions such as: `exp()`, `sqrt()`, `sin()`, and `pi`.

(a) $F(x,y) = yA\exp\left[-\frac{(x-x_0)^2}{2\sigma_x^2}\right]$ where $A = 1.5,\ x_0 = 0.5,\ \sigma_x = 3.0.$

(b) $g(x,y) = \frac{\sqrt{x+y}}{\sqrt{x}+\sqrt{y}}$

(c) $f(x,y) = \exp\left(\frac{-\sin(\pi x)}{(x+y)^2+xy^6}\right)$

(d) $G(x,y) = \log_{10}(x) + \ln(y)$

Problem 2

Create a Python code in a `Jupyter notebook` to calculate the mean molar mass of dry air based on the composition and masses reported in the following table:

Molecule	Molar Mass ($g \cdot mol^{-1}$)	Composition (%)
N_2	28.013	78.084
O_2	31.999	20.946
Ar	39.948	0.9290
CO_2	44.010	0.041

The mean molar mass of dry air is given by the weighted average:

$$< M >_{\text{air}} = \sum_i x_i M_i$$

where x_i is the fractional composition and M_i is the molar mass of the ith component.

Assignment 3

Problem 3

In a `Jupyter notebook`, create a Python code to write an `XYZ` file for a water molecule with the following atomic Cartesian coordinates using Python lists (or dictionaries) and simple loops.

Atom	x (bohr)	y (bohr)	z (bohr)
O	0.00000000	0.00000000	0.00000000
H	1.80941647	0.00000000	0.00000000
H	-0.45334744	1.75170319	0.00000000

Use an **if**-statement to handle units: If the units are bohr, convert all coordinates to ångström. Otherwise (for any other value), assume the numbers are already in ångström. Write the XYZ file `water.xyz` with the standard format:

1. First line: number of atoms.
2. Second line: a short comment, e.g. `H2O coordinates in Å`.
3. Then one line per atom: `symbol x y z` with 6 decimals.

Hint: Below is an example of Python code to write to a text file. *Note* that `\n` is used to break a line.

```
with open('example.txt', 'w') as f:
    f.write('This is an example, line 1' + '\n')
    f.write('This is an example, line 2' + '\n')
```