

# Programming in Python[1]

Mattia Monga

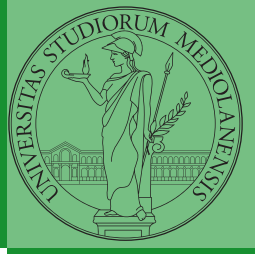
Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

Academic year 2020/21, II semester



# Lecture I: Programming in Python for quantitative biologists

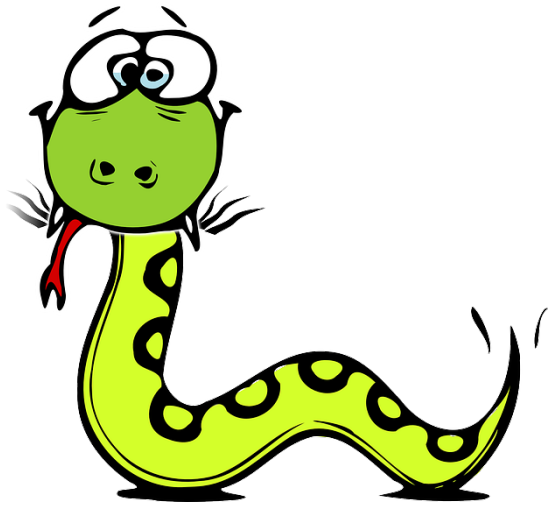
# Programming in Python (for quantitative biologists)






The course introduces imperative programming by referring to the Python language.

1. Python3 and its object-oriented features;
2. Python3 libraries that can be useful in scientific computation and data analysis, in particular `NumPy` and `pandas`.



Everything will be available on:

`mattiamon.ga/pyqb`

(a.k.a. `https://mameli.docenti.di.unimi.it/pyqb`)

Please: fill in the survey, subscribe to Zulip.

# Course schedule




- Mondays, Fridays 9:30–11:30 (am)
- Lectures: 40h (online), Labs: 16h (face-to-face if possible)
- Labs always on Friday, 12/3 19/3 9/4 23/4 7/5 21/5 4/6 11/6
- We will explore different setups: (1) a "scaffolded" one for the first steps, (2) the plain python interpreter, and finally (3) the notebooks popular in scientific practice
- Tutor: dott. Davide Paolillo (computer science master student)
- Text: every Python3 reference/book/tutorial is ok, you can access freely to the book linked on the website
- Final test: write (small) python programs without help

# Why Python?




Programming can be approached in many "languages", the fundamental skills are general... but you cannot learn without referring to a specific language.

- A precise requirement of the teaching committee
- Very popular in the scientific landscape
- Easy to learn, many useful libraries, free software
- Alternatives: Fortran, C, Matlab, Mathematica, R, Julia, ...
- Python is slower, but it is considered easier to understand and manage

# Which Python?




We will use Python3 (current version is 3.9): be careful when looking around, Python2 is still very common (but deprecated) and incompatible. Python supports different "paradigms", we will focus on:

- Imperative programming: programs describe changes in *registers* and the *executing environment*;
- Object-oriented: complex (imperative) programs are organized around objects in order to hide and isolate complexity.

This is a programming course: I will try to propose example that I believe could be useful in your daily practice, but I'm not a biologist.

# Programming




Programming in science can serve two (almost opposite) goals:

1. Understanding every detail of a computational process;
2. Compose computational process by assembling powerful build blocks of which you understand very little.

Most of the current popularity of programming is related to goal 2. . . with many *sorcerer's apprentices*. But this course will focus mainly on goal 1. In the last part of the course we will bend towards 2, hopefully with a solid background.

Programming can be both hard and addictive: Teach Yourself Programming in Ten Years

# Fundamental concepts of Python




The programmer describes computational processes in terms of:

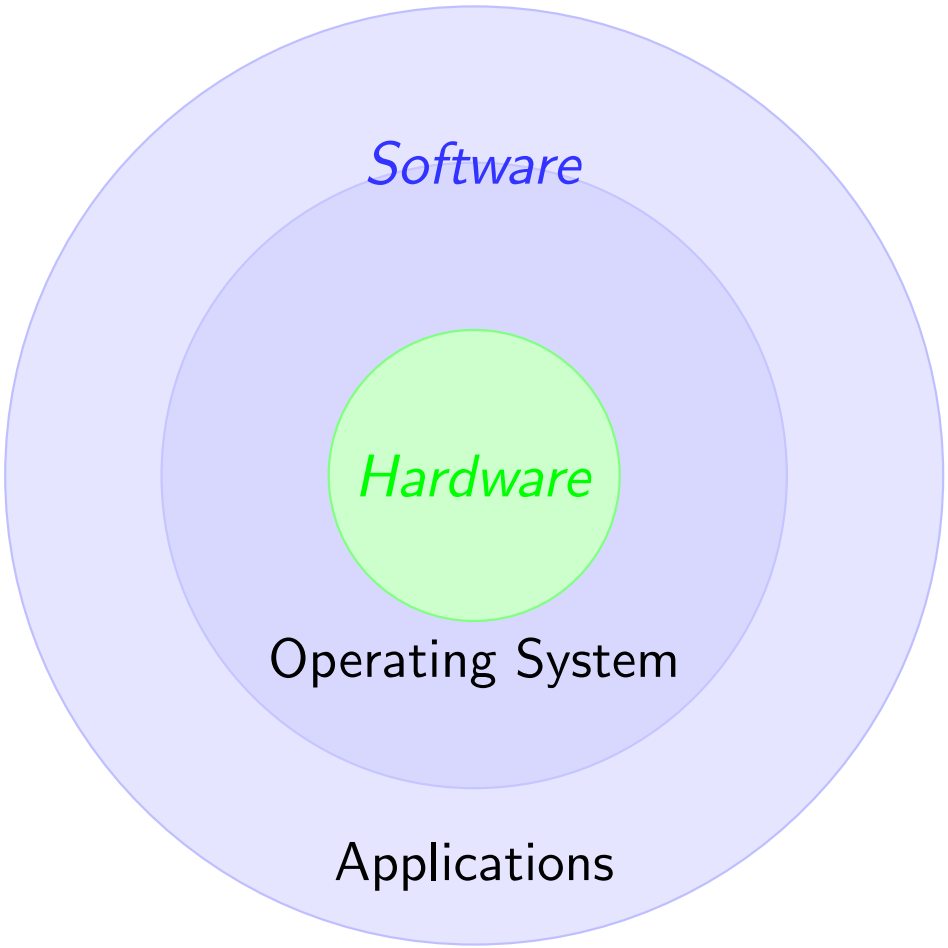
objects : all the entities manipulated by the program, each has an identity (can be distinguished) and a value, that is an element in a specific type (a set of values together with the operations that make sense on them)

basic types : integers (`int`), floats, strings (`str`), functions; they can be composed in more complex types

variables : names used to refer to objects; the same name can refer to different objects during the same process

special commands : the only way to change the execution environment (i.e., the "virtual machine" provided by the operating system) is to use system calls; syscalls change from system to system (e.g., Linux vs. Windows), but Python wraps them and they appear like the functions written by the programmers (e.g., `print`), even if they could not be programmed in Python.

# The onion model




- Operating System: it is the only program interpreted directly by the hardware; other pieces of software get interpreted by the virtual machine provided by it.
- Applications: programs (e.g., the python interpreter or python programs) executed within the protected environment created by the operating system.

# Let's try!



https://python.di.unimi.it/

You can use it without any personal account, but if you want support you must create one, putting me as the "guru": `mmonga`

This platform will be used for the first lessons, since it requires no setup at all: everything happens in the browser (and the server).
(Thanks to the University of Waterloo, Canada for providing the CS Circles)

# Assignment




This is the fundamental statement for imperative programming:

- A name, known as variable, is needed to refer to objects.
  `professor = "Mattia"`
- = is not symmetrical, read it as becomes: Left-hand-side is always a variable, right-hand-side is an object, that can be either a literal or anything referred by another variable.
- A variable can change its value with another, following, assignment. Thus, the same variable may refer to different objects.
  `professor = "Violetta"`
- Basic objects (numbers, strings, Boolean values) are immutable (the variable change, not the object; different objects have always different identity)
- Tracking a program means to track the values of all the variables of a program during its execution.

# Basic operations



- Binary operators: `5 + 2`, they compute a new object by using the two objects on which they apply;
- Unary operators: `-(-5)`;
- Functions: `max`, they compute a new object by using an arbitrary number of objects (in general 0–..., `max` takes at least 1) passed as parameters (or arguments) when the function is called (`max(3, 6, something_else)`); sometimes the object computed is `None`;
- Syntactically appear as functions, but *commands* like `print("Hello!")` are actually used to request side effects in the executing environment.

Documentazione ufficiale di Python (3.9)

# Different approaches


Problem: exchange the name of two objects (Chapter 1, last exercise).

- Know the basic syntax of variables and assignment =
- Know the semantics of what you write: assigning an object to a variable delete any previous assignment;

# Different approaches


Problem: exchange the name of two objects (Chapter 1, last exercise).

- Know the basic syntax of variables and assignment =
- Know the semantics of what you write: assigning an object to a variable delete any previous assignment;
- Natural strategy: use a temporary name to "save" the value during the exchange;

# Different approaches


Problem: exchange the name of two objects (Chapter 1, last exercise).

- Know the basic syntax of variables and assignment =
- Know the semantics of what you write: assigning an object to a variable delete any previous assignment;
- Natural strategy: use a temporary name to "save" the value during the exchange;
- "Fox" strategy: know language or library tricks For example Python has a "multiple assignment" construct `x, y = y, x`, or a special library function `swap(x, y)` could exist;

# Different approaches




Problem: exchange the name of two objects (Chapter 1, last exercise).

- Know the basic syntax of variables and assignment =
- Know the semantics of what you write: assigning an object to a variable delete any previous assignment;
- Natural strategy: use a temporary name to "save" the value during the exchange;
- "Fox" strategy: know language or library tricks For example Python has a "multiple assignment" construct `x, y = y, x`, or a special library function `swap(x, y)` could exist;
- "Hedgehog" strategy: study the problem in depth, e.g., if objects are numbers you can exploit arithmetic.

```
x = x + y
y = x - y
x = x - y
```

# Homework




Finish chapters 1, 1E, 2, 2X, 3, 4.
It shouldn't take more than three hours (one hour per day...), but exercising is crucial.