Python programming

Lorenzo Stacchio

PhD student in Computer Science

Department for Life Quality Studies

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Computer Science basics: fast recap

Lorenzo Stacchio

PhD student in Computer Science

Department for Life Quality Studies

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Computer types (almost)

Mainframe: is a type of computer characterized by high-level data processing performance of a centralized type, therefore opposed to that of a distributed system such as a computer cluster. Typically they are present in large computer systems such as data centers or organizations (public and private) where high levels of multi-user, huge volumes of data, large processing performance, combined with high reliability are required. (Wikipedia)

A personal computer lends itself to its own personal use and customization by the user in everyday use. At the hardware level it follows the von Neumann architecture and is composed of a central CPU processing unit, a central memory unit (RAM) and finally a data storage unit (Disk). (~ Wikipedia)

Von Neumann architecture
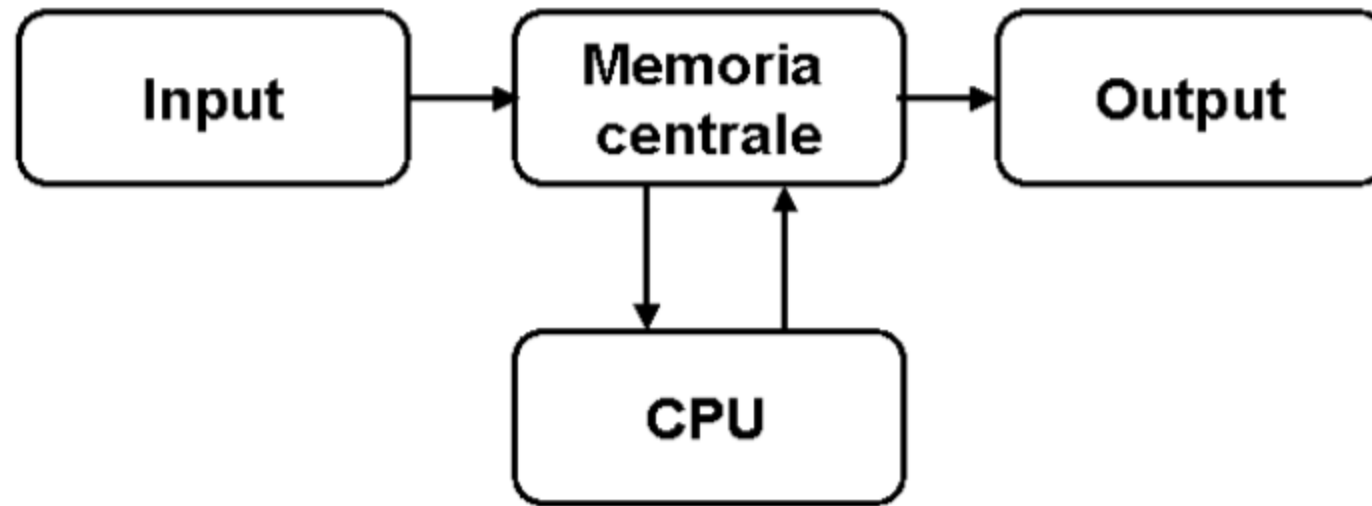
- In all types of electronic computer, we always have one (or more) computing unit (CPU, Central Processing Unit), a central memory that we can think of as a huge record that contains the binary numbers to be processed where the CPU picks up and writes data, and devices that allow the user to enter data (Input) and read results (Output;

- Typically, the input is associated with the peripherals for the use of the computer (hard disk, mouse, keyboard etc.); the output is instead often defined as the way to view the results of the actions we are performing.

Software manipulates data
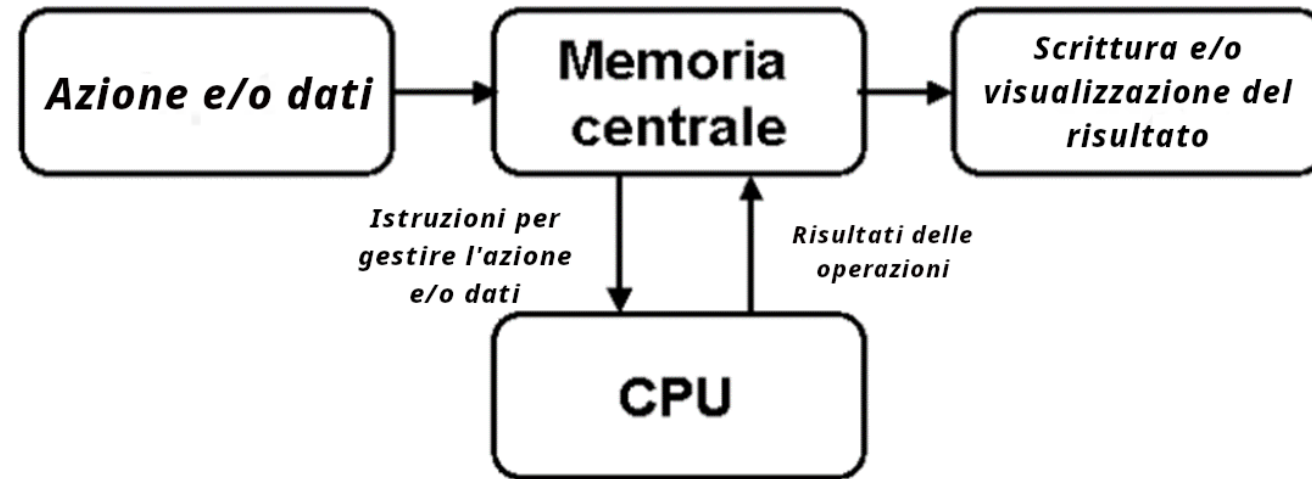
- **Software can be defined as a way to manipulate and transform data:**
    - A touch of the mouse changes the window we are using;
    - Writing on a text sheet changes the overall representation of the text;

- **Analyzing data means transforming that data in such a way as to obtain the answers we seek:**
    - Statistics (mean, variance etc.);
    - Profiling of an account based on the actions that the user performs;
    - Classification of an animal based on the pixels of an image.

Automatic calculus and binary encoding

- The computer is an automatic information processor, which performs operations on objects (data) to produce other objects (transformed data or results);

- The computer we have described is however "electronic" in that the objects it receives at the input, processes and supplies at the output are electrical signals, voltage levels and their processing takes place by means of semiconductor electronic circuits;

- However, this definition can be relaxed with a digital computer: the meaning attributed to the voltage levels is not linked, in fact, to their absolute value, but to their being higher or lower than a certain threshold, in which case the value 1 or 0 is attributed. The input and output data in electronic computers are therefore sequences of 0 or 1 values (bit, from BInary digiT);

- The execution of actions is requested from the computer through suitable directives, called instructions, coded in binary code;

- All computers, therefore, perform operations on binary sequences called strings.

- Numbers, letters, images and all the information that we want to automatically process are also commonly represented by sequences of symbols belonging to an alphabet that is much larger than 2.

- The case of binary symbols simply represents the minimal case, that is the number minimum of symbols sufficient to represent information.

28 x 28
784 pixels

Image pixels come byte
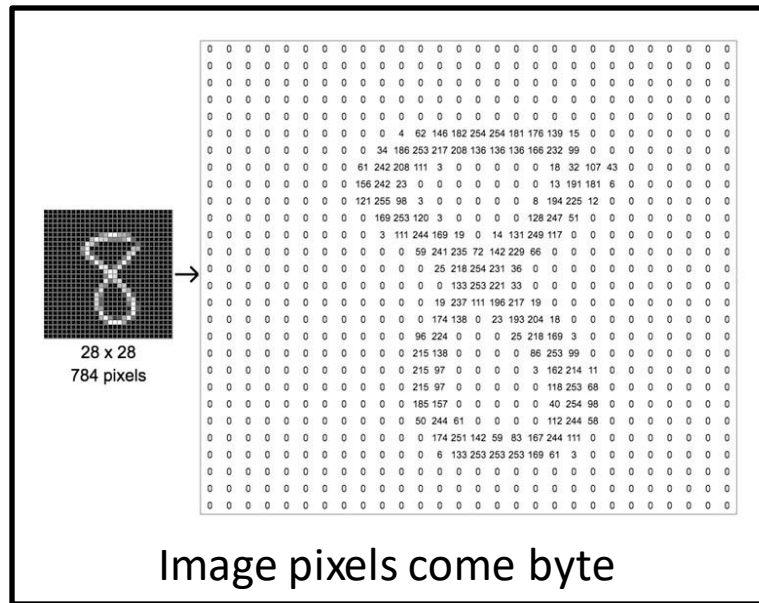
Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

Caratteri alfabetici in bit

- Computer memories, therefore, store binary sequences. To give orders of magnitude of the quantity of information, the various multiples of the bit are used:

Memory capacity hierarchy and conversion chart

UNIT	ABBREVIATION	APPROXIMATE SIZE
bit	b	Binary digit, single 1 or 0
byte	B	8 bits
kilobyte	KB	1,024 bytes or 10^3 bytes
megabyte	MB	1,024 KB or 10^6 bytes
gigabyte	GB	1,024 MB or 10^9 bytes
terabyte	TB	1,024 GB or 10^{12} bytes
petabyte	PB	1,024 TB or 10^{15} bytes
exabyte	EB	1,024 PB or 10^{18} bytes
zettabyte	ZB	1,024 EB or 10^{21} bytes
yottabyte	YB	1,024 ZB or 10^{24} bytes

Software and algorithms

- So far, we have treated software simply as one or more procedures for transforming data;

- However, it seems that these procedures take place in a chaotic manner, without a precise order and purpose. In fact, this is the most wrong interpretation we can give to software;

- Modern software typically consists of a set of algorithms;

- An algorithm is defined as the sequence of instructions that describes how, starting from a certain input, it returns a certain output;

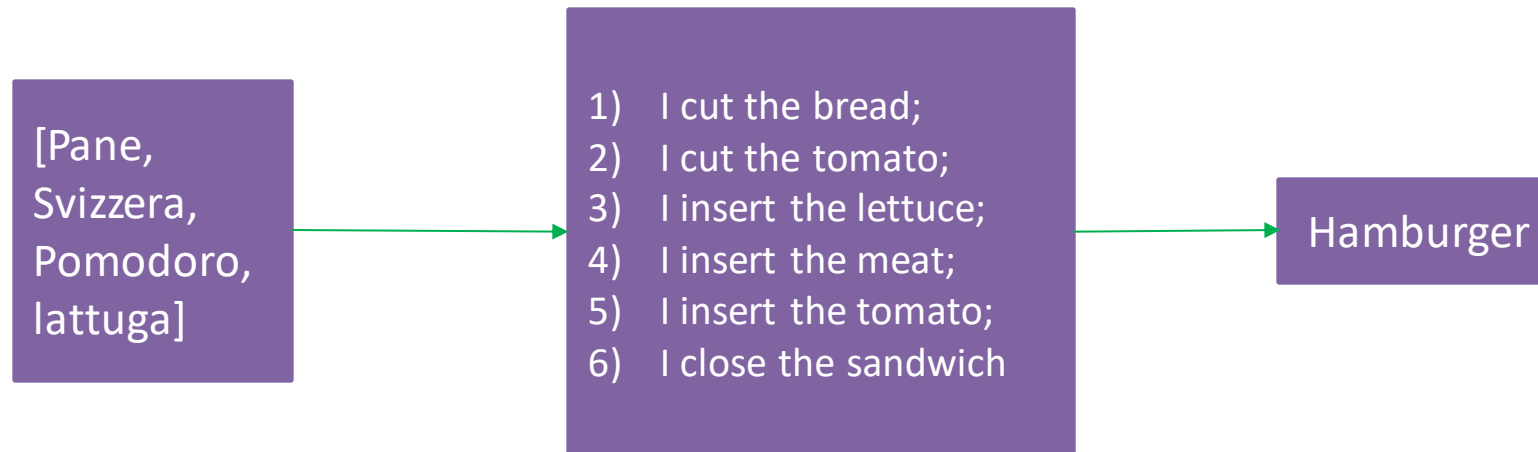| Input	→	Process	→	Output

Deterministic algorithms

- We can compare an algorithm to a recipe. For example, when preparing a sandwich, a series of steps are followed: the ingredients are taken, a slice of bread is placed on top, and another slice of bread is placed on top. If we were to formalize the preparation of a sandwich algorithmically:

| Ingredients	→	Cook	→	Sandwich

- An algorithm is said to be deterministic if and only for the same type of input the same output is produced. If I decide to use bread, Swiss meat, tomato and lettuce and assemble them, I will only be able to produce a hamburger (I certainly don't make a pizza, at least in most countries of the world ☺).

[Pane, Svizzera, Pomodoro, lattuga]	→	1) I cut the bread; 2) I cut the tomato; 3) I insert the lettuce; 4) I insert the meat; 5) I insert the tomato; 6) I close the sandwich	→	Hamburger

Deterministic algorithms and equal opportunities

- However, it is not certain that the same output cannot correspond to two different inputs;

- Let's take as an example an algorithm that defines the sum of two integers:

| 3,6 | → | Sum | → | 9 |

| 6,3 | → | Sum | → | 9 |

Commutatività
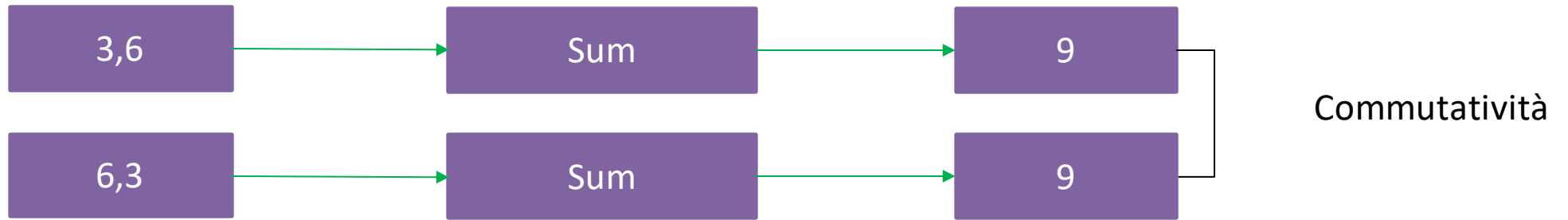
- We define a "Discriminator" algorithm that returns 0 if it receives a number lower than or equal to 10 and 1 if it receives a number greater than 10;

| 2 | → | **Discriminator** | → | 0 | | 10 | → | **Discriminator** | → | 0 |

| 524 | → | **Discriminator** | → | 1 |

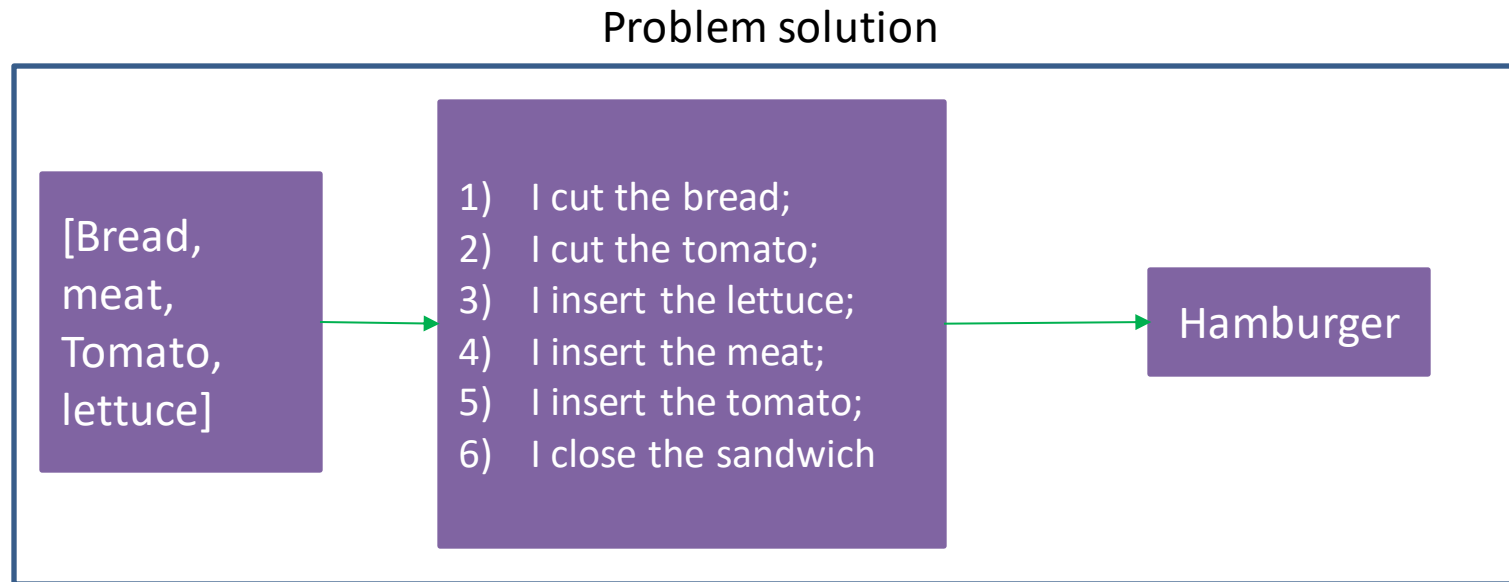- **This phenomenon can be obtained using the flow control operators;**

Non deterministic algorithms

- An algorithm is said to be non-deterministic if and only for the same type of input different outputs can be produced. An algorithm can exhibit nondeterministic behaviors in several ways:

  - Making use of probabilistic functions;
  - If it has a concurrent execution nature (we will discuss this in the lesson dedicated to Big Data paradigms);

- Ad esempio, definiamo un algoritmo non deterministico, che per un qualunque numero intero **n**, restituisca un numero decimale casuale tra **[0,n].**
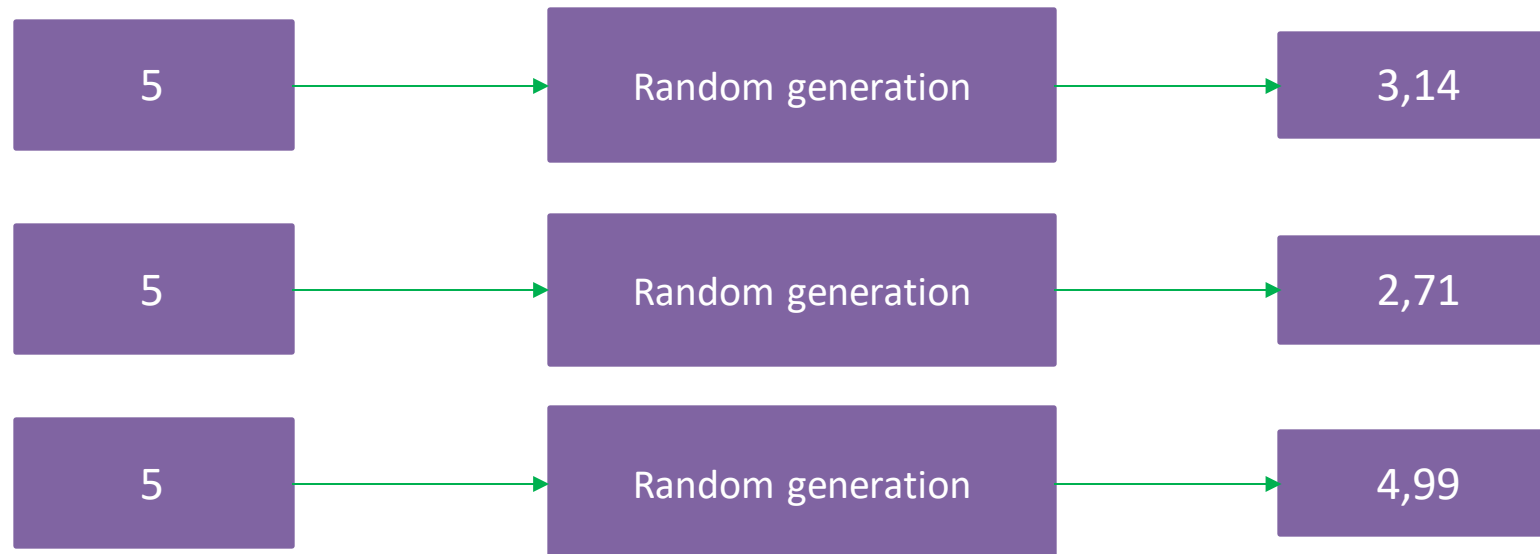
5	→	Random generation	→	3,14
5	→	Random generation	→	2,71
5	→	Random generation	→	4,99

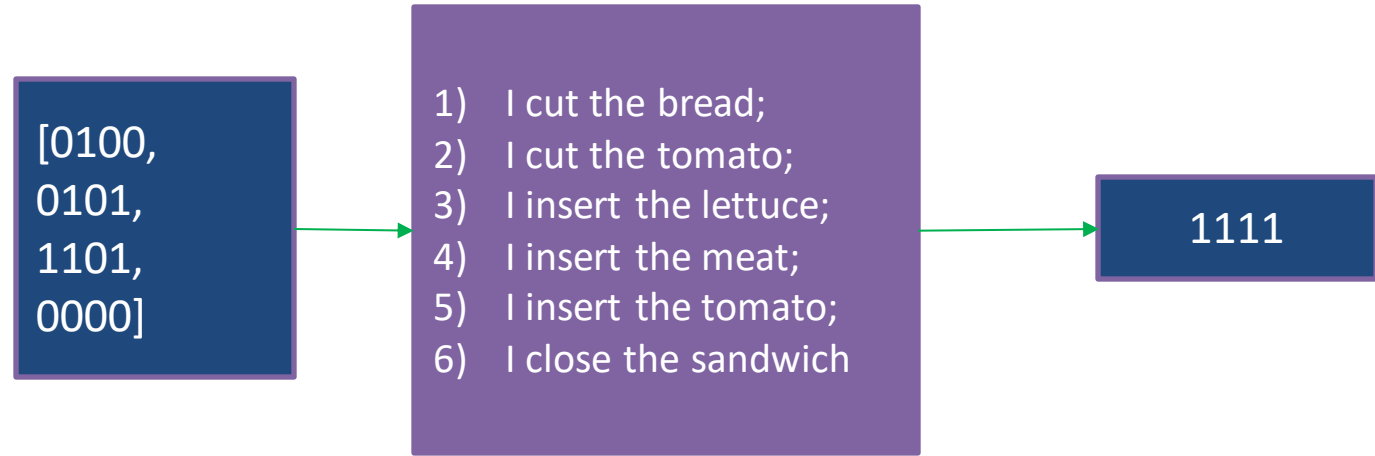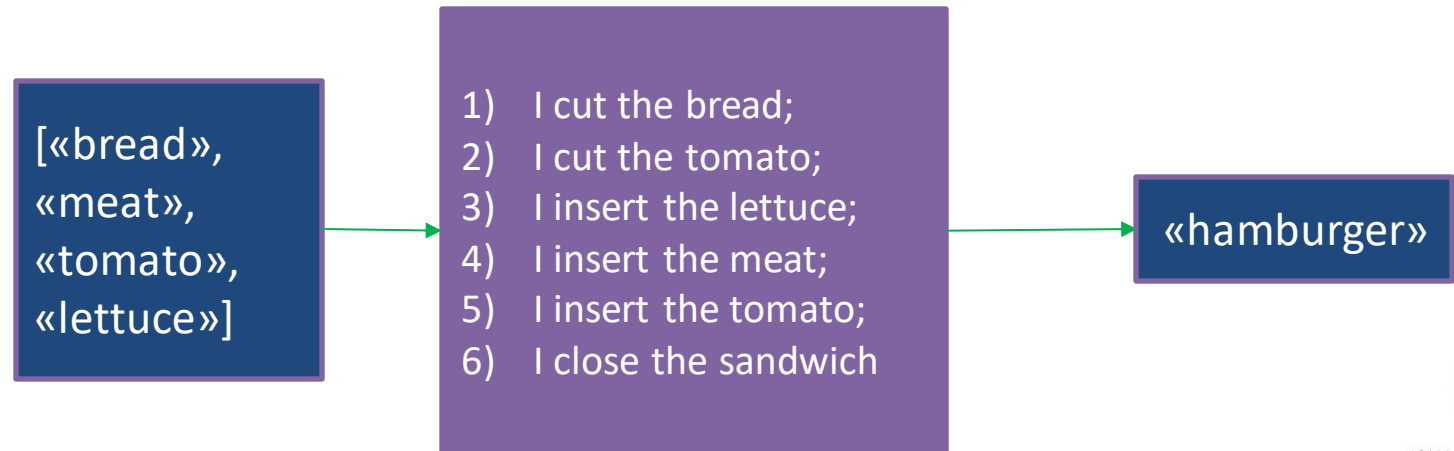# Algorithm and problem solving

- Suppose we have a certain problem and want to solve it algorithmically. To do this, some conditions must be verified:
    - The solution to the problem must be known;

Problem solution

[Bread, meat, Tomato, lettuce]

1) I cut the bread;
2) I cut the tomato;
3) I insert the lettuce;
4) I insert the meat;
5) I insert the tomato;
6) I close the sandwich

Hamburger

Algorithm and problem solving

- Suppose we have a certain problem and want to solve it algorithmically. To do this, some conditions must be verified:
    - The solution to the problem must be known;
    - Input data should be codified in binary (or their representations);

Binary data

[0100,
0101,
1101,
0000]

→

1) I cut the bread;
2) I cut the tomato;
3) I insert the lettuce;
4) I insert the meat;
5) I insert the tomato;
6) I close the sandwich

→

1111

Strings (binary representation)

[«bread»,
«meat»,
«tomato»,
«lettuce»]

→

1) I cut the bread;
2) I cut the tomato;
3) I insert the lettuce;
4) I insert the meat;
5) I insert the tomato;
6) I close the sandwich

→

«hamburger»

Algorithm and problem solving

- Suppose we have a certain problem and want to solve it algorithmically. To do this, some conditions must be verified:
  - The solution to the problem must be known;
  - Input data should be codified in binary (or their representations);
  - The steps that provide the solution should be codified in a language that our computer can understand.
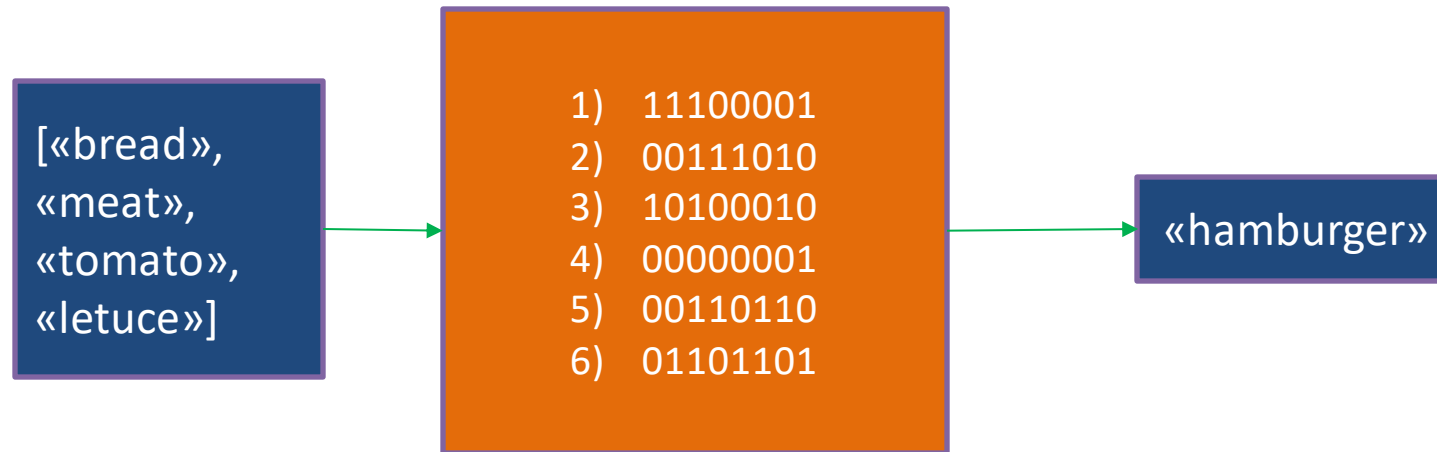
Algorithms must be understandable by a computer

- A computer, by itself, comprises only a series of instructions written for the microprocessor and encoded in what is called "machine language", that is, a series of operation codes (a sequence of bytes) and operands (data also written in binary) and is able to work on numerical data of limited type and size;

- Although machine language is not strictly binary, programming a modern application working at this level would still have prohibitive complexity;

- Fortunately, programmers don't need to know either binary or machine language;

- In fact, there are so-called "high-level" languages that contain constructs that resemble natural language and through which algorithms can be written which will then be transformed entirely into machine language by the so-called "compilers", or controlled and interpreted line by line by programs that are called "interpreters".

- There are therefore compiled languages (e.g., C, C ++, C #), both compiled and interpreted languages (e.g., Python, Java);

Algorithms for problem solving: recap
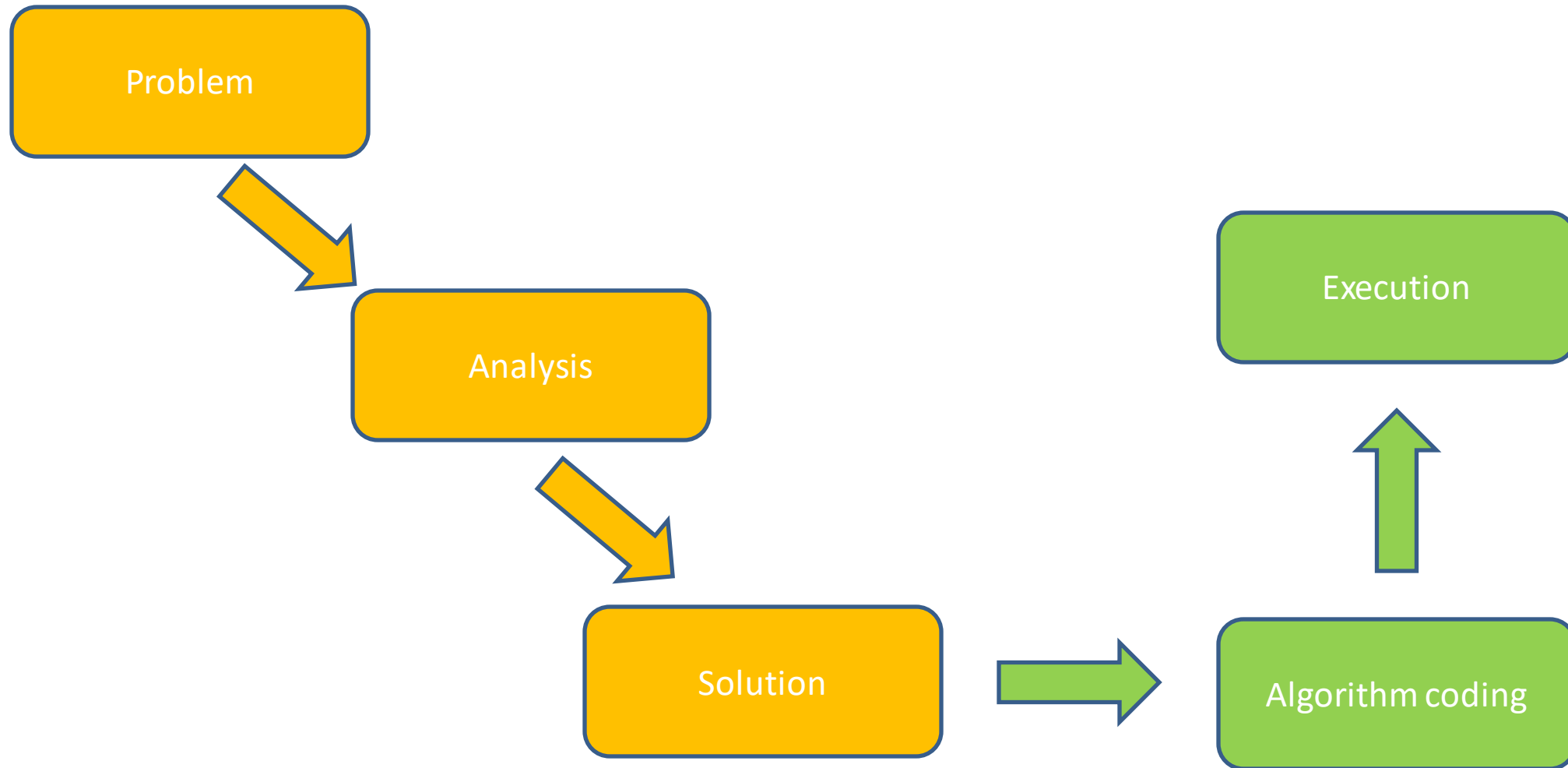
Python study material

Python per tutti

Esplorare dati con Python 3

Charles R. Severance

Pensare in Python

Come pensare da Informatico

Seconda Edizione, Versione 2.2.23

Why python?

- Guido van Rossum, inventor of python, was fond of the BBC comedy series "Monty Python's Flying Circus" (1970s). Van Rossum therefore thought of a short, unique and mysterious name for his new language: Python;

Quoting Allen Downey:

- "Initially we taught C ++ to first year students ... two years later I was convinced that C ++ was an unsuitable choice to introduce students to computer science: while on the one hand C ++ is certainly a very powerful language, it nevertheless proves to be be extremely difficult to teach and learn. "

- "I was constantly struggling with the difficult syntax of C ++ and was also unnecessarily losing many of my students. [...] "

- "I needed an easy-to-teach and learn language that could run on our lab's Linux machines as well as on Windows and Macintosh systems."

- "It had to support procedural programming as well as other paradigms"

- "I wanted it to be open-source (free) and have an active developer community. Python seemed the best candidate. "

Python is a simple language

- As an insight behind the syntactic and development ease of python, let's see how to code the first instruction coded by every developer: the printing of the phrase "Hello World".

C++	Java	Python
```cpp #include <iostream> using namespace std; int main() { cout<<"Hello World"; return 0; } ```	```java public class Main { public static void main(String[] args) { System.out.println("Hello World"); } } ```	```python print("Hello world") ```

Say 'Hello World' in 28 Different Programming Languages

Our first python instruction

- Let's open our first Google Colab;

- Once opened and logged in with your google account, this sheet will become your personal copy!

- So, you can modify as you prefer!

Python is portable and interpreted

- The Python language is a portable language.

- The same Python code can be executed on Windows and other platforms such as Linux and Mac, without requiring any modification (with the same version and libraries installed);

- This is possible because the Python language, similar to Java but not to the same extent, is both a compiled and interpreted language and makes use of a Virtual Machine.

Understand Python Interpreter
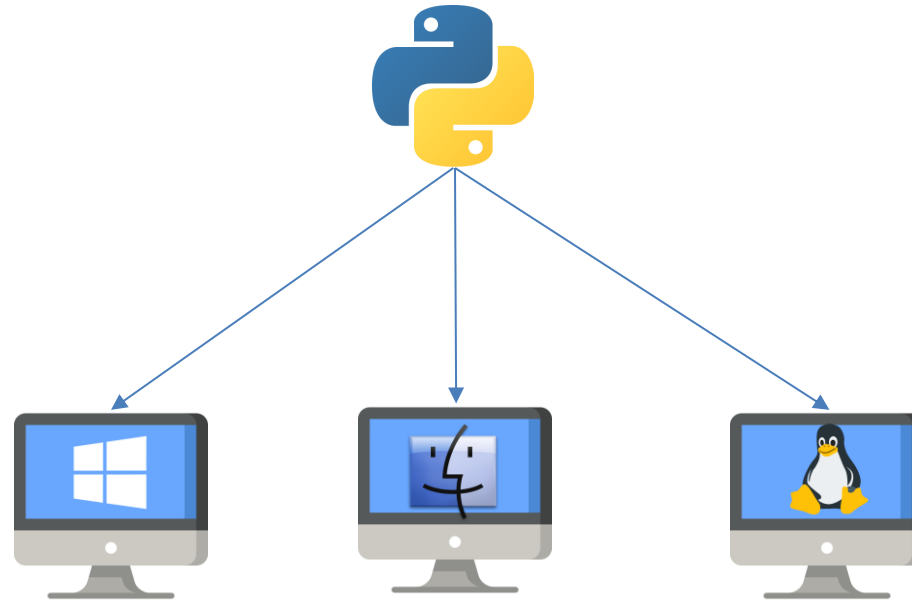
Compiled programming languages
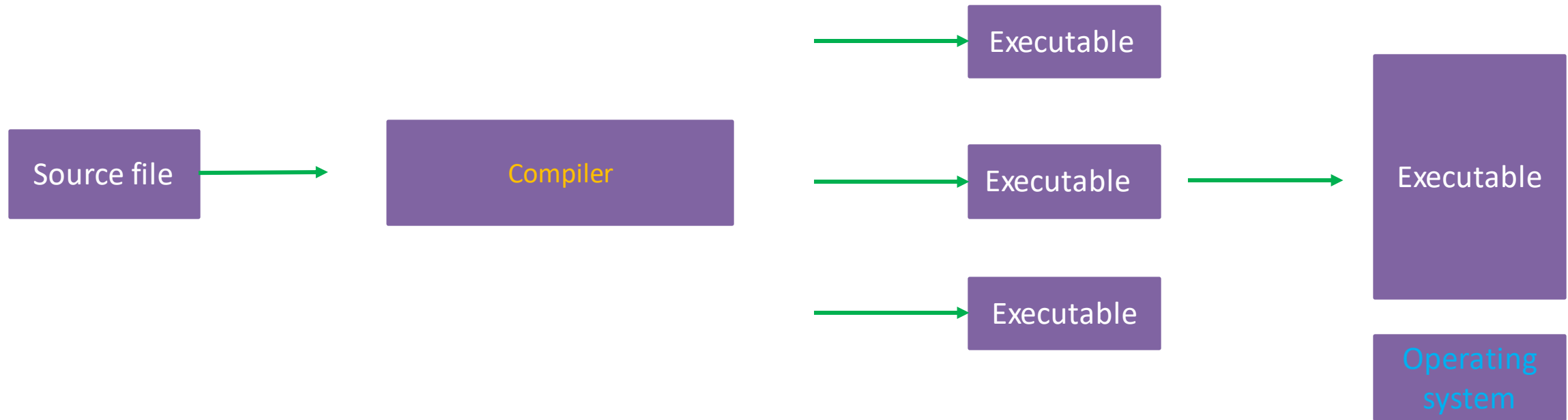
- The compiler translates a high-level code of the executable code for the physical architecture of the computer and for the operating system in which we find ourselves at a certain moment;

- The generated code is directly executable without further steps (efficiency);

- The compilation process occurs only once and concerns the entire source file;

- A compiled language will therefore have to be retranslated every time we change physical architecture and / or operating system.

Interpreted programming languages
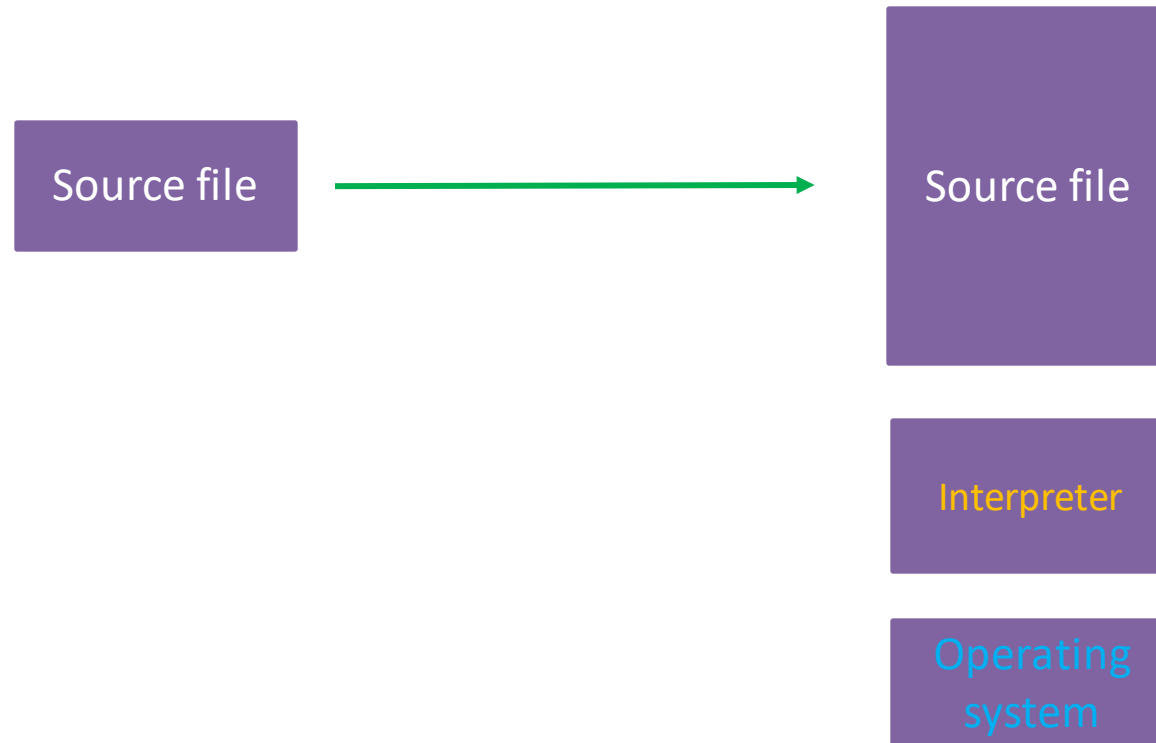
- The interpreter translates high-level code line by line for a certain architecture and operating system and executes it immediately;

- This has an advantage: the same source code can be executed on different platforms, because the interpreter will translate the source file according to the architecture!

- At the same time, however, the language is less efficient;

Compiled and interpreted languages

Source file	→	Compiler	→	Bytecode	→	Bytecode
						VM
						Operating system

Click to add text
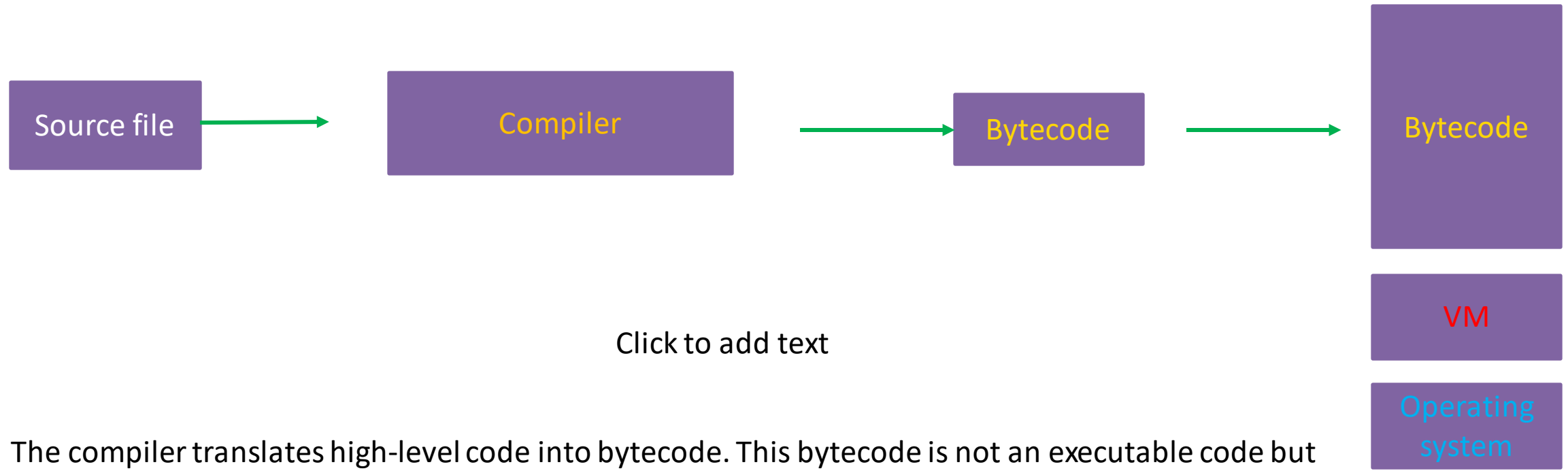
- The compiler translates high-level code into bytecode. This bytecode is not an executable code but it is an abstract and ideal encoding of the machine language (more easily interpretable).

- The Virtual Machine will then translate this bytecode into machine code for the reference architecture and operating system;

- We have combined the benefits of both worlds: we have avoided lengthy compilation for each architecture and reduced the inefficiency of a fully interpreted language!

Python has an enormous amount of Data Science libraries

Big Data & Data Science & Machine Learning

Deep Learning

Let's code!

Variables and operators

Values and types

- A value is one of the fundamental artifacts for a programmer; Bound

- A value can be defined as the representation of some entity that can be manipulated by a program; Bound

- The members of a certain type are the values of that type;

- So far, we have seen the value "Hello, World" which belongs to the **string type** (which represents a series of characters);

- The string type is identified as it is inserted between the two **""** and is fundamental, since it allows you to easily communicate with the user who is using the software;

- Clearly the string type is just a prime example of the types that we can find within the python ecosystem;
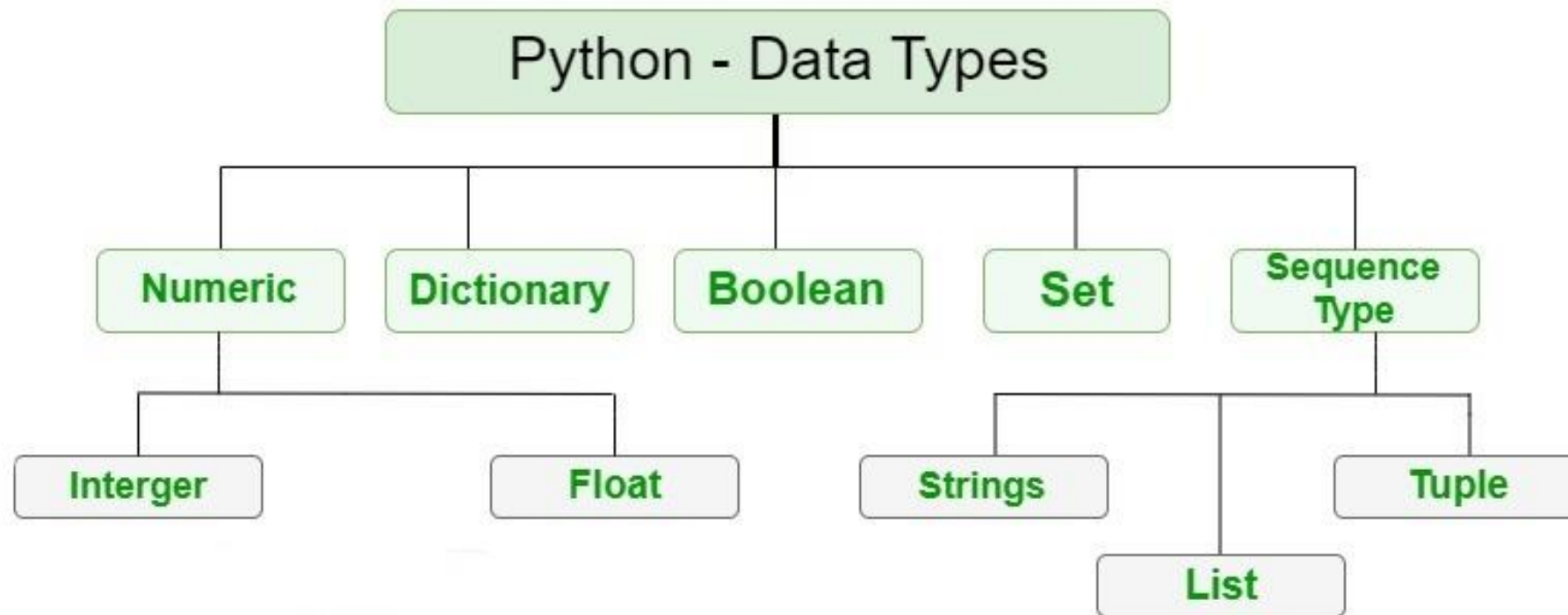
Type casting: explicit vs implicit

- The notebook shows that the sum between an integer and a float returns a float type;

- This is due to the mechanism called implicit type casting; This mechanism automatically rages and changes the type of values based on the operations we perform;

- For example, the sum between 3.5 and 2 must return 5.5 unless we want the integer part of the result;

- To get the whole part of the result we will have to carry out an explicit casting operation, that is, explicitly communicate to our interpreter that we want the sum to give us back an integer;

- This casting can be done for the primitive types seen in the previous slide but we must be very careful because we will not be able to explicitly convert all types to all other types;

Variables

- One of the most important features in a programming language is the ability to manipulate variables.

- **A variable is a name that refers to a value of a certain type.**

- Python, unlike other languages, is dynamically typed, so we can assign a value of a certain type to a variable and then change it whenever we want;

- Variables are named with the **snake_case** notation;

- We can use variables to perform the same operations we would do with normal values;

- **You won't be able to use Python's reserved keywords to name your variables;**

and	continue	else	for	import	not	raise
assert	def	except	from	in	or	return
break	del	exec	global	is	pass	try
class	elif	finally	if	lambda	print	while

Expressione evaluation and operators

- An expression is a combination of values, variables, and operators.

- Operators are special symbols that represent mathematical processing, such as addition and multiplication. The values the operator uses in calculations are called operands;

```
20+32    ore-1    ore*60+minuti    minuti/60    5**2    (5+9)*(15-7)
```

- The use of the symbols +, -, / and parentheses are the same as their use in mathematics. The asterisk (*) is the symbol of multiplication and the double asterisk (**) is the symbol of exponentiation.

- When a variable appears in place of an operand it is replaced by the value it represents before the operation is performed.

Boolean operators

- In all programming languages there are Boolean operators, that is operators that allow to formalize the basic logical constructs.

- We will only look at a few constructs, which will be especially useful when we talk about flow control operators;

- They all have the same priority and the operations are performed from left to right;

p	q	not p	not q	p and q	not (p and q)	p or q	not (p or q)
T	T	F	F	T	F	T	F
T	F	F	T	F	T	T	F
F	T	T	F	F	T	T	F
F	F	T	T	F	T	F	T

https://en.wikipedia.org/wiki/Truth_table

Operatori booleani matematici

- We can also compare the values between two numbers!

p	q	p > q	p >= q	p < q	p <=q
5	7	F	F	T	T
7	7	F	T	F	T
7	9	T	T	F	F

When a value is False in Python

- *None* is given to represent nothing;

- *False*

- A *zero* of any kind → 0, 0.0

- *An empty string "''"*

- *Any empty sequence* → (), []

- *Any empty dictionary* → {}

} The sequence and dictionary elements are part of the data structures, which we will see in a while

If it does not belong to one of these categories, a data is considered TRUE for the Boolean logic implemented in Python

String expressions

- In general, you cannot perform mathematical operations on strings, even if their contents appear to be a number. If we assume that the message is of type string, the examples given below are illegal:

<center>

`messaggio-1`   `"Ciao"/123`   `messaggio*"Ciao"`   `"15"+2`

</center>

- However, the + operator works as a string concatenator;

<center>

«banana» + «frutta» = «bananafrutta»

</center>

- The operator * also works as a string repeater;

<center>

«banana» * 3 = «bananabananabanana»

«banana» + «banana» + «banana» = «bananabananabanana»

</center>

Time for exercises

type() and print() functions

- Up until now we have used functions without knowing what they were;

- type () and print () are in fact two types of functions!

Let's analyze the "type" function with input "32"

- The name of the function is **type** and it returns the **type** of the number **32** (i.e., an integer).

- The value of the variable, which is called the function argument, must be enclosed in parentheses (in this case it is 32).

- It is common to say that a function "accepts" or "takes" an argument and "returns" or "returns" a result.

- The result is also called the return value of the function. In this case, it corresponds to the type of variable 32, ie an integer;

Let us now analyze the function print (32)

- The function name is print and allows you to print a value on the screen.

- The value, which is called the function argument, must be enclosed in parentheses (in this case it is 32).

- The print function does not return any value!

- In fact, if I were to print a print (inception) I would have as a result None, that is the value that represents nothing;

```
print(print(5))
```

```
5
None
```

Functions

- In general, therefore, a function:

  - **It has an identifying name;**

  - **Defines none, one or more arguments;**

  - **Returns none, one or more values;**

  - **It defines a series of instructions within it;**

Let's code!

Mathematical functions

- There are many Python libraries to perform mathematical operations;

- However, python itself exposes a module called Math which allows you to perform practically all known algebraic operations (and more);

- To call a function of a module we must specify the name of the module that contains it and the name of the function separated by a period. This format is called dot notation.

```
>>> decibel = math.log10 (17.0)
>>> angolo = 1.5
>>> altezza = math.sin(angolo)
```

```
>>> gradi = 45
>>> angolo = gradi * 2 * math.pi / 360.0
>>> math.sin(angolo)
```

https://en.wikipedia.org/wiki/Turn_(angle)

Libraries and modules

- Libraries are collections of code that programmers can use to optimize their activities (and not re-invent the wheel);

- We have re-implemented the addition, subtraction etc. ... but they were already present in the python language!

- You must know "if you think something trivial, probably someone will certainly have implemented it";

- A module is a code file designed to perform a certain behavior (math is a module of the python language);

- Often library and module are synonymous but are used in different contexts;

- What is certain is that both libraries and modules are made up of sub-modules;

- For example, the pandas library or module defines many sub-modules (which are always code files) that implement some operations in isolation;

Variable scope: local vs global

- The scope or scope of a variable is the part of a program within which it can be referred to.

- The variables declared within a function are called local to the function since they are accessible only within its body;

- Variables defined anywhere else in the code are called global, and can be accessed at any point;

- Variables therefore have a level of visibility: variables defined at level **x** are visible at level x but also at level **x+1, x+2** and so on.

- Let's see an example to understand this concept which is of fundamental importance in programming;

```python
# We are at level 1 of the visibility hierarchy
# globally scoped variable

global_variable = "global"

def examin_variable_scope():
    # We are at level 2 of the visibility hierarchy
    local_variable_1 = "local_1" # local scope
    local_variable_2 = "local_2" # local scope
    print("Local scope variables%s" % [local_variable_1,local_variable_2])
    print("Global scoper variables%s" % [global_variable])
```

Variable space

Name	Value	Scope level
global_variable	"global"	1

```
# Declare and assign variable
global_variable = "global"
```

Variable space

Name	Value	Scope level
global_variable	"global"	1

```python
# Declare the function
def examin_variable_scope():
  # We are at level 2 of the visibility hierarchy
  local_variable_1 = "local_1" # local scope
  local_variable_2 = "local_2" # local scope
  print("Local scope variables%s" % [local_variable_1,local_variable_2])
  print("Global scoper variables%s" % [global_variable])
```

Variable space

Name	Value	Scope level
global_variable	"global"	1
local_variable_1	"local_1"	2
local_variable_2	"local_2"	2

```
# Call the function
examin_variable_scope()
```

Variable space

Name	Value	Scope level
global_variable	"global"	1
local_variable_1	"local_1"	2
local_variable_2	"local_2"	2

```python
# Execute local assignments

local_variable_1 = "local_1" # local scope
local_variable_2 = "local_2"
```

Variable space

Name	Value	Scope level
global_variable	"global"	1

```
# Function ends
```

Shadowing di una variabile

- The scope or scope of a variable is the part of a program within which it can be referred to.

- We can obscure the visibility of lower-level variables by assigning different values to the same variable names;

- Let's see an example to understand the concept starting from the previous one;

```python
def examin_variable_scope():
    # We are at level 2 of the visibility hierarchy
    local_variable_1 = "local_1" # local scope
    local_variable_2 = "local_2" # local scope
    global_variable = "global_modified"
    print("Local scope variables%s" % [local_variable_1,local_variable_2])
    print("Global scoper variables%s" % [global_variable])
```

Variable space

Name	Value	Scope level
variabile_globale	"globale"	1

`examin_variable_scope()`

Variable space

The variable is not overwritten at a lower level!

Name	Value	Scope level
global_variable	"global"	1
global_variable	"global_modified"	2
local_variable_1	"local_1"	2
local_variable_2	"local_2"	2

A new one is created that has visibility only from that level onwards!

```
local_variable_1 = "local_1" # local scope
local_variable_2 = "local_2" # local scope
global_variable = "global_modified"
```

Data Structures

Data structures

- So far, we have talked about "simple" data types that represent "simple" values (e.g., numbers, strings);

- Data structures, on the other hand, are more "complex" data since they do not contain a value but a sequence of values;

- Data structures are also called sequences;

- E.g., A string is a sequence of characters;

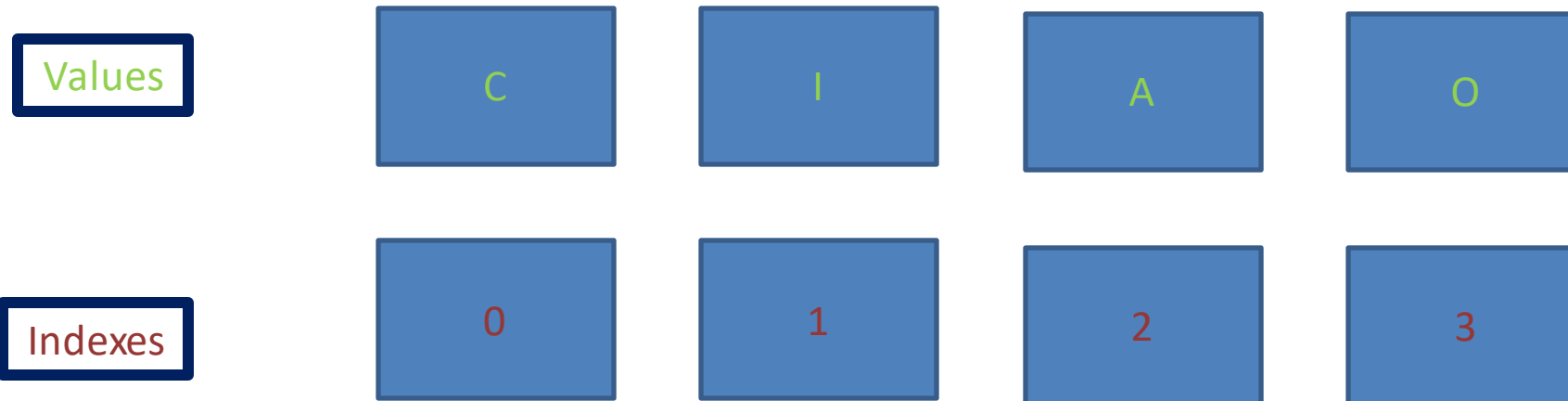- We will talk about the most known and used data structures;

List	Dictionary

Set	Tuple

Sequence

- A sequence, understood as a data structure, is always composed of three factors:

- A series of values;

- The index of each element;

- The number of elements this set contains;

- The string is a sequence of characters! Let's take the string "HELLO" as an example

Values	C	I	A	O
Indexes	0	1	2	3

Lists and tuples

- Both lists and tuples are sequences and can contain any type of elements (integers, characters, strings and etc...);

- A single list or a single tuple can also contain objects of different types (e.g., I can build a list with a character, an integer and a string)

Values	z	5	ciao
Indexes	0	1	2
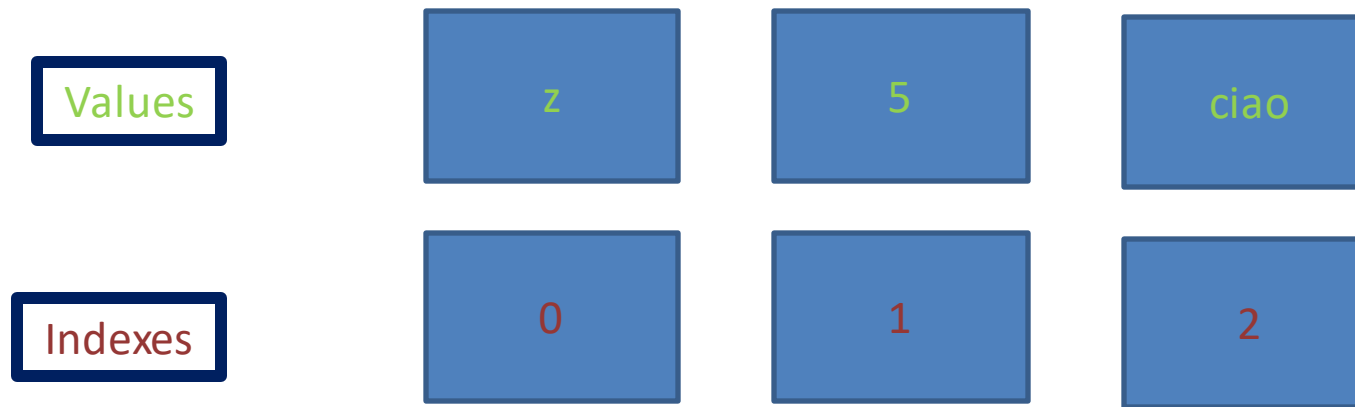
- However, they differ in one fundamental aspect:

  - The list is a mutable data structure, once defined it can be modified;

  - The tuple is an immutable data structure, once defined it cannot be modified;

Lists

- A list is a mutable data structure made up of 0 or more elements that are separated by commas and enclosed in a pair of square brackets;

```
empty_list = []
print(empty_list)

integer_list = [10,20,30]
print(integer_list)

another_empty_list = list() # this is a callable class as print() or type()
print(another_empty_list)

[]
[10, 20, 30]
[]
```

- As we have said, the sequences are composed of elements that are identified by indices:

  - If a list has n elements, I can use the numbers 0 to n-1 to access the various elements;

  - We can also index backwards using the negative sign (convenient when we want the last elements of the list);

```
integer_list = [10,20,30]

# I access the first and second elements, remember that we start from scratch!
print("Element with index 0 is:", integer_list[0])
print("Element with index 1 is:", integer_list[1])

Element with index 0 is: 10
Element with index 1 is: 20
```

List is a mutable data structure

- As anticipated, once defined, a list can be modified!

- There are several ways to edit a list:

  - We can change the value of an already present element;

  - We can add new elements;

  - We can remove elements;

- There are even different ways to do the same thing!

- One of the simplest ways to manipulate a list is through the use of the methods exposed by the List class!

Main methods to modify a list

- **Element** = value of any type;

- **Index** = value or integer variable used to manage the manipulation of values;

Method name	Description
append()	Adds an item to the end of the list
clear()	Removes all elements of the list
copy()	Return a copy of the list
count(element)	Counts the occurrences of elements within the list
extend()	Add any sequence to the list
index(element)	Returns the index of the first occurrence of the element
insert(element, index)	Add element at desired index position
pop(index)	Removes the element in the index position
remove(element)	Removes the first item value found
reverse()	Reverse the order of the list
sort()	Sort the list

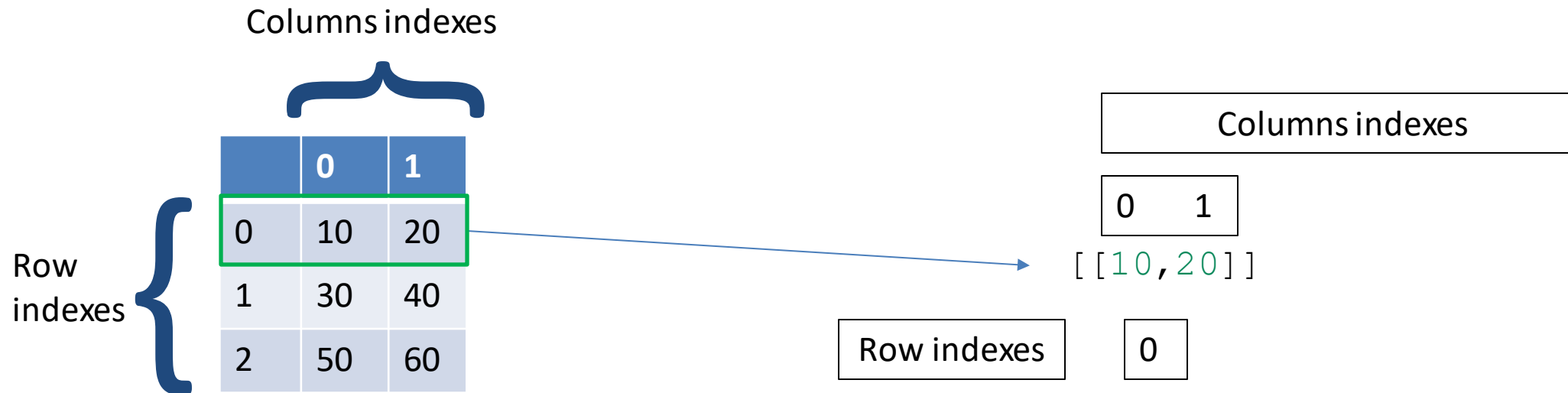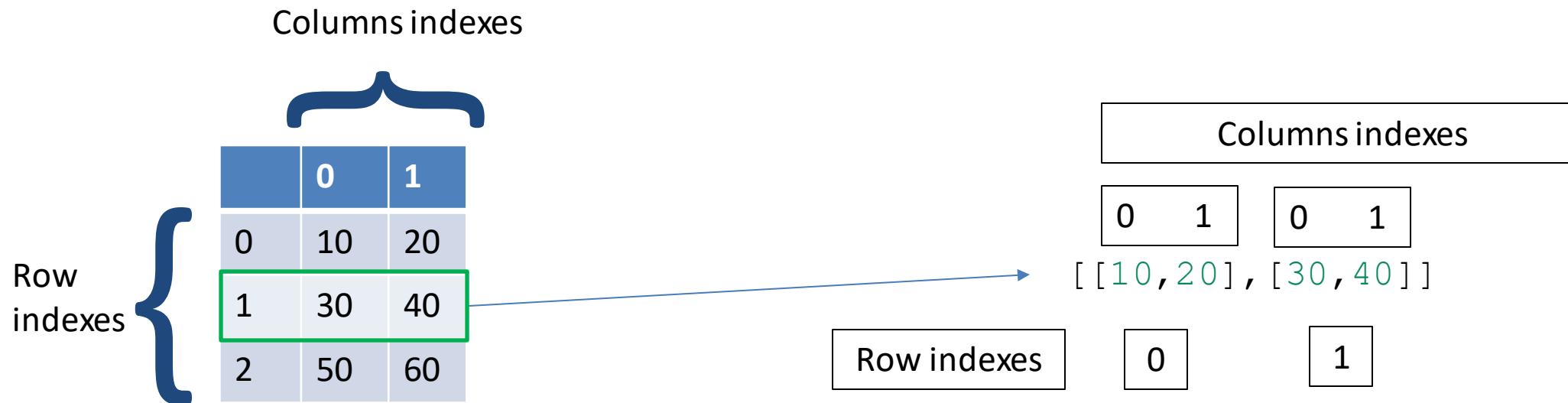ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Let's code!

Bidimensional list

- A two-dimensional list is a list made up of lists;

- We can interpret it as the squashed representation of a table made up of rows and columns;

Columns indexes

	0	1
0	10	20
1	30	40
2	50	60

Row indexes

Columns indexes

0   1

`[[10,20]]`

Row indexes   0

Liste bidimensionali

- A two-dimensional list is a list made up of lists;

- We can interpret it as the squashed representation of a table made up of rows and columns;

Columns indexes

	0	1
0	10	20
1	30	40
2	50	60

Row indexes

Columns indexes

| 0 | 1 | | 0 | 1 |

`[[10,20], [30,40]]`

| Row indexes | | 0 | | 1 |

Liste bidimensionali

- A two-dimensional list is a list made up of lists;

- We can interpret it as the squashed representation of a table made up of rows and columns;
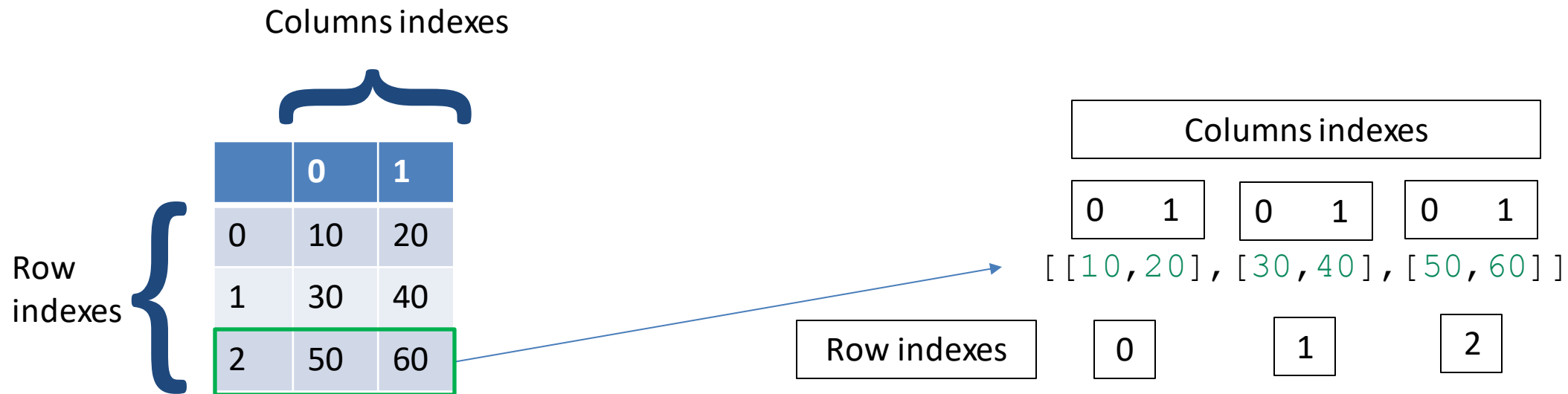
Tuples

- A tuple is an immutable data structure made up of 0 or more elements that are separated by commas and enclosed in a pair of round brackets;

```
empty_tuple = ()

print(empty_tuple, type(empty_tuple))

integer_tuple = (5,6,7)
print(integer_tuple)

another_empty_tuple = tuple()
print(another_empty_tuple)

() <class 'tuple'>
(5, 6, 7)
()
```

- Like lists, tuples also have elements that are identified by indexes. We can perform the same indexing operations seen with lists (e.g., Indexing to a specific item by index);

```
# possiamo indicizzare una tupla
integer_tuple = (5,6,7)

print(integer_tuple[0]) # prendiamo un elemento della tupla

print(integer_tuple[1:]) # slice di una tupla

5
(6, 7)
```

Tuples are immutable

- Once initialized, the tuples cannot be changed:

    - New values cannot be entered;

    - Values cannot be deleted;

    - Values cannot be changed;

Method name	Description
count(element)	Returns the number of times element appears in the tuple
index(element)	Search the tuple for the index of the first occurrence of an element

- Why use the tuple instead of a list? The tuple is optimized and is much more efficient than the list in terms of speed and memory usage!

Dictionary

- In Python, dictionaries are implemented by the Dict class;

- A dictionary is a mutable data structure made up of key-value pairs;

- The only immutable value of a dictionary is the key itself (to eliminate it, the key-value pair must be eliminated in its entirety!);

- All keys are unique to each other!

```python
empty_dictionary = {}

print(empty_dictionary, type(empty_dictionary))

dictionary_string_integers = {"one":1, "two":2}
print(dictionary_string_integers)

another_empty_dictionary = dict()
print(another_empty_dictionary)
```
```
{} <class 'dict'>
{'uno': 1, 'due': 2}
{}
```

- Like lists, tuples also have elements that are identified by indexes. In this case, however, the indices are not numeric, but are represented by the keys of the various pairs!

- There is no concept of slicing, but we can simulate it through the use of cycles (which we will see shortly)!

```python
dictionary_string_integers = {"one":1, "two":2, "three":3}
print(dictionary_string_integers["one"])
```

Dictionary is mutable and not sorted

- As we said, the only immutable value of a dictionary is the key;

- If we want to get rid of it, we need to delete the whole key-value pair, using the del command;

- We can change the value of an existing key-value pair at any time;

- **Two things should perplex you:**

  - **We have never used numerical indices → This means that there is no sorting, unless we define it explicitly!**

  - **Why did we use a string as a key? → We can use any value as a key, although I recommend using primitive values such as strings, numbers etc …**

Why use dictionaries?

- **Dictionaries are highly efficient and above all comfortable to use;**

- **Let's think about the case of user profiling:**

  - If I used a list or a tuple, I should know which index the user is in;

  - If I use a dictionary, I can index by its username!

Method name	Description
clear()	Removes all items from the dictionary
copy()	Return a copy of the dictionary
fromkeys((keys,values))	Returns a dictionary with specific keys and values
get(keys)	Returns the value of a specific key
items()	Returns a list of dictionary key-value tuples
keys()	Returns the list of dictionary keys
pop(keys)	Removes the key element
popitem()	Removes the last couple inserted
setdefault(keys, values)	Returns the value of the key and if it does not exist, inserts the key with the specified value
update((keys,values))	Updates the dictionary by adding the specified keys and values
values()	Returns a list of all dictionary values

Set

- In Python, sets are implemented by the Set class;

- A set is like a dictionary composed only of keys:

  - All values are unique and never repeated;

  - How dictionaries have no sorting!

  - We can use all the set operations we know from mathematics (e.g., intersection, difference)!

- Unlike a dictionary, we have no way to index values!

```
# empty_set = {} ---> NO! we creat an empty dict like this!

empty_set = set()
print(empty_set,type(empty_set))

integer_set = {1,2,3}
print(integer_set,type(integer_set))


set() <class 'set'>
{1, 2, 3} <class 'set'>
```

```
integer_set = {1,2,3}
print(integer_set[1]) # error

---------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-56-7e450c104977> in <module>()
      1 integer_set = {1,2,3}
----> 2 print(integer_set[1]) # errore

TypeError: 'set' object is not subscriptable
```

SEARCH STACK OVERFLOW

Code structures

Block of codes

- So far, we've been using blocks of code without knowing the nomenclature;

- A code block is simply two or more lines of code grouped with the same indentation level!

- An indentation level consists of 4 spaces!

4-line code block

1-line code block

```python
number1 = int(input('Insert first integer: ')) # cast to integer
number2 = int(input('Insert second integer: ')) # cast to integer


number_1_maggiore_number_2 = first_number_more_equal_number2(number1,number2)


if number_1_maggiore_number_2 == None:
    raise NotImplementedError("You don't have coded the function first_number_more_equal_number2")
```

Statement

- Individual code statements are referred to as simple statements

Simple statement

```python
number1 = int(input('Insert first integer: ')) # cast to integer
number2 = int(input('Insert second integer: ')) # cast to integer

number_1_maggiore_number_2 = first_number_more_equal_number2(number1,number2)

if number_1_maggiore_number_2 == None:
    raise NotImplementedError("You don't have coded the function first_number_more_equal_number2")
```

- Instructions that are broken down into multiple lines of code and logical lines are called compound statements

Composite statement

```python
number1 = int(input('Insert first integer: ')) # cast to integer
number2 = int(input('Insert second integer: ')) # cast to integer

number_1_maggiore_number_2 = first_number_more_equal_number2(number1,number2)

if number_1_maggiore_number_2 == None:
    raise NotImplementedError("You don't have coded the function first_number_more_equal_number2")
```

The compound statement If: the conditional control operator

- The if-elif-else construct verifies the truthfulness of one or more Boolean expressions;

- **This is a control operator since based on the veracity of a Boolean expression we may or may not execute a different block of code!**

- If expression takes the **True value**, then we will execute code block 1
  otherwise

  |
  |
  V

- If expression_2 took the value True, then we would execute code block 2
  otherwise

  |
  V

- …

- If no expression were to occur true, we would execute code block n

If expression:
       code block 1
Elif expression_2:
       code block 2
Elif … :
       code block …
Else:
       code block n

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Let's code!

Control operator

- A control operator can control the flow of the program;

- So far, we have declared instructions that were carried out one after the other;

- **From now on we could decide to control the flow of execution by allowing:**

  - The conditional execution of a line or block of code;

  - The repeated execution of a line or a block of code until an expression is verified;

  - The repeated execution of a line or a block of code for a defined number of times;

  - The execution of a change on the elements of a data structure (scrolling);

} CICLI

The Loop Wars

If you only knew the power
of the dark side.

- A loop can be defined as one or more lines of code executed until an exit condition occurs;

- They are useful because they allow you to create a compressed representation of entire blocks of code;

The Loop Wars Episode 1: The While Construct

```
while True:
    code block
    if expression:
        break
code block
```

- **A while loop executes one or more lines of code as long as a Boolean expression remains True;**

- **Basically, we could build an infinite cycle, creating a program that could hypothetically last until the end of time ...** could you tell me how?

- In the while loop it is allowed to use the conditional if construct to create an exit condition using the break construct;

The Loop Wars Episode 2: The for Construct

```
For element in sequence:
            code block
        if expression:
                    break
```

- **A for loop executes one or more lines of code for each element of a sequence (eg strings, lists);**

- **Typically used to carry out manipulations on the elements of the sequence or actions based on the value of the element;**

- **Also in this case we can use the conditional control operator and exit the loop (it doesn't happen often);**

The Loop Wars Episode 3: cycle for n times

- The range function **(start, stop, step)** allows you to iterate over a sequence of integers, which start from start and end in stop, each time making a jump equal to steps;

- However, **stop** is considered non-inclusive!

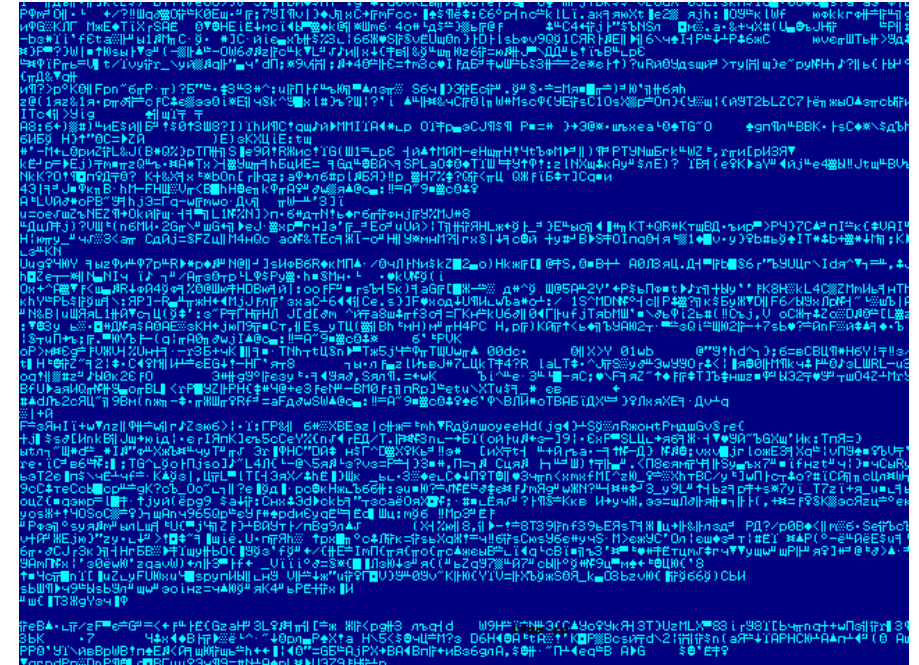- For example, calling the range (2,10,2) function would create the following sequence:

  [2,4,6,8]

- Calling the range function (2,11,2), would create the following sequence:

  [2,4,6,8,10]

How programmers would be without loops

Objects

Python is an object-oriented language

- Python is a language that not only allows you to define software defined by objects!

- Object-oriented programming is used intensively... for reasons we won't explain;

- We are not here to introduce the entire object-oriented programming paradigm but only to make you understand what an object is;

- This is because during the course of the next lessons we will use Classes, objects and methods without fully knowing their implementation!

What is a class?

- A class is made up of a set of attributes and a set of methods;

- The attributes are typically called "data of a class", that is, data that somehow together represent an entity;

- The methods are used to check the status of the object!

Class person:
    name;
    surname;
    age;
    fiscale code;

What is an object?

- An object is nothing more than an instance of a class;

- An instance of a class is nothing more than the class itself, but it is supplied with values for its attributes!

person_lorenzo = person(lorenzo, stacchio, 24, …)

Attributes passed as if they
were function arguments!

What is a method?

- Entering values within the attributes defines an object with a certain status!

- However, we need a construct that defines a way to manipulate this state!

- This construct is precisely the methods!

- Methods can be simplistically defined as functions of an object that change its state!

```
Class person:
        name;
        surname;
        age;
        fiscal_code;

        def birthday():
          if today== birthday:
                age +=1
```

Why did I tell you about the objects?

- All the tools present in the various python libraries for data science will expose objects and their methods;

- From your point of view, it will be no different than calling a function but it is good that you know how certain mechanisms under the box work, to facilitate the detection of bugs and facilitate the correction of the code!

- In the next lessons, we will see the logistic regression implemented in sklearn, a python library for machine learning;

```python
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression

X, y = load_iris(return_X_y=True)
clf = LogisticRegression(random_state=0).fit(X, y)
clf.predict(X[:2, :])
clf.predict_proba(X[:2, :]) clf.score(X, y)
```

This is an object!
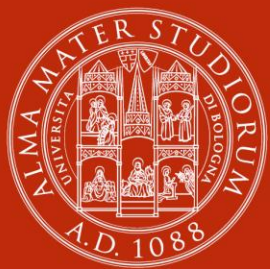
Now, we can create a mini Eliza!

Design a little Eliza

- What should a chatbot designed for e-commerce have?

    - Must manage basic interactions with the user;
    - It must show the user what is in the shop;
    - Manage any purchases.

- What tools seen during this lesson can we use to build a simple chatbot?

    - Variables;
    - Control operators;
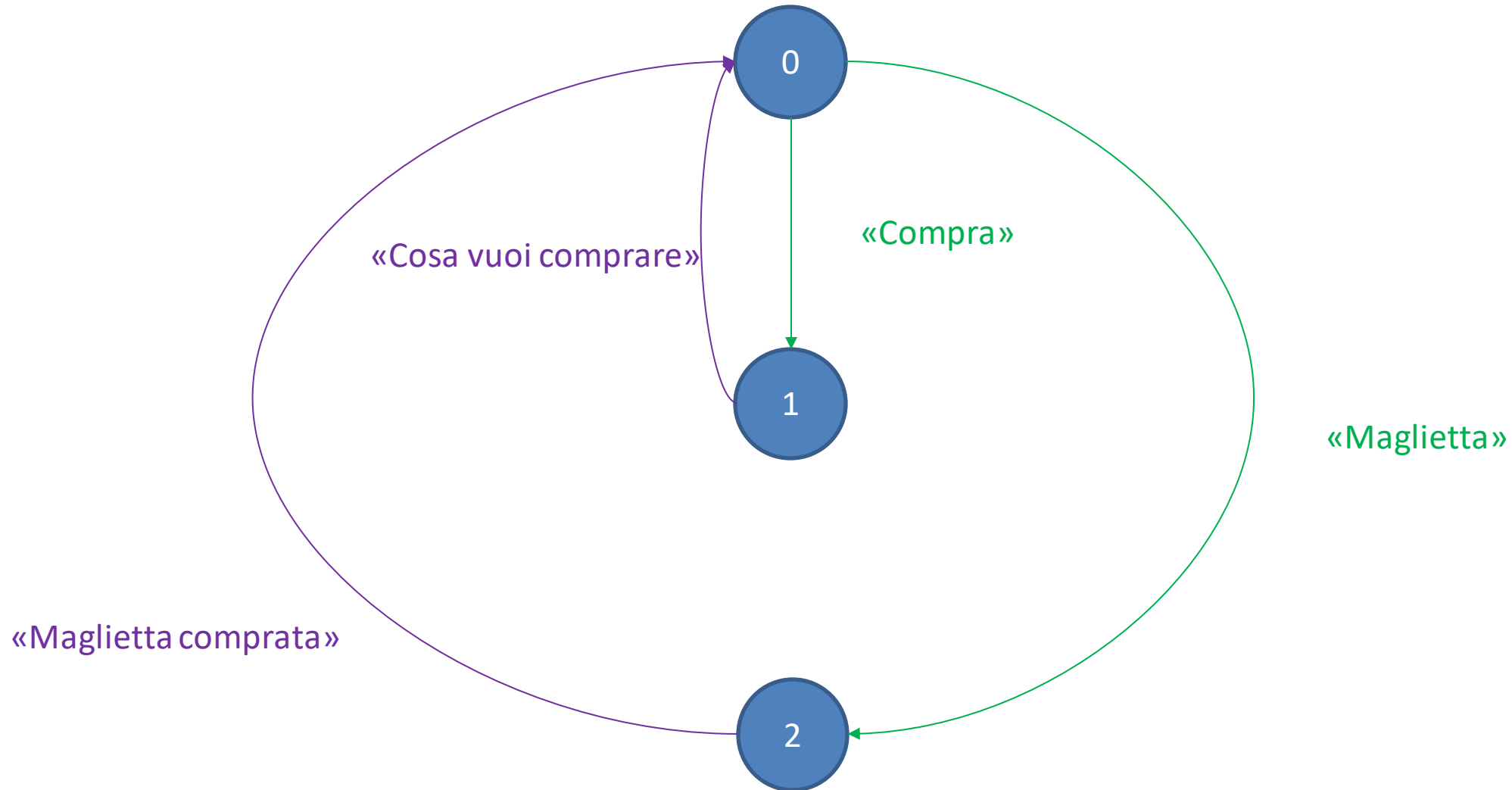   - Lists;
    - Dictionaries;

Eliza: conversational levels

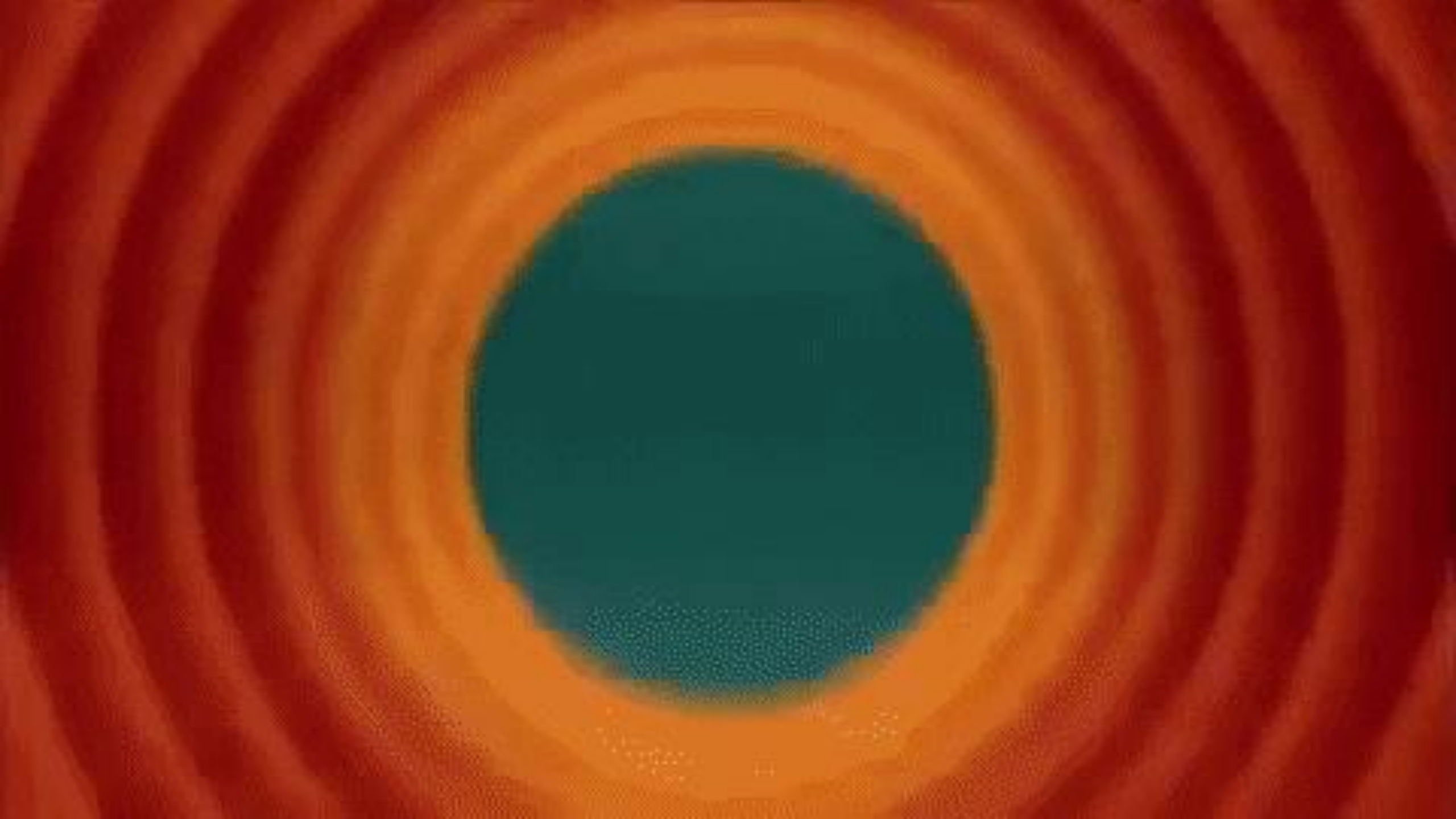- Eliza should implement conversation levels, based on user action;
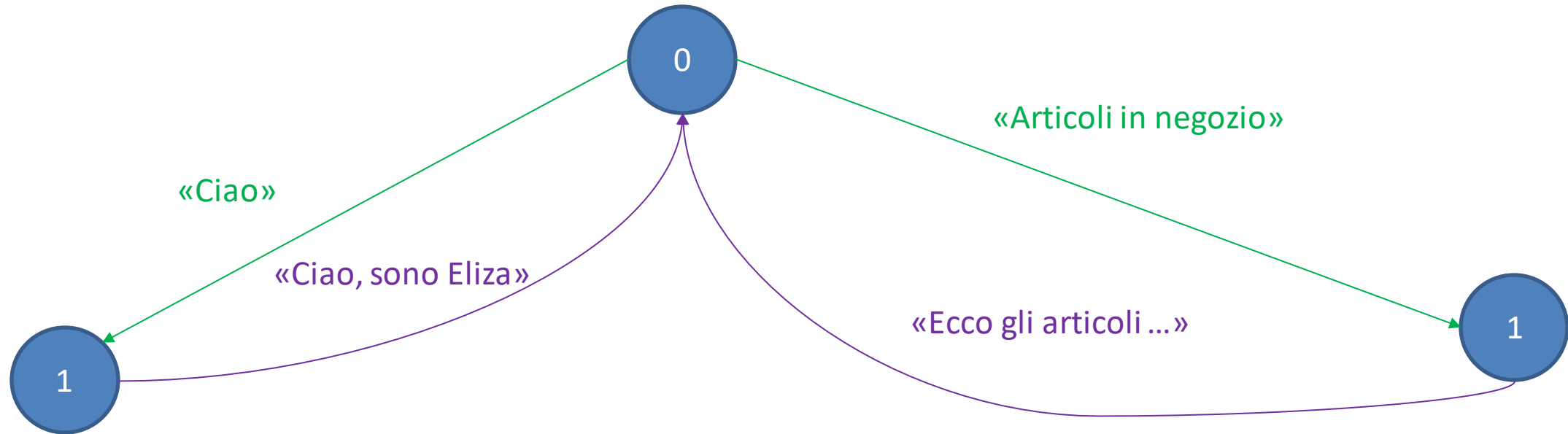
Eliza: conversational levels

- Eliza should implement conversation levels, based on user action;

Let's code!

Lorenzo Stacchio

Department for Life Quality Science

lorenzo.stacchio2@unibo.it