Python programming

Lorenzo Stacchio

Studente di dottorato in Computer Science

Dipartimento di Scienze per la Qualità della Vita

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Basi di Informatica

Lorenzo Stacchio

Studente di dottorato in Computer Science

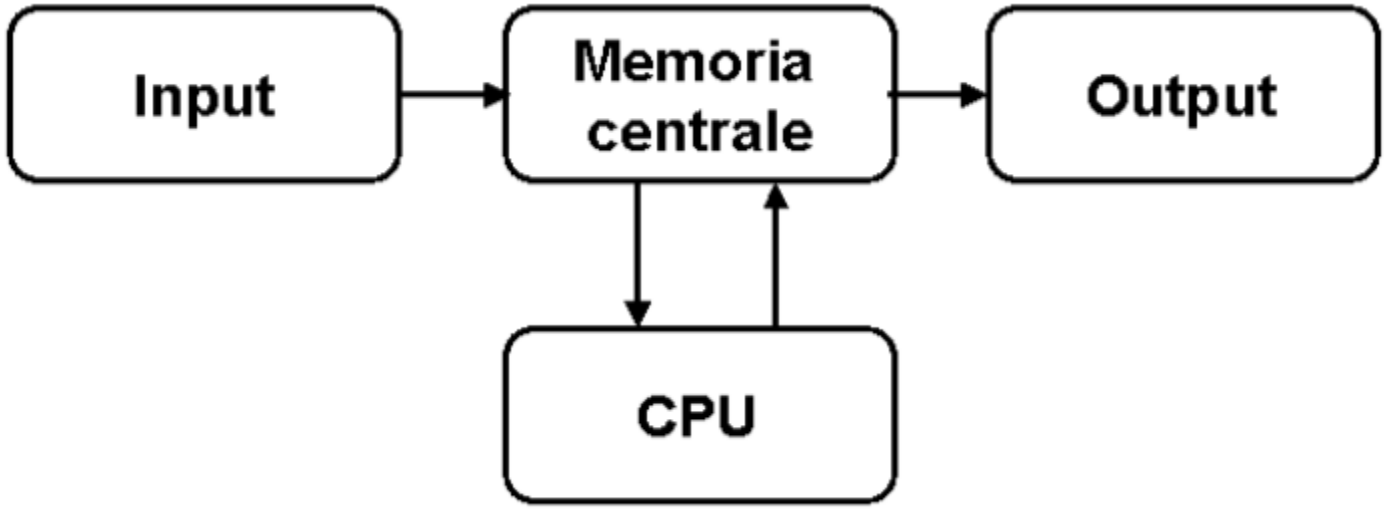
Dipartimento di Scienze per la Qualità della Vita

Tipi di Computer (quasi)

Mainframe: è un tipo di computer caratterizzato da prestazioni di elaborazione dati di alto livello di tipo centralizzato, opposto dunque a quello di un sistema distribuito come un cluster computer. Tipicamente sono presenti in grandi sistemi informatici come i centri elaborazione dati o organizzazioni (pubbliche e private) dove sono richiesti elevati livelli di multiutenza, enormi volumi di dati, grandi prestazioni elaborative, unite ad alta affidabilità. (Wikipedia)

Un **personal computer** si presta all'utilizzo proprio personale e alla personalizzazione da parte dell'utente nell'uso quotidiano. A livello hardware segue l'architettura di von Neumann ed è composto da un' unita' di elaborazione centrale CPU, da un'unità di memoria centrale (RAM) e infine da un'unità di archiviazione dati(Disco). (~ Wikipedia)

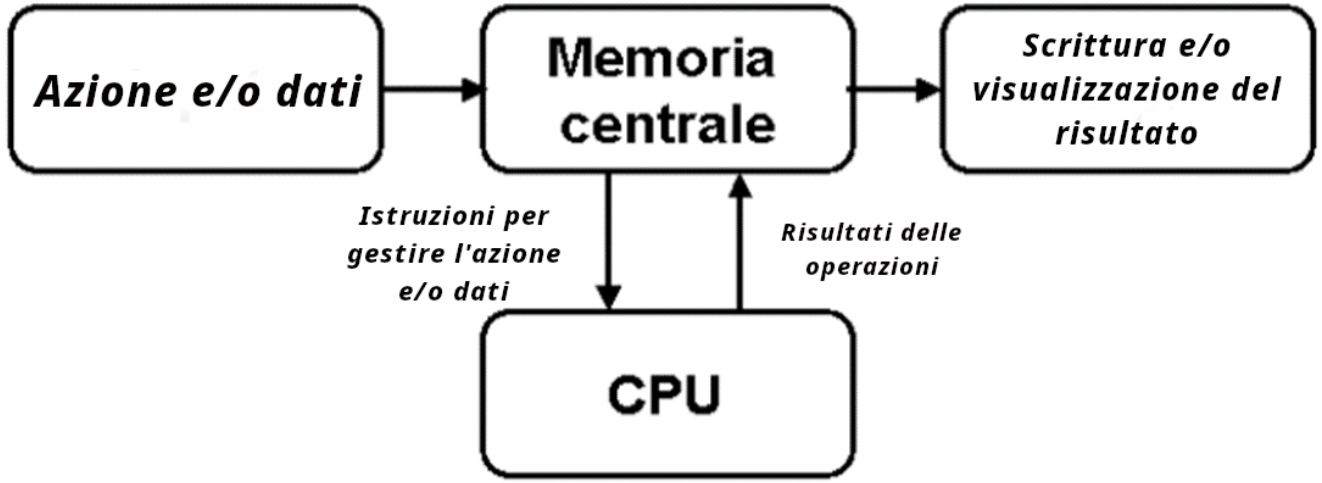
Architettura di Von Neumann



- In tutti i tipi di calcolatore elettronico, abbiamo sempre una (o più) unità di calcolo (CPU, Central Processing Unit), una memoria centrale che possiamo pensare come un enorme casellario che contiene i numeri binari da elaborare dove la CPU preleva e scrive dati, e dispositivi che permettano all'utente di inserire dati (Input) e leggere risultati (Output).

- Tipicamente l'input, viene associato alle periferiche per l'utilizzo del computer (disco rigido, mouse, tastiera etc.); l'output viene invece spesso definito come il modo per visualizzare i risultati delle azioni che stiamo eseguendo.

Il software: un modo per transformare i dati



- **Un software può essere definito come un modo per manipolare e trasformare dei dati**:
  - Un tocco del mouse cambia la finestra che stiamo utilizzando;
  - Scrivere su un foglio di testo cambia la rappresentazione globale del testo;

- **Analizzare dei dati** significa **trasformare** quei dati in modo tale da ottenere le risposte che cerchiamo:
  - Statistiche (media, varianza etc.);
  - Profilazione di un account basandosi sulle azioni che l'utente effettua;
  - Classificazione di un animale basandosi sui pixel di una immagine.

Calcolo automatico e codifica binaria

- Il calcolatore è un elaboratore automatico d'informazione, che esegue operazioni su oggetti (**dati**) per produrre altri oggetti (**dati trasformati o risultati**);

- Il calcolatore che abbiamo descritto è tuttavia "elettronico" in quanto gli oggetti che riceve in ingresso, elabora e fornisce in uscita sono segnali elettrici, livelli di tensione e la loro elaborazione avviene mediante circuiti elettronici a semiconduttore;

- Questa definizione può essere però rilassata ad **elaboratore digitale**: il significato attribuito ai livelli di tensione non è legato, infatti, al loro valore assoluto, ma al loro essere superiori od inferiori a una determinata soglia, **nel qual caso si attribuisce il valore 1 o 0**. **I dati in ingresso ed in uscita nei calcolatori elettronici sono quindi sequenze di valori 0 o 1** (bit, da BInary digiT);

- L'esecuzione di azioni viene richiesta all'elaboratore attraverso opportune direttive, dette istruzioni, codificate in **codice binario**;

- Tutti i calcolatori, quindi, effettuano operazioni su sequenze binarie dette **stringhe**.

- **Numeri, lettere, immagini e tutte le informazioni che vogliamo elaborare** automaticamente vengono anch'esse comunemente rappresentate mediante sequenze di simboli appartenenti ad un alfabeto di dimensioni però ben più grandi di 2. **Il caso dei simboli binari rappresenta semplicemente il caso minimale**, cioè il numero minimo di simboli sufficiente per rappresentare informazione.

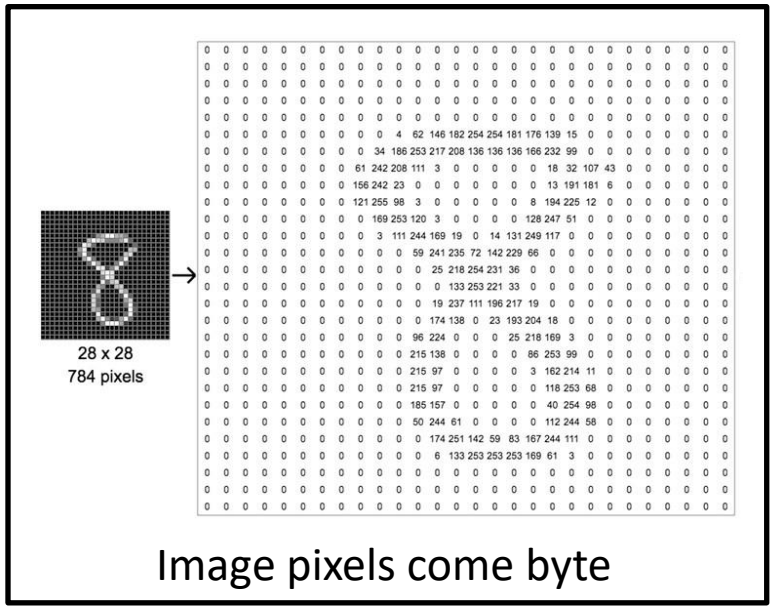


Image pixels come byte

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

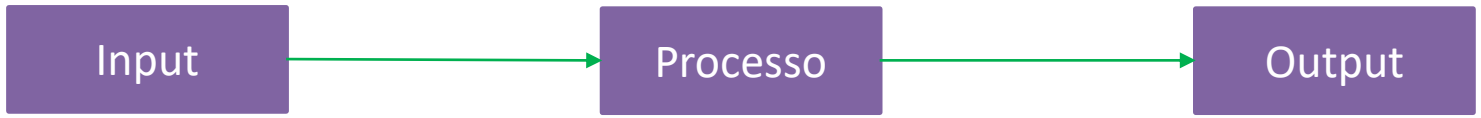
- Le memorie dei calcolatori, quindi, immagazzinano sequenze binarie. Per dare gli ordini di grandezza della quantità di informazione si utilizzano i vari multipli del bit:

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 e algoritmi

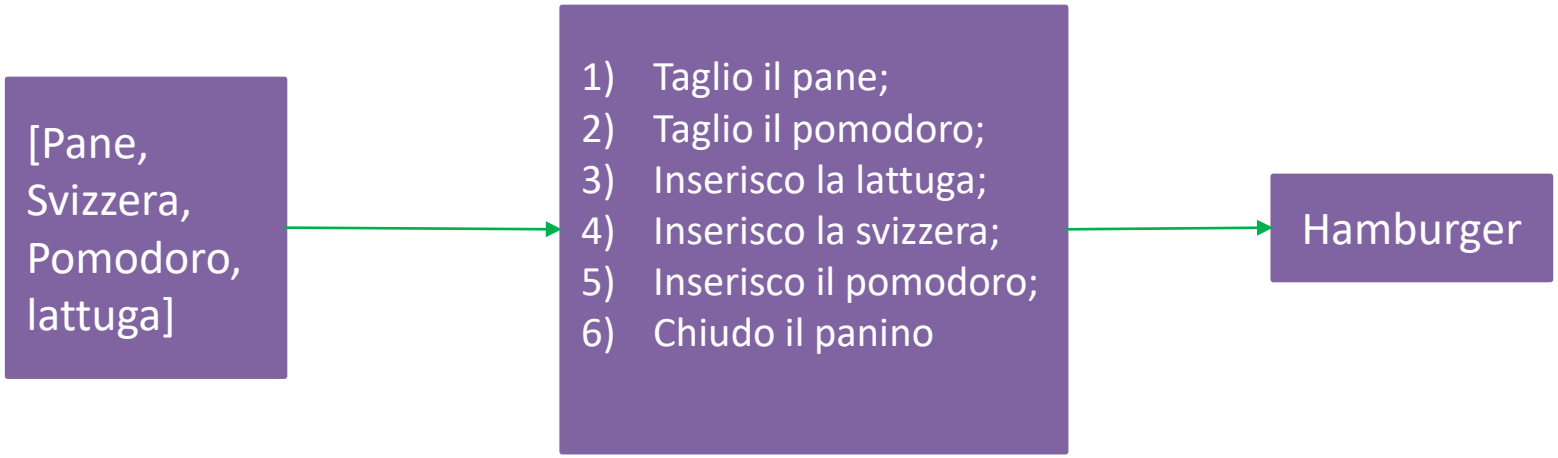
- Finora, abbiamo trattato il software semplicemente come una o più procedure per trasformare dati;
- Tuttavia sembra che queste procedure avvengano in maniera caotica, senza un ordine e uno scopo ben preciso. In effetti questa è l'interpretazione più sbagliata che possiamo dare ad un software;
- Un software moderno, è tipicamente formato un insieme di algoritmi;
- Un algoritmo è definito come la sequenza di istruzioni che descrive come, **a partire da un certo input**, restituisca un **determinato output**;



Algoritmi deterministici

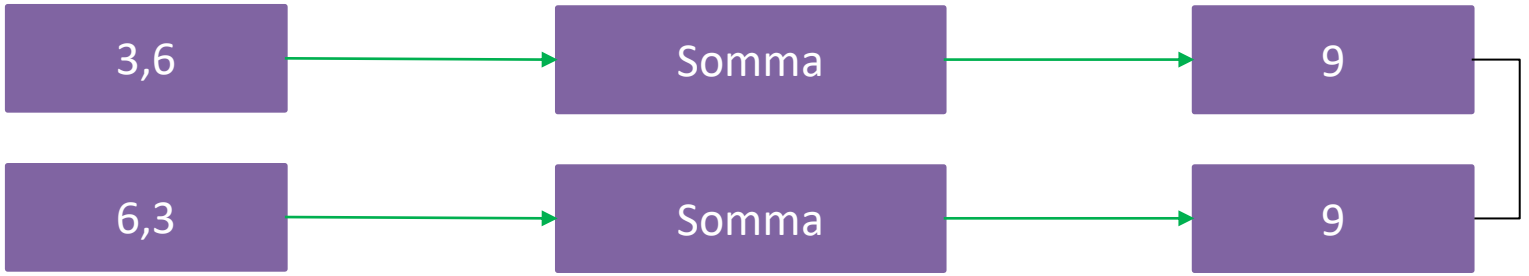
- Possiamo paragonare un algoritmo a una ricetta. Ad esempio, preparando un panino, si eseguono una serie di passaggi: si prendono gli ingredienti, si inseriscono sopra una fetta di pane, si appoggia sopra un'altra fetta di pane. Se dovessimo formalizzare la preparazione di un sandwich algoritmicamente:

- **Un algoritmo è detto deterministico se e solo per la stessa tipologia di input si produce lo stesso output**. Se **decido** di **utilizzare** pane, svizzera di carne, pomodoro e lattuga e li **assemblo** potrò **produrre esclusivamente** un hamburger (di certo non faccio una pizza, per lo meno nella maggior parte dei paesi del mondo ☺).



Gli algoritmi deterministici e le pari opportunità

- Non è tuttavia detto che, a due input diversi non possa corrispondere lo stesso output;
- Prendiamo come esempio un algoritmo che definisca la somma di due numeri interi:



Commutatività

- Definiamo **un algoritmo «Discriminatore» che restituisca 0 se riceve in input un numero inferiore o pari a 10** e **1 se riceve un numero maggiore di 10**;

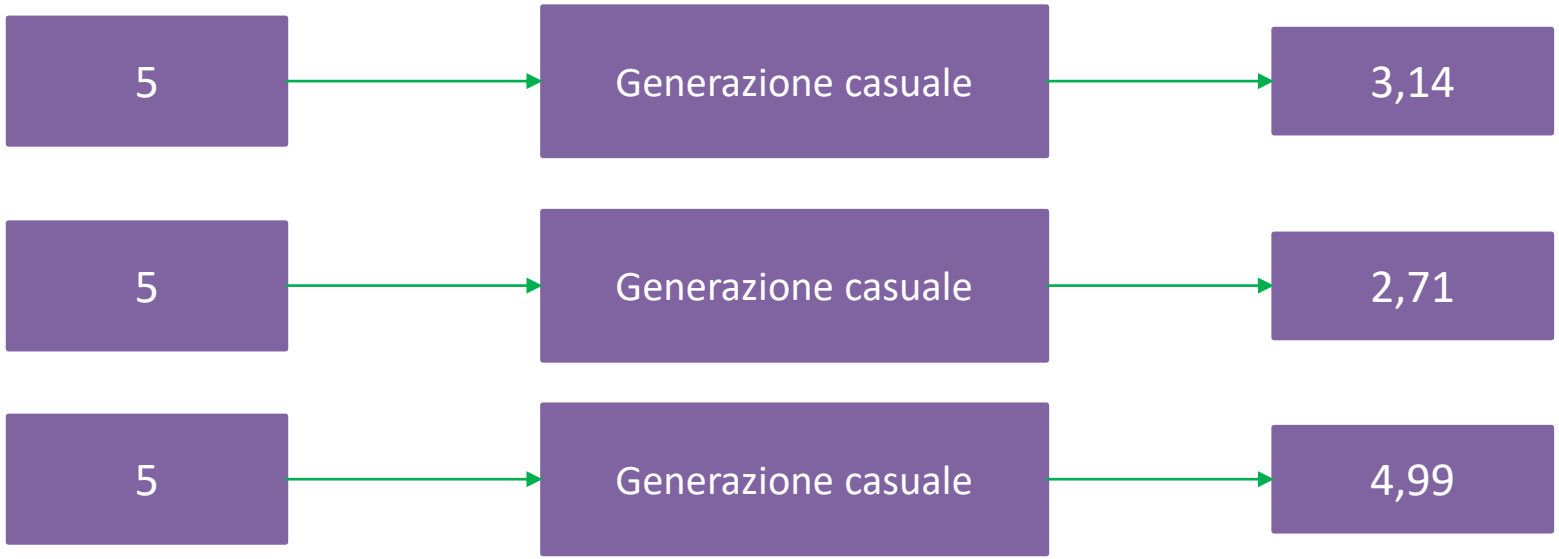
- **Questo fenomeno** è ottenibile utilizzando gli **operatori di controllo del flusso;**

Algoritmi non deterministici

- **Un algoritmo è detto non deterministico se e solo per la stessa tipologia di input si possono produrre diversi output**. Un algoritmo, può esibire comportamenti non deterministici in diversi modi:

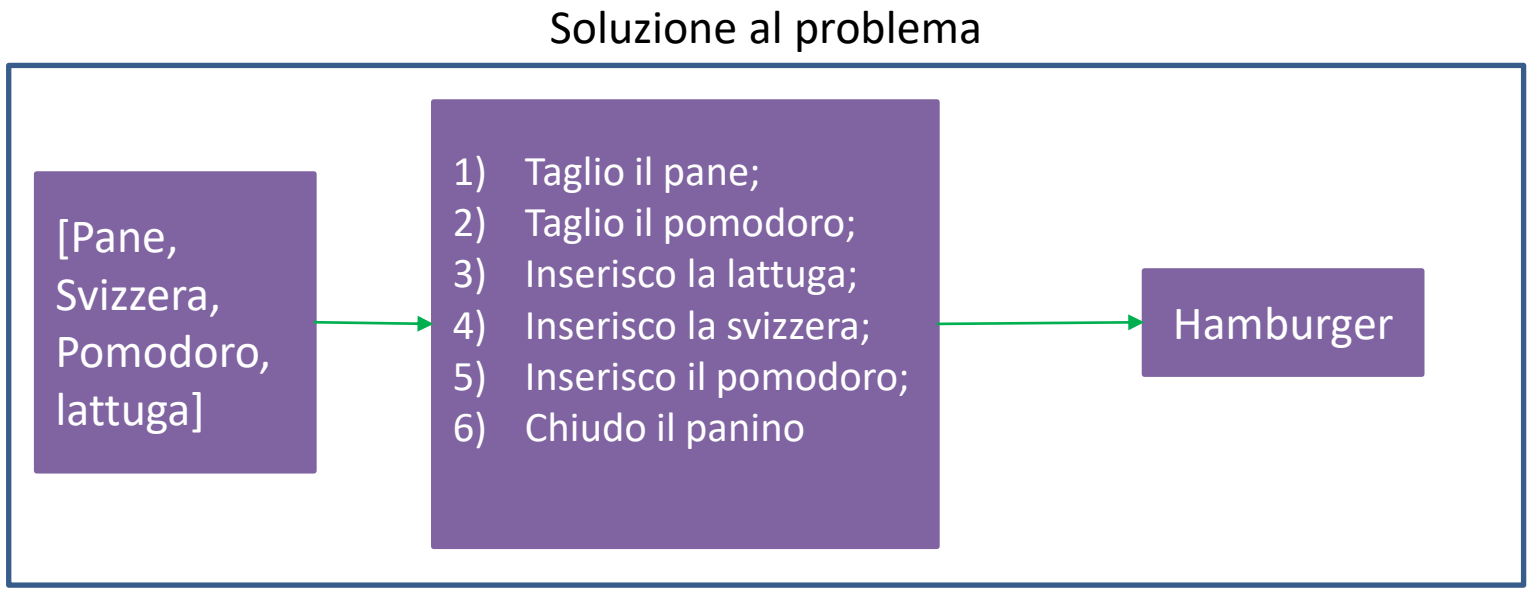
  - Facendo uso di **funzioni probabilistiche**;
  - Se ha una natura di esecuzione **concorrente** (ne parleremo nella lezione dedicata ai paradigmi Big Data);

- Ad esempio, definiamo un algoritmo non deterministico, che per un qualunque numero intero **n**, restituisca un numero decimale casuale tra **[0,n].**



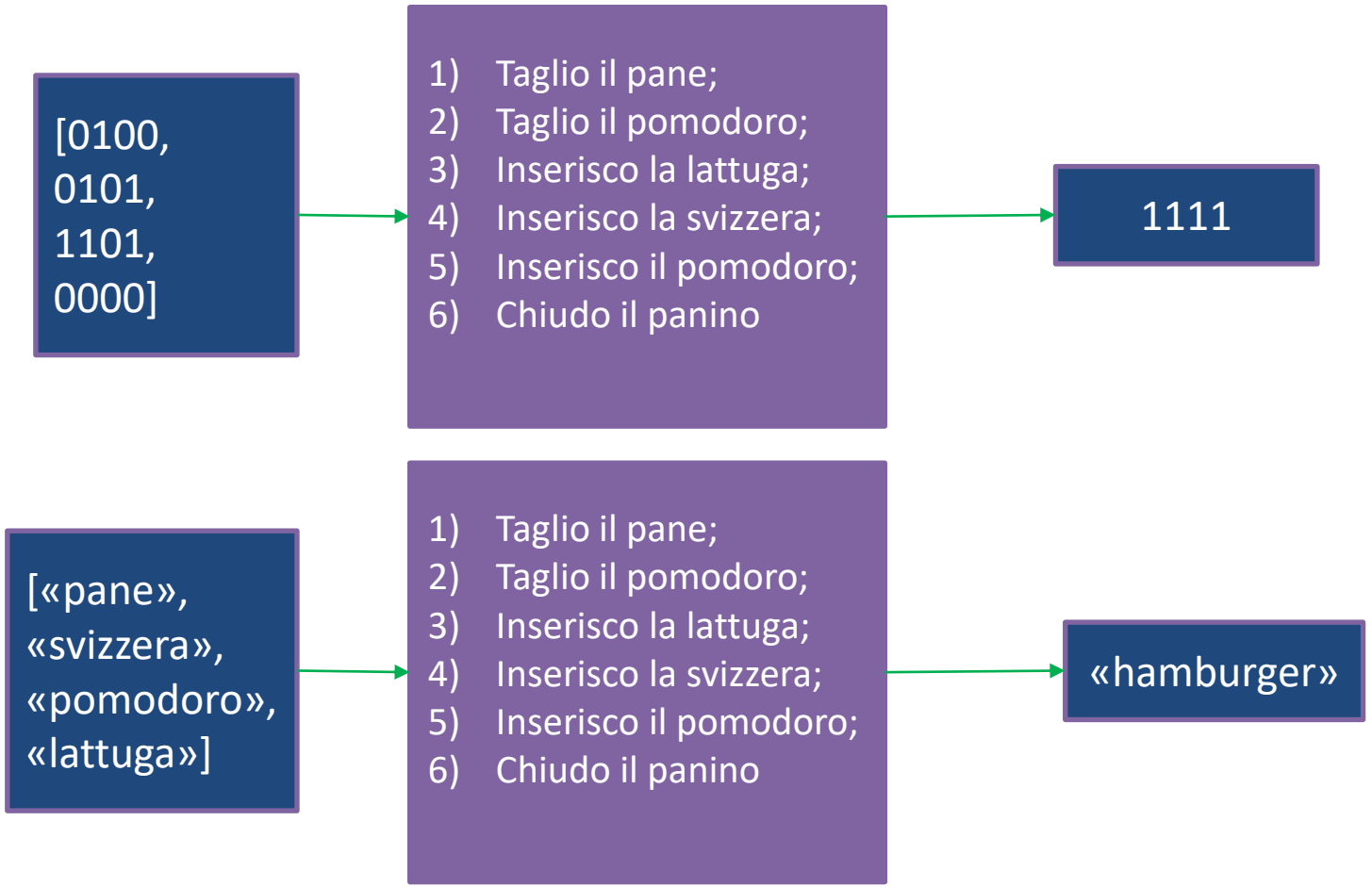
Algoritmica per la risoluzione dei problemi

- Supponiamo di avere un determinato problema e di volerlo risolvere algoritmicamente. Per far questo dovranno essere verificate alcune condizioni:
  - La soluzione del problema deve essere nota;



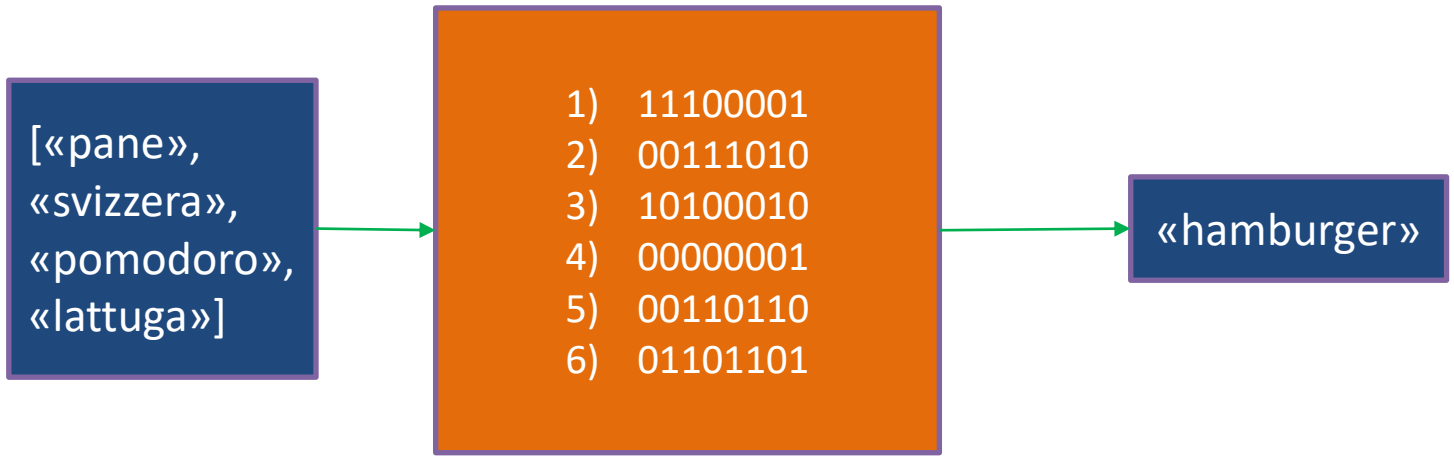
Algoritmica per la risoluzione dei problemi

- Supponiamo di avere un determinato problema e di volerlo risolvere algoritmicamente. Per far questo dovranno essere verificate alcune condizioni:
  - La soluzione del problema deve essere nota;
  - I dati in ingresso devono essere codificati in maniera adeguata (in generale numeri binari o loro rappresentazioni);



Algoritmica per la risoluzione dei problemi

- Supponiamo di avere un determinato problema e di volerlo risolvere algoritmicamente. Per far questo dovranno essere verificate alcune condizioni:
  - La soluzione del problema deve essere nota;
  - I dati in ingresso devono essere codificati in maniera adeguata (in generale numeri binari o loro rappresentazioni);
  - I passi che la compongono debbono poter essere tradotti in **un linguaggio comprensibile dal calcolatore**.



Scrivere algoritmi in linguaggi comprensibili dal calcolatore

- Un calcolatore, di per sé, comprende soltanto una serie di istruzioni scritte per il microprocessore e codificate in quello che viene detto “linguaggio macchina”, cioè una serie di codici di operazioni (una sequenza di byte) e di operandi (dati scritti anche essi in binario) ed è in grado di lavorare su dati numerici di tipo e dimensione limitati;

- Nonostante il linguaggio macchina non sia binario stresso, programmare un applicazione moderna lavorando a questo livello avrebbe comunque sempre una complessità proibitiva;

- Fortunatamente (no)i programmatori, non hanno bisogno di conoscere né il binario né il linguaggio macchina;

- Esistono infatti linguaggi detti **«di alto livello»** che contengono **costrutti** che **ricordano il linguaggio naturale** e attraverso i quali si possono scrivere algoritmi che verranno poi **trasformati interamente** in **linguaggio macchina** dai cosiddetti «**compilatori»**, oppure controllati e interpretati linea dopo linea da programmi che vengono detti **«interpreti»**.

- Esistono quindi linguaggi **compilati** (es. **C, C++, C#**), linguaggi **sia compilati che interpretati** (es. **Python, Java**);

Algoritmi: Analisi, soluzione e codifica del problema

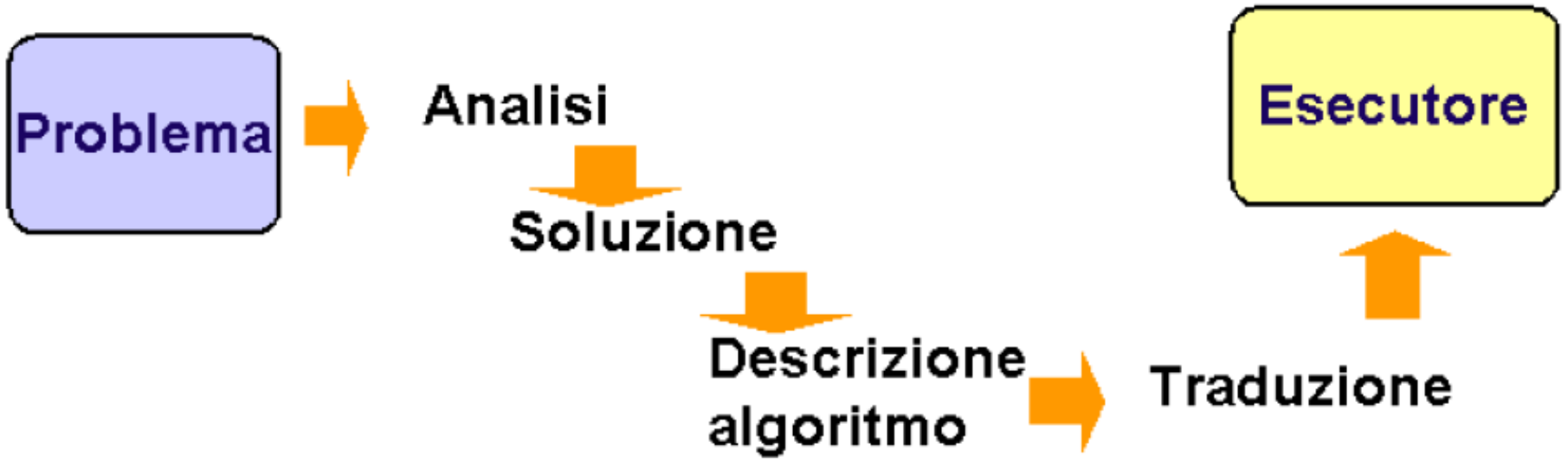


Figura 17: Dal problema allo sviluppo di programmi

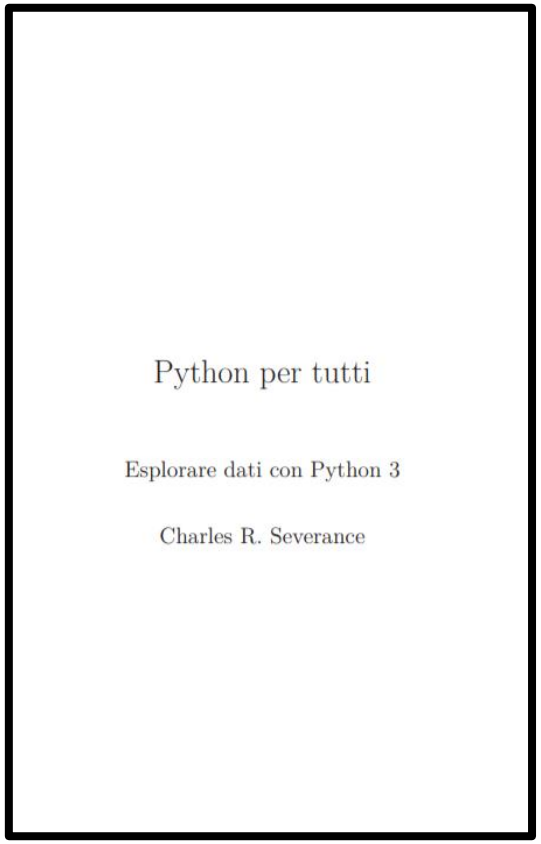
Programmazione in Python

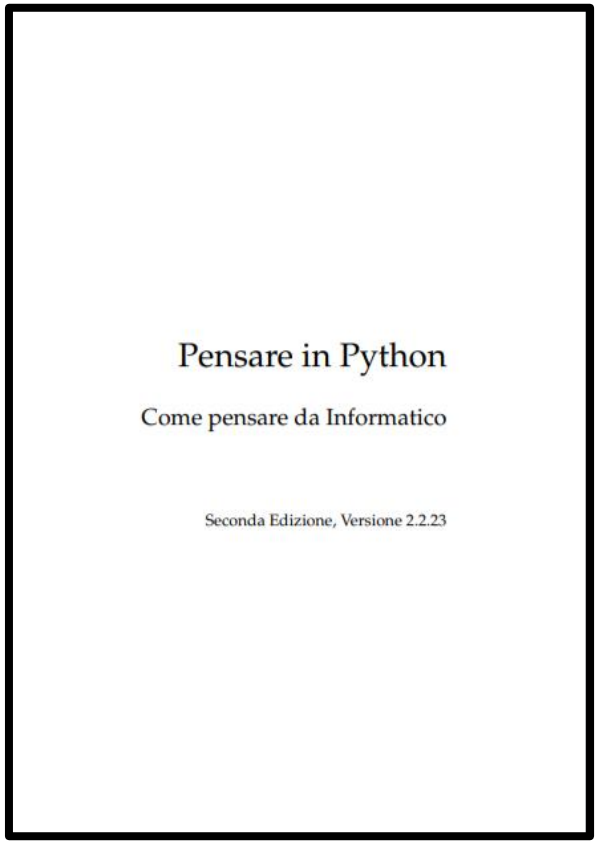
Lorenzo Stacchio

Studente di dottorato in Computer Science

Dipartimento di Scienze per la Qualità della Vita

Materiale di approfondimento su python





Udemy

YouTube

GitHub

Perché Python?

- Guido van Rossum, inventore di python, era appassionato della serie comica targata BBC "Monty Python's Flying Circus" (anni '70). Van Rossum ha pensato quindi ad un nome corto, unico e misterioso per il suo nuovo linguaggio: **Python;**

Citando Allen Downey:

1. « Inizialmente insegnammo C++ agli studenti del primo anno…due anni più tardi io ero convinto che il C++ fosse una scelta non adeguata per introdurre gli studenti all'informatica: mentre da un lato il C++ é certamente un linguaggio molto potente, esso si dimostra tuttavia essere estremamente difficile da insegnare ed imparare.»

2. **«Mi trovavo costantemente alle prese con la difficile sintassi del C++ e stavo inoltre inutilmente perdendo molti dei miei studenti**. […]»

3. «Avevo bisogno di un **linguaggio facile da insegnare e imparare** che potesse **girare** tanto sulle macchine **Linux** del nostro laboratorio quanto sui sistemi **Windows** e **Macintosh**.»

4. «Doveva supportare tanto la **programmazione procedurale quanto altri paradigmi**»

5. «Lo volevo **open-source** (gratuito) e che avesse **un'attiva comunità di sviluppatori**. Python sembrò il migliore candidato.»

Python è un linguaggio (più) semplice

- Come intuizione dietro la facilità sintattica e di sviluppo di python, vediamo come codificare la prima istruzione codificata da ogni sviluppatore: la stampa della frase «Hello World».

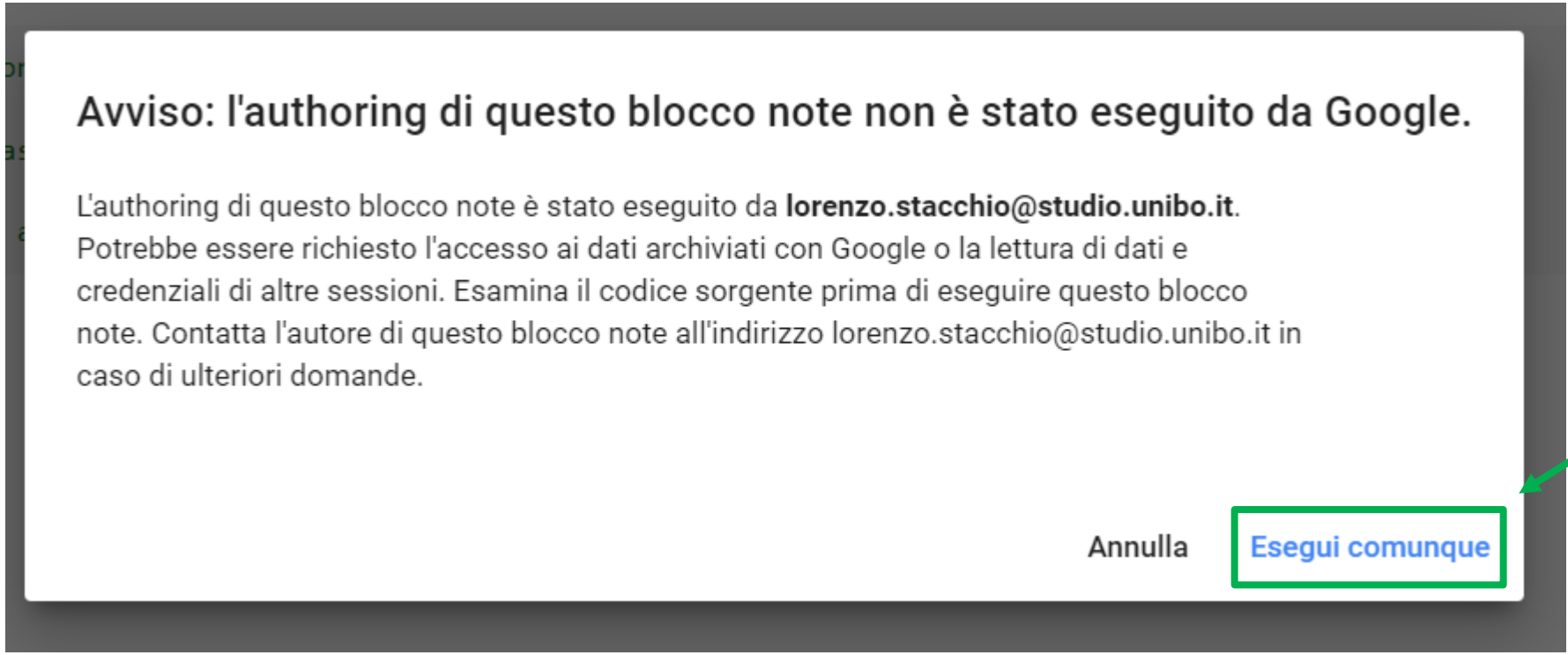
C++	Java	Python

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

Say 'Hello World' in 28 Different Programming Languages

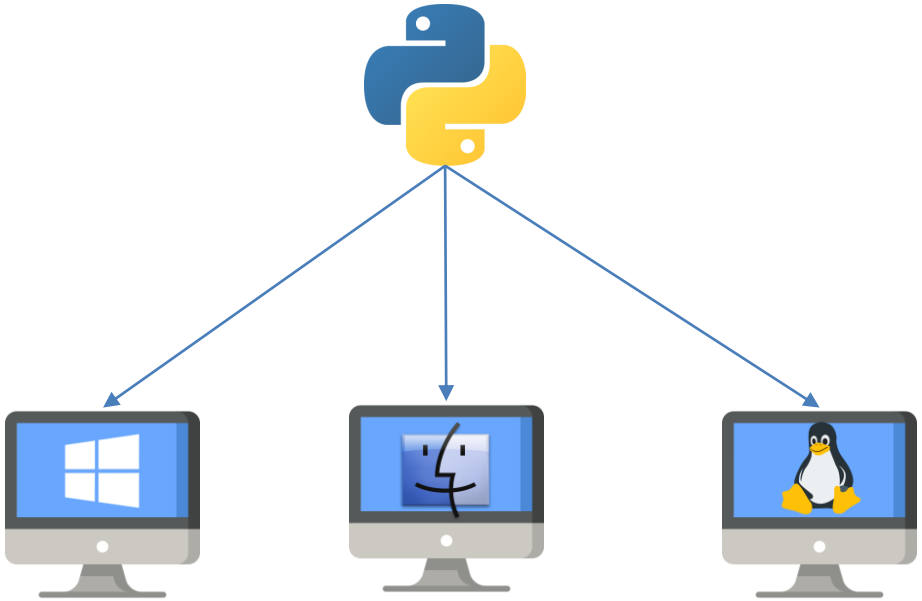
La nostra prima istruzione!

- Apriamo il nostro primo Google Colab;
- Una volta aperto ed avendo loggato con il vostro account google, questo foglio diventerà la vostra copia personale!
- Per cui potrete modificare a vostro piacimento!



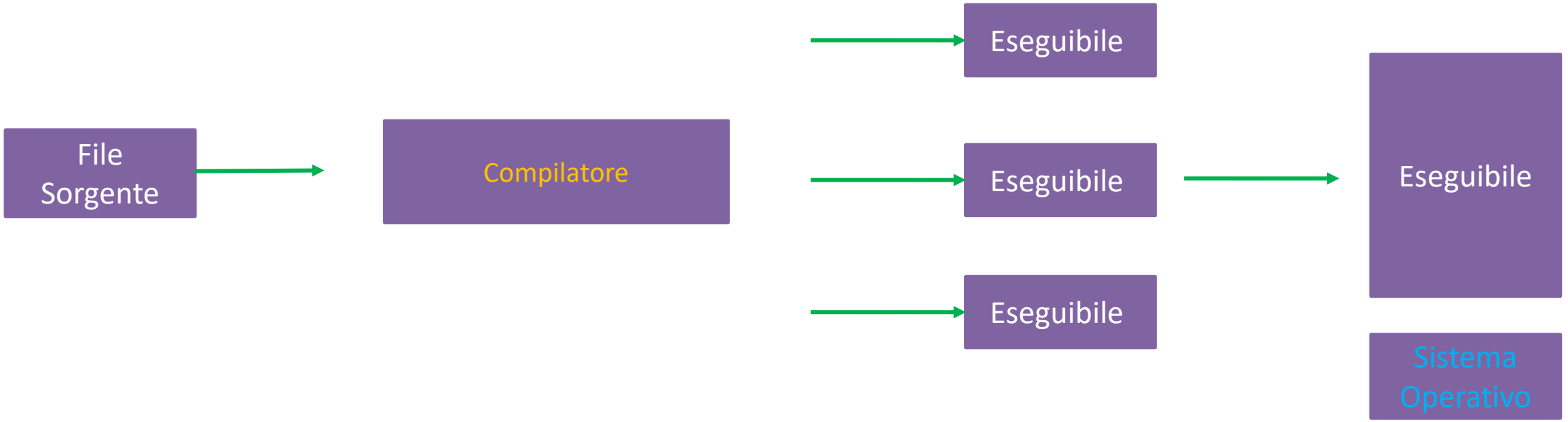
Python è un linguaggio portabile e interpretato

- Il linguaggio Python è un **linguaggio portabile**.
- Lo stesso codice Python può essere eseguito su Windows e altre piattaforme come Linux e Mac, senza che sia richiesta alcuna modifica (a parità di versione e librerie installate);
- Questo è possibile poichè il linguaggio Python, similmente a Java ma non in ugual misura, è un linguaggio sia **compilato che interpretato e fa uso di una Virtual Machine.**



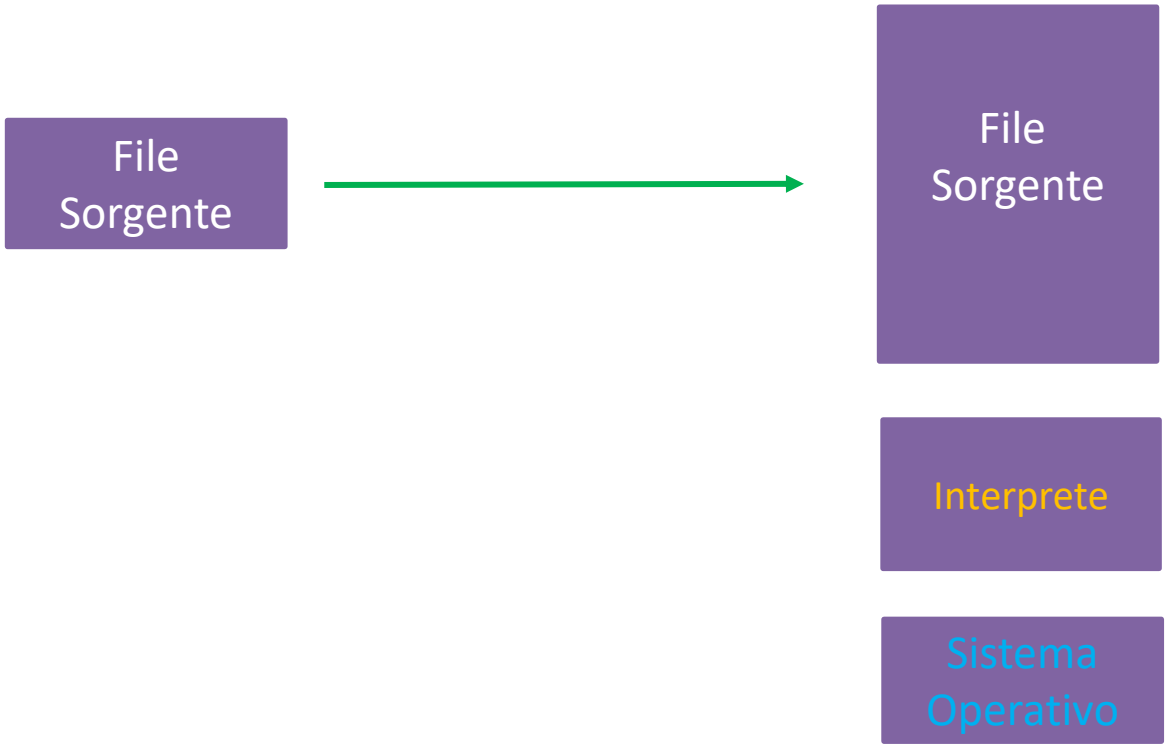
[Fonte di approfondimento Python Interpreter]

Linguaggi compilati



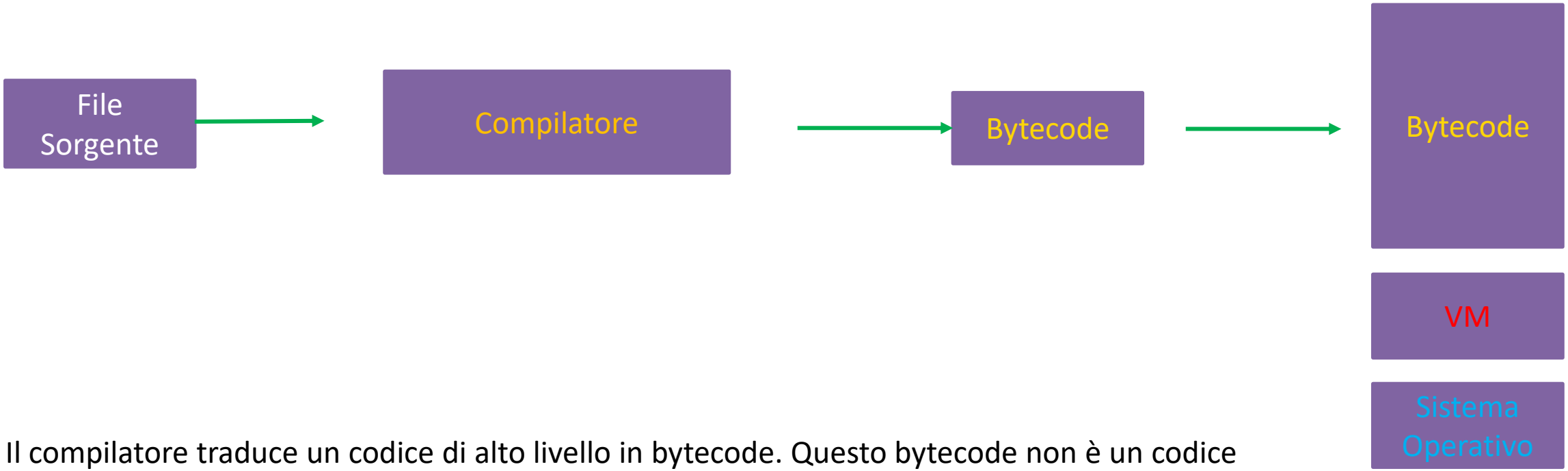
- Il compilatore traduce un codice di alto livello del codice eseguibile per l'architettura fisica del computer e per il sistema operativo in cui ci troviamo in un certo momento;
- Il codice generato è direttamente eseguibile senza ulteriori passaggi (efficienza);
- Il processo di compilazione avviene una sola volta e riguarda l'intero file sorgente;
- Un linguaggio compilato, dovrà quindi essere ritradotto ogni volta che cambiamo architettura fisica e/o sistema operativo.

Linguaggi interpretati



- L'interprete traduce un codice di alto livello riga per riga per una certa architettura e un certo sistema operativo e la esegue immediatamente;
- Questo porta un vantaggio: lo stesso codice sorgente può essere eseguito su piattaforme differenti, perchè sarà l'interprete a tradurre il file sorgente in base all'architettura!
- Allo stesso tempo però il linguaggio risulta meno efficiente;

Linguaggi compilati e interpretati



- Il compilatore traduce un codice di alto livello in bytecode. Questo bytecode non è un codice eseguibile ma è una codifica astratta e ideal del linguaggio macchina (più facilmente interpretabile).
- Sarà poi la Virtual Machine a tradurre questo bytecode in codice macchina per l'architettura e il sistema operativo di riferimento;
- Abbiamo unito I vantaggi di entrambi I mondi: abbiamo evitato una lunga compilazione per ogni architettura e ridotto l'inefficienza di un linguaggio completamente interpretato!

Python ha un'attiva comunità di sviluppatori

Big Data & Data Science & Machine Learning

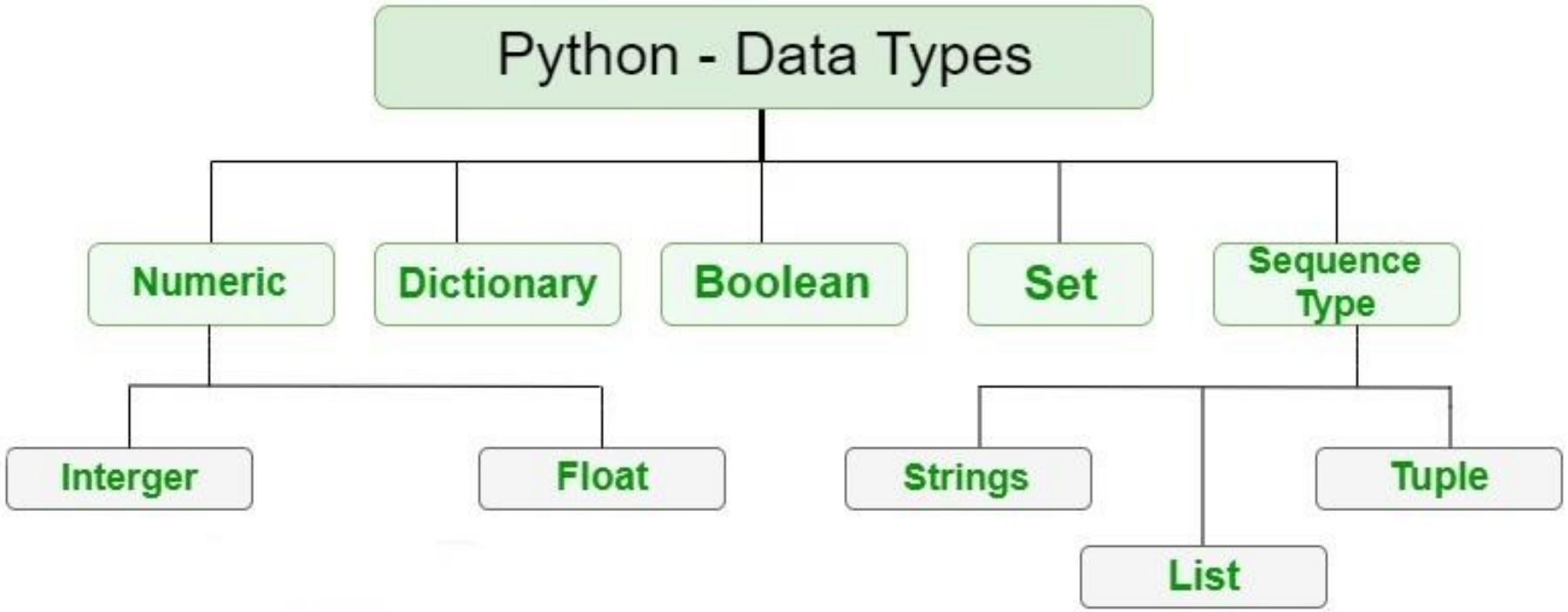
Deep Learning

Let's code!

Variabili e operatori

Valori e tipi

- Un valore è uno degli artefatti fondamentali per un programmatore;
- Un valore è definibile come la rappresentazione di qualche entità che può essere manipolata da un programma;
- I membri di un certo **tipo** sono i valori di quel **tipo;**
- Finora abbiamo visto il valore «Hello, World» che appartiene al **tipo stringa** (che rappresenta una serie di caratteri);
- Il tipo stringa viene identificato poiché inserito tra le due " " ed è fondamentale, poiché consente di comunicare in maniera agevole con l'utente che sta utilizzando il software;
- Chiaramente il **tipo** stringa è solo un primo esempio dei tipi che possiamo trovare all'interno dell'ecosistema python;



Casting dei tipi: implicito ed esplicito

- Dal notebook si evince che la somma tra un intero e un float restituisca un tipo float;
- Questo è dovuto al meccanismo chiamato **casting implicito dei tipi**; Questo meccanismo automaticamente infierisce e cambia il tipo dei valori in base alle operazioni che noi svolgiamo;
- Ad esempio, la somma tra 3.5 e 2 deve restituire 5.5 a meno che non desideriamo la parte intera del risultato;
- Per ottenere la parte intera del risultato dovremo effettuare un'operazione di **casting esplicito**, cioè comunicare esplicitamente al nostro interprete che noi vogliamo che la somma ci restituisca un intero;
- Questo casting può essere effettuato per i tipi primitivi visti nella tabella precedente ma dobbiamo fare molta attenzione perché non potremo convertire esplicitamente tutti i tipi in tutti gli altri tipi;

Variabili

- Una delle caratteristiche più importanti in un linguaggio di programmazione è la capacità di manipolare variabili.
- **Una variabile è un nome che si riferisce ad un valore di un certo tipo.**
- Python, a differenza di altri linguaggi è **dinamicamente tipato**, per cui possiamo assegnare un valore di un certo tipo ad una variabile e poi cambiarlo quando vogliamo;
- Le variabili vengono nominate con la notazione **snake_case;**
- **Possiamo usare le variabili per svolgere le stesse operazioni che svolgeremmo con dei valori normali;**
- **Non potrete usare le parole chiavi riservate di Python per nominare le vostre variabili;**


```
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
```

Valutazione delle espressioni e operatori

- Un'espressione è una combinazione di valori, variabili e operatori.

- Gli **operatori** sono simboli speciali che rappresentano elaborazioni di tipo matematico, quali la somma e la moltiplicazione. I valori che l'operatore usa nei calcoli sono chiamati **operandi**;

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

- L'uso dei simboli +, -, / e delle parentesi sono uguali a all'uso che se ne fa in matematica. **L'asterisco** (*) è il simbolo della moltiplicazione ed il **doppio asterisco** (**) quello dell'elevamento a potenza.

- Quando una variabile compare al posto di un operando essa è rimpiazzata dal valore che rappresenta prima che l'operazione sia eseguita.

Operatori booleani

- In tutti i linguaggi di programmazione sono presenti operatori booleani, ossia operatori che consentano di formalizzare i costrutti logici di base.
- Noi vedremo solo alcuni costrutti, che saranno utili soprattutto quando parleremo di **operatori di controllo del flusso**;
- Hanno tutti la stessa priorità e le operazioni si eseguono da sinistra verso destra;

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

- Possiamo confrontare anche i valori fra due numeri!

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

Quando un dato è False

- **None** → dato per rappresentare il nulla;
- **False**
- Uno **zero** di **qualunque tipo** → 0, 0.0
- Una stringa vuota → ‘’
- Una qualunque sequenza **vuota** → (), []
- Un qualunque dizionario **vuoto** → {}

Gli elementi sequenza e dizionario **fanno parte delle strutture dati, che vedremo fra un po’**

Se non appartiene a una di queste categorie, un dato viene considerato **TRUE** per la logica booleana implementata in Python

Espressioni su stringhe

- In generale non puoi effettuare operazioni matematiche sulle stringhe, anche se il loro contenuto sembra essere un numero. Se supponiamo che messaggio sia di tipo string gli esempi proposti di seguito sono illegali:

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

- Tuttavia, l'operatore **+** funziona come **concatenatore di stringhe**;

«banana» + «frutta» = «bananafrutta»

- Anche l'operatore ***** funziona come **ripetitore di stringhe**;

«banana» * 3 = «bananabananabanana»

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

Time for exercises

Funzioni e scope di variabili

Funzioni type() e print()

- Fino ad ora abbiamo usato delle funzioni senza sapere cosa fossero;
- type() e print() sono infatti due tipi di funzioni!

Analizziamo la funzione **type(32)**

- Il nome della funzione è **type** e **ritorna** il **tipo del numero 32** (cioè un intero).
- Il valore della variabile, che è chiamato **argomento della funzione**, deve essere racchiuso tra parentesi (in questo caso è **32**).
- E' comune dire che una funzione **"accetta" o "prende"** un **argomento** e **"restituisce" o "ritorna"** un risultato.
- Il risultato è anche detto **valore di ritorno della funzione**. In questo caso, corrisponde al **tipo della variabile 32,** cioè un intero;

Analizziamo ora la funzione **print(32)**

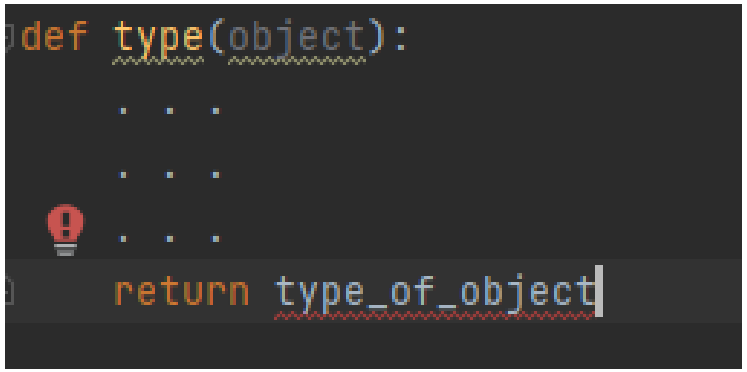
- Il nome della funzione è **print** e consente di stampare a video un valore.
- Il valore, che è chiamato **argomento della funzione**, deve essere racchiuso tra parentesi (in questo caso è **32**).
- La funzione print non ritorna alcun valore!
- Infatti se io facessi la print di una print (inception) avrei come risultato **None**, cioè il valore che rappresenta il nulla;

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

```
5
None
```

Funzioni

- In generale quindi una funzione:
  - **Possiede un nome identificativo;**
  - **Definisce nessuno, uno o più argomenti;**
  - **Restituisce nessuno, uno o più valori;**
  - **Definisce una serie di istruzioni al suo interno;**



Let's code!

Funzioni matematiche

- Esistono tantissime **librerie** Python per svolgere operazioni matematiche;
- Tuttavia, python stesso espone un **modulo chiamato Math** che permette di svolgere praticamente tutte le operazioni algebriche note (e anche altro);
- Per **chiamare una funzione di un modulo** dobbiamo specificare il nome del modulo che la contiene e il nome della funzione separati da un punto. Questo formato è chiamato **notazione punto**.

```
>>> 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)

Librerie e moduli

- Le librerie sono raccolte di codice che i programmatori possono utilizzare per ottimizzare le loro attività (e non re-inventare la ruota);
- Noi abbiamo re-implementato la somma, la sottrazione etc… ma erano già presenti nel linguaggio python!
- Dovete essere consci del fatto che «se pensate una cosa banale, probabilmente qualcuno l'avrà sicuramente implementata»;
- Un modulo è un file di codice atto a eseguire un certo comportamento (**math** è un modulo del linguaggio python);
- Spesso libreria e modulo sono **sinonimi** ma vengono utilizzati in contesti diversi;
- La cosa certa è che sia le librerie che i moduli sono formati da **sotto-moduli**;
- Ad esempio, la libreria o modulo **pandas**, definisce molti sotto-moduli (che sono sempre file di codice) che implementano alcune operazioni in maniera **isolata**;

Scope di una variabile: locale vs globale

- Lo **scope o ambito di visibilità** di una variabile è la parte di un programma all'interno del quale si può fare riferimento ad essa.
- Le variabili dichiarate all'interno di una funzione sono dette **locali alla funzione** dal momento che sono accessibili soltanto all'interno del suo corpo;
- Le variabili definite in qualunque altro punto del codice vengono dette **globali**, e possono essere accedute in qualunque punto;
- Le variabili hanno quindi un livello di visibilità: **le variabili con livello x sono visibili a livello x e anche a livello x + 1, x + 2 e così via.**
- Vediamo un esempio per capire questo concetto che è di fondamentale importanza in programmazione;

```
# Siamo a livello 1 della gerarchia di visibilità
# variabile con scope globale

variabile_globale = "globale"

def funzione_per_esaminare_scope(variabile_locale_1, variabile_locale_2):
  # Siamo a livello 2 della gerarchia di visibilità
  variabile_locale_1 = "locale_1" # scope locale
  variabile_locale_2 = "locale_2" # scope locale
  print("Variabili con scope locali %s" % [variabile_locale_1,variabile_loc
ale_2])
  print("Variabili con scope locali %s" % [variabile_globale])
```

Spazio delle variabili

Nome	Valore	Livello di visibilità
`variabile_globale`	"globale"	1

```
# Eseguiamo questa riga di codice
variabile_globale = "globale"
```

Spazio delle variabili

Nome	Valore	Livello di visibilità
variabile_globale	"globale"	1

```
# Dichiariamo ora la funzione
def funzione_per_esaminare_scope(variabile_locale_1, variabile_locale_2):
  variabile_locale_1 = "locale_1" # scope locale
  variabile_locale_2 = "locale_2" # scope locale
  print("Variabili con scope locali %s" % [variabile_locale_1,variabile_locale_2])
  print("Variabili con scope locali %s" % [variabile_globale])
```

Spazio delle variabili

Nome	Valore	Livello di visibilità
variabile_globale	"globale"	1
variabile_locale_1	"globale"	2
variabile_locale_2	"globale"	2

```
# Chiamiamo la funzione ricordando la definizione
# funzione_per_esaminare_scope(variabile_locale_1, variabile_locale_2)

funzione_per_esaminare_scope(variabile_globale,variabile_globale)
```

Spazio delle variabili

Nome	Valore	Livello di visibilità
`variabile_globale`	"globale"	1
`variabile_locale_1`	"locale_1"	2
`variabile_locale_2`	"locale_2"	2

```
# Eseguiamo le istruzioni di assegnamento

variabile_locale_1 = "locale_1" # scope locale
variabile_locale_2 = "locale_2" # scope locale
```

Spazio delle variabili

Nome	Valore	Livello di visibilità
`variabile_globale`	"globale"	1

```
# La funzione conclude la sua chiamata
```

Shadowing di una variabile

- Lo **scope o ambito di visibilità** di una variabile è la parte di un programma all'interno del quale si può fare riferimento ad essa.
- Possiamo **oscurare** la visibilità di variabili di livello più basso assegnando dei valori differenti agli stessi nomi di variabili;
- Vediamo un esempio per capire il concetto partendo da quello precedente;

```
# Dichiariamo ora la funzione
def funzione_per_esaminare_scope(variabile_locale_1, variabile_locale_2):
  variabile_locale_1 = "locale_1" # scope locale
  variabile_locale_2 = "locale_2" # scope locale
  variabile_globale = "globale_modificata"
  print("Variabili con scope locali %s" % [variabile_locale_1,variabile_locale_2])
  print("Variabili con scope locali %s" % [variabile_globale])
```

Spazio delle variabili

Nome	Valore	Livello di visibilità
`variabile_globale`	"globale"	1
`variabile_locale_1`	"globale"	2
`variabile_locale_2`	"globale"	2

```
# Chiamiamo la funzione ricordando la definizione
# funzione_per_esaminare_scope(variabile_locale_1, variabile_locale_2)

funzione_per_esaminare_scope(variabile_globale,variabile_globale)
```

Non si sovrascrive la variabile a un livello inferiore!

Se ne crea una nuova che ha visibilità solo da quel livello in poi!

Spazio delle variabili

Nome	Valore	Livello di visibilità
`variabile_globale`	"globale"	1
`variabile_globale`	"globale_modificata"	2
`variabile_locale_1`	"locale_1"	2
`variabile_locale_2`	"locale_2"	2

```
# Eseguiamo le istruzioni di assegnamento

variabile_locale_1 = "locale_1" # scope locale
variabile_locale_2 = "locale_2" # scope locale
variabile_globale = "globale_modificata"
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Strutture dati

Strutture dati

- Finora abbiamo parlato di **tipi di dati** «semplici» che rappresentano valori «semplici» (es. numeri, stringhe);
- Le strutture dati sono invece dati più «**complessi**» poiché non contengono un valore ma una **sequenza di valori**;
- Le strutture dati vengono anche chiamate **sequenze**;
- Es. Una stringa è una **sequenza di caratteri**;
- Parleremo delle strutture **dati più note e utilizzate (anche nel mondo Big Data);**

LISTA

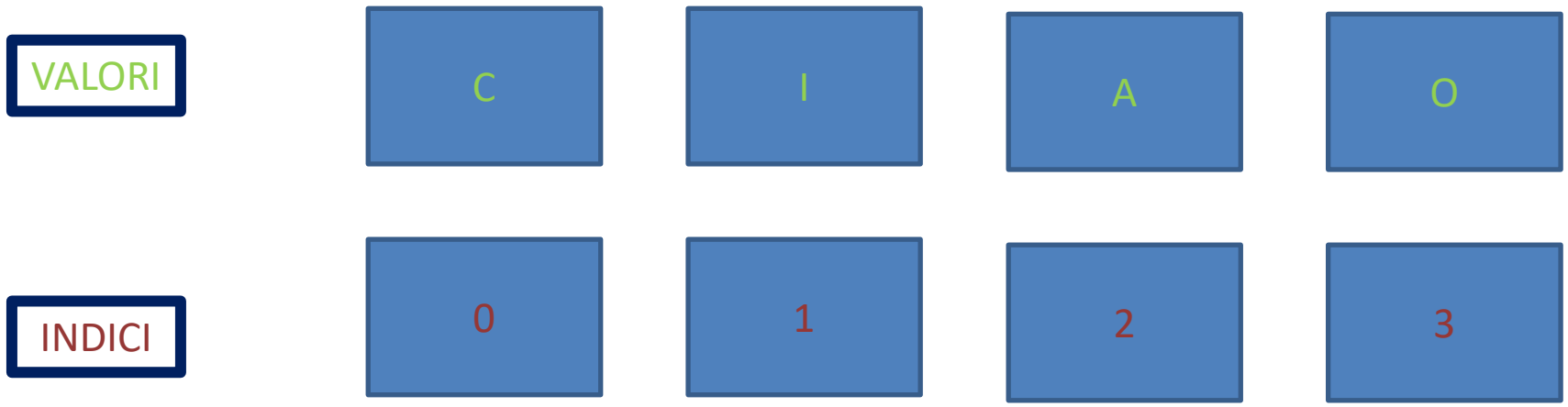
DIZIONARIO

SET

TUPLA

Concetto di sequenza

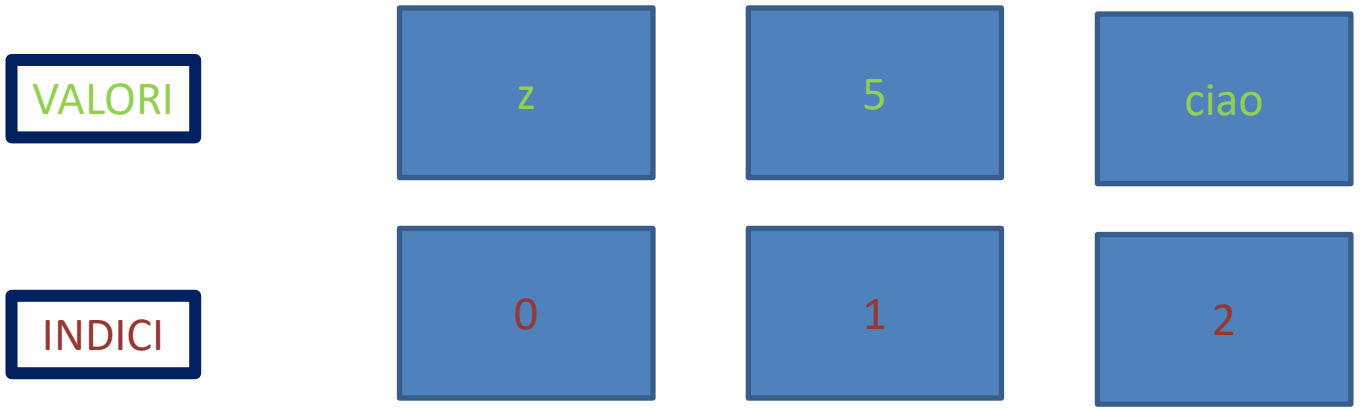
- Una sequenza, intesa come struttura dati, è sempre composta da tre fattori:
  - Un serie di valori;
  - L'indice di ogni elemento;
  - Il numero di elementi che questo insieme contiene;
- **La stringa è una sequenza di caratteri**! Prendiamo come esempio la stringa «CIAO»



Liste e Tuple

- Sia le **liste** che le **tuple** sono sequenze e possono contenere qualunque tipo di elementi (interi, caratteri, stringhe e etc…);

- Un'unica lista o un'unica tupla può contenere anche oggetti di tipo diverso (es. posso costruire una lista con un carattere, un intero e una stringa)



- Tuttavia esse differiscono per un aspetto fondamentale:

  - La lista è una struttura dati **mutabile,** una volta definita **può essere modificata**;

  - La tupla è una struttura dati **immutabile,** una volta definita **non può essere modificata**;

Liste

- Una lista è una **struttura dati mutabile** composta da 0 o più elementi che sono separati da virgole e racchiuse tra una coppia di parentesi quadre;

```
lista_vuota = []
print(lista_vuota)

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


lista_vuota_utilizzando_classe_callable = list()
print(lista_vuota_utilizzando_classe_callable)
```

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

- Come abbiamo detto, le sequenze sono composte da elementi che sono identificati tramite **indici:**
  - Se una lista ha n elementi, posso usare i numeri da **0** a **n-1** per accedere ai vari elementi;
  - Possiamo anche indicizzare al contrario utilizzando il segno negativo (comodo quando vogliamo gli ultimi elementi della lista);

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

print(lista_con_interi[0],lista_con_interi[1])

print(lista_con_interi[-1],lista_con_interi[-2])
```

Liste

- Possiamo anche indicizzare più elementi di una lista e prendere una sotto parte!
- Questa operazione è detta **slicing di una lista** e può essere effettuata allo stesso modo su tutte le **sequenze**;

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

# prendiamo i primi due elementi della lista
print(lista_con_interi[:2])# indice esclusivo, da 0 a n-1
```

```
[10, 20]
```

```
print(lista_con_interi[1:2])# indice esclusivo, da 1 a n-1
```

```
[20]
```

```
print(lista_con_interi[1:])# indice inclusivo, da 1 fino alla fine
```

```
[20, 30]
```

La lista è una struttura dati mutabile

- Come anticipato, una volta definita, una lista **può essere modificata**!
- Ci sono molti modi per modificare una lista:
  - Possiamo cambiare il valore di un elemento già presente;
  - Possiamo aggiungere nuovi elementi;
  - Possiamo rimuovere degli elementi;
- Ci sono anche modi diversi per effettuare la stessa operazione!
- Uno dei modi più semplici per manipolare una lista è attraverso **l'utilizzo dei metodi** esposti dalla classe **List**!

Metodi principali per manipolare una lista

- **Elemento** = valore di un qualunque tipo;
- **Indice** = valore o variabile intera che serve per gestire la manipolazione dei valori;

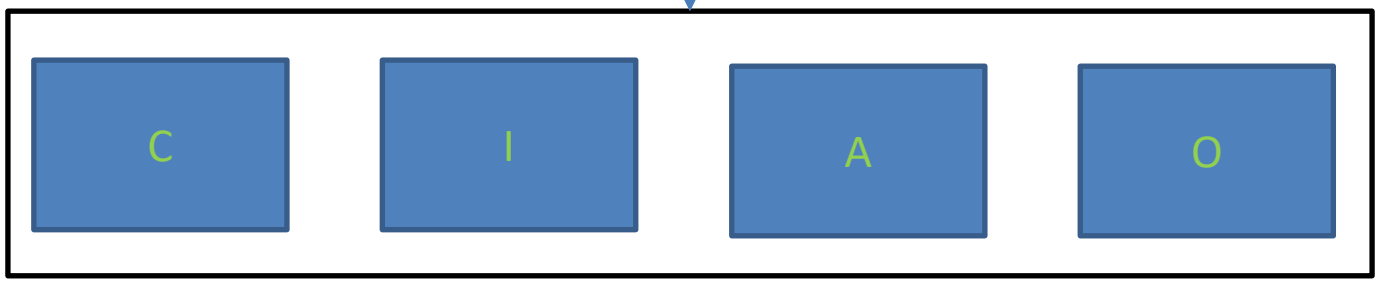
Nome metodo	Descrizione
append()	Aggiunge un elemento alla fine della lista
clear()	Rimuove tutti gli elementi della lista
copy()	Ritorna una copia della lista
count(elemento)	Conta le occorrenze di **elemento** all'interno della lista
extend()	Aggiunge una qualunque **sequenza** alla lista
index(elemento)	Ritorna l'indice della prima occorenza di **elemento**
insert(elemento, indice)	Aggiunge **elemento** nella posizione **indice** desiderata
pop(indice)	Rimuove l'element in posizione **indice**
remove(elemento)	Rimuove il primo valore **elemento** trovato
reverse()	Inverte l'ordine della lista
sort()	Ordina la lista

Memoria condivisa tra liste

- Se **assegno ad una nuova variabile il valore di un'altra variabile di tipo lista**, senza usare il metodo **copy(), la nuova variabile condividerà gli elementi con la prima!**

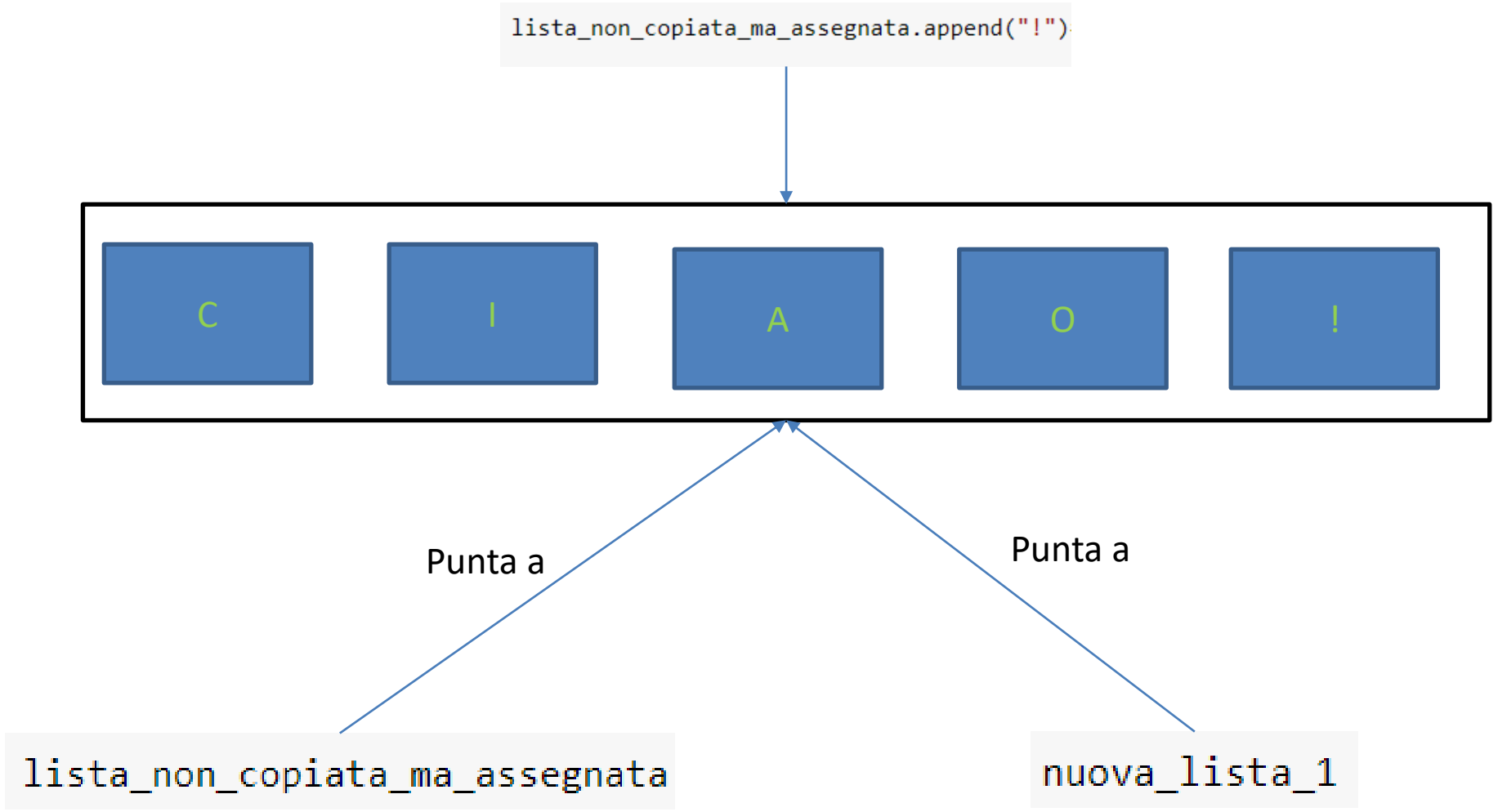
```
nuova_lista_1 = ["C","I","A","O"]

lista_non_copiata_ma_assegnata = nuova_lista_1
```



Memoria condivisa tra liste

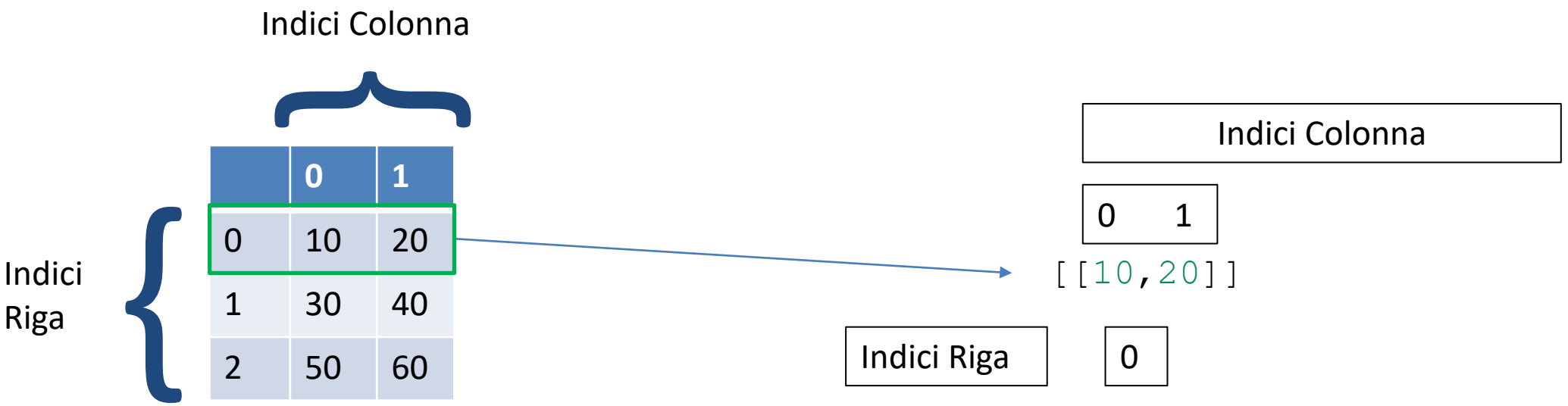
- Se effettuo operazioni sulla **lista_non_copiata_ma_assegnata influenzerò anche nuova_lista_1**



Let's code!

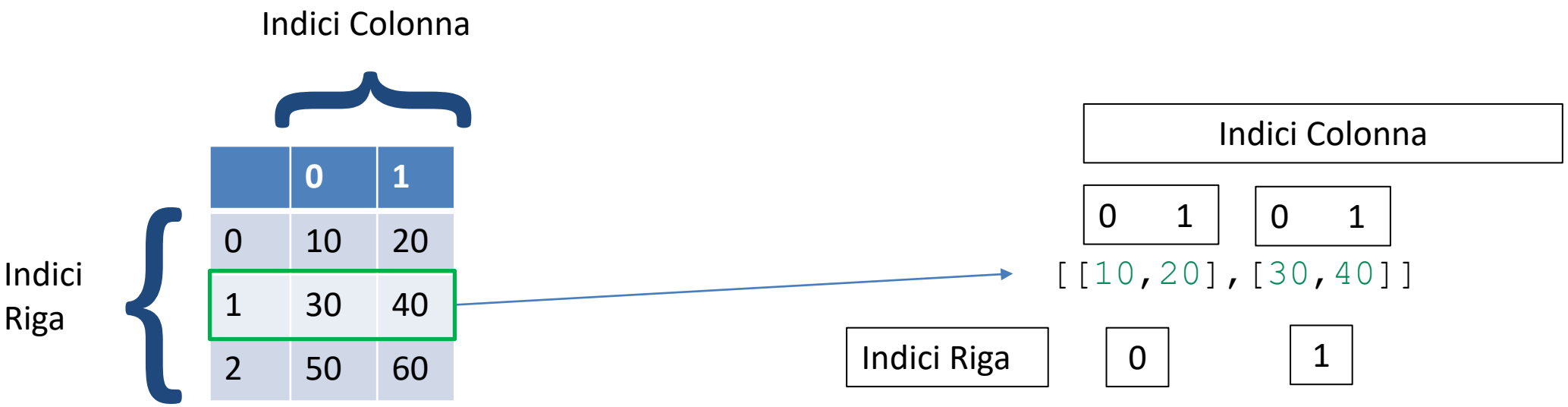
Liste bidimensionali

- Una lista bidimensionale è una **lista composta da liste;**
- Possiamo interpretarla come la rappresentazione schiacciata di una tabella formata da righe e colonne;



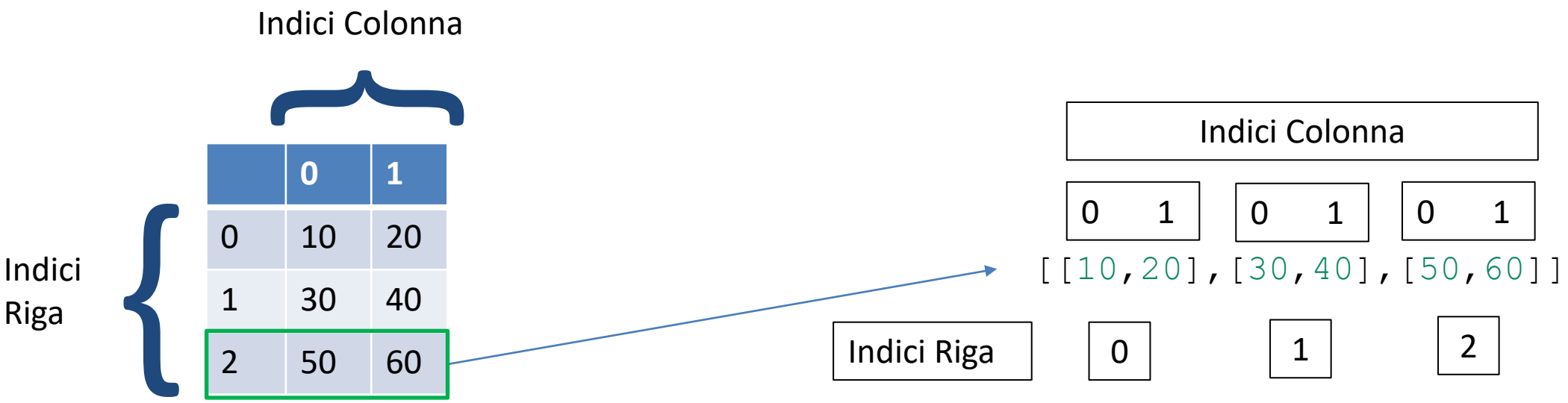
Liste bidimensionali

- Una lista bidimensionale è una **lista composta da liste;**
- Possiamo interpretarla come la rappresentazione schiacciata di una tabella formata da righe e colonne;



Liste bidimensionali

- Una lista bidimensionale è una **lista composta da liste;**
- Possiamo interpretarla come la rappresentazione schiacciata di una tabella formata da righe e colonne;



Tuple

- Una tupla è una **struttura dati immutabile** composta da 0 o più elementi che sono separati da virgole e racchiuse tra una coppia di parentesi tonde;

```
tupla_vuota = ()

print(tupla_vuota, type(tupla_vuota))


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

tupla_vuota_utilizzando_classe_callable = tuple()
print(tupla_vuota_utilizzando_classe_callable)
```

- Come le liste, anche le tuple hanno elementi che sono identificati tramite **indici.** Possiamo effettuare le stesse operazioni di indicizzazione viste con le liste:
  - Indicizzazione a un elemento specifico tramite indice;
  - Slicing;
  - …

```
tupla_con_interi = (5,6,7)

print(tupla_con_interi[0])

print(tupla_con_interi[1:])
```

```
5
(6, 7)
```

Le Tuple sono strutture dati immutabili

- Una volta inizializzate, le tuple non possono essere modificate:
  - Non si possono inserire nuovi valori;
  - Non si possono cancellare valori;
  - Non si possono modificare valori;

Metodo	Descrizione
count(elemento)	Ritorna il numero di volte in cui **elemento compare nella tupla**
index(elemento)	Cerca nella tupla l'indice di prima occorrenza di **elemento**

- Perché usare la tupla invece di una lista?

La tupla è ottimizzata ed è molto più efficiente della lista in termini di velocità e uso di memoria!

Dizionario

- In Python i dizionari sono implementate dalla classe **Dict**;
- Un dizionario è una **struttura dati mutabile** composta da coppie **chiave-valore;**
- L'unico valore immutabile di un dizionario è proprio la chiave (per eliminarla, bisogna eliminare la coppia chiave-valore per intero!);
- **Tutte le chiavi sono uniche fra loro**!

```
dizionario_vuoto = {}

print(dizionario_vuoto, type(dizionario_vuoto))

dizionario_con_interi = {"uno":1, "due":2}
print(dizionario_con_interi)

dizionario_vuoto_utilizzando_classe_callable = dict()
print(dizionario_vuoto_utilizzando_classe_callable)
```

- Come le liste, anche le tuple hanno elementi che sono identificati tramite **indici.** In questo caso però, gli indici non sono numerici, ma sono rappresentati dalle **chiavi** delle varie coppie!
- Non esiste il concetto di slicing, ma possiamo simularlo attraverso l'uso dei **cicli** (che vedremo fra poco)!

```
dizionario_con_interi = {"uno":1, "due":2}
print(dizionario_con_interi["uno"])
```

Dizionario è una struttura dati mutabile e non ordinata

- Come dicevamo, l'unico valore immutabile di un dizionario è proprio la chiave;
- Se vogliamo eliminarla, bisogna eliminare la coppia chiave-valore per intero, usando il comando **del**;
- Possiamo modificare in qualunque momento il valore di una coppia chiave-valore già esistente;

- **Due cose dovrebbero lasciarvi perplessi:**
  - Non abbiamo mai usato indici numerici → Questo significa che non c'è ordinamento, a meno che non lo definiamo in maniera esplicita!
  - Perché abbiamo usato una stringa come chiave? → Possiamo usare qualunque valore come chiave, anche se io consiglio di usare valori primitivi come stringhe, numeri etc…

Perché usare i dizionari?

- **I dizionari sono altamente efficienti e soprattutto comodi da usare;**
- Pensiamo al caso della profilazione di un utente:
  - Se usassi una lista o una tupla, dovrei sapere in quale indice l'utente si trova;
  - Se uso un dizionario, posso indicizzare tramite il suo username!

Metodo	Descrizione
clear()	Rimuove tutti gli elementi dal dizionario
copy()	Ritorna una copia del dizionario
fromkeys((chavi,valori))	Ritorna un dizionario con specifiche chiavi e valori
get(chiave)	Ritorna il valore di una specifica **chiave**
items()	Ritorna una lista di tuple chiave-valore del dizionario
keys()	Ritorna la lista delle chiavi del dizionario
pop(chiave)	Rimuove l'elemento **chiave**
popitem()	Rimuove l'ultima coppia inserita
setdefault(chiave, valore)	Ritorna il valore della **chiave** e se **non esiste** inserisce la chiave con il **valore** specificato
update((chavi,valori))	Aggiorna il dizionario aggiungendo le chiavi e I valori specificati
values()	Ritorna una lista di tutti I **valori** del dizionario

Set

- In Python gli insiemi sono implementate dalla classe **Set**;
- Un insieme è come un dizionario composto solamente da **chiavi**:
  - Tutti i valori sono unici e mai ripetuti;
  - Come i dizionari non hanno un ordinamento!
  - Possiamo usare tutte le operazioni insiemistiche che conosciamo dalla matematica (es. intersezione, differenza)!

```
insieme_vuoto = set()
print(insieme_vuoto,type(insieme_vuoto))

insieme_di_interi = {1,2,3}
print(insieme_di_interi,type(insieme_di_interi))
```

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

- A differenza di un dizionario, non abbiamo modo di indicizzare i valori!

```
insieme_di_interi = {1,2,3}
print(insieme_di_interi[1]) # errore
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-fc5e639fd687> in <module>()
      1 insieme_di_interi = {1,2,3}
----> 2 print(insieme_di_interi[1])

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

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Strutture di codice

Blocchi di codice

- Finora, abbiamo utilizzato blocchi di codice senza saperne la nomenclatura;
- Un blocco di codice è semplicemente formato da due o più linee di codice raggruppate con lo stesso livello di indentazione!
- Un livello di indentazione è formato da 4 spazi!

Blocco di codice da 4 righe

Blocco di codice da 1 riga

```
numero_1 = int(input('Inserisci il primo numero intero: ')) # castiamo a intero per evitare di inse
numero_2 = int(input('Inserisci il secondo numero intero: ')) # castiamo a intero per evitare di in

numero_1_maggiore_numero_2 = primo_numero_maggiore_uguale_numero2(numero_1,numero_2)

if numero_1_maggiore_numero_2 == None:
    raise NotImplementedError("Non ha implementato la funzione primo_numero_maggiore_uguale_numero2")
```

Statement

- Singole istruzioni di codice vengono definite **statement semplici**

Statement semplice

```
numero_1 = int(input('Inserisci il primo numero intero: ')) # castiamo a intero per evitare di inse
numero_2 = int(input('Inserisci il secondo numero intero: ')) # castiamo a intero per evitare di in
numero_1_maggiore_numero_2 = primo_numero_maggiore_uguale_numero2(numero_1,numero_2)
if numero_1_maggiore_numero_2 == None:
  raise NotImplementedError("Non ha implementato la funzione primo numero maggiore uguale numero2")
```

- Istruzioni che sono scomposte in più linee di codice e linee logiche sono dette **statement composti**

Statement composto

```
numero_1 = int(input('Inserisci il primo numero intero: ')) # castiamo a intero per evitare di inse
numero_2 = int(input('Inserisci il secondo numero intero: ')) # castiamo a intero per evitare di in
numero_1_maggiore_numero_2 = primo_numero_maggiore_uguale_numero2(numero_1,numero_2)
if numero_1_maggiore_numero_2 == None:
  raise NotImplementedError("Non ha implementato la funzione primo numero maggiore uguale numero2")
```

Lo statement composto If: l'operatore di controllo condizionale

- Il costrutto **if-elif-else,** verifica la veridicità di una o più espressioni booleane;

- Questo è un **operatore di controllo** poiché **in base alla veridicità di una espressione booleana potremmo o meno eseguire un diverso blocco di codice**!

- Se expression assumesse valore True, allora eseguiremmo il blocco di codice 1 **altrimenti**

- Se expression_2 assumesse valore True, allora eseguiremmo il blocco di codice 2 **altrimenti**

- …

- Se nessuna espressione dovesse verificarsi vera, eseguiremmo il blocco di codice n

```
If expression:
        blocco di codice 1
Elif expression_2:
        blocco di codice 2
Elif … :
        blocco di codice …
Else:
        blocco di codice n
```

Let's code!

Operatore di controllo

- **Un operatore di controllo riesce a controllare il flusso del programma;**
- Finora, abbiamo dichiarato istruzioni che si eseguivano una dopo l'altra;
- D'ora in poi potremmo decidere di controllare il flusso di esecuzione permettendo:
  - L'esecuzione **condizionale** di una riga o un blocco di codice;
  - L'esecuzione **ripetuta** di una riga o un blocco di codice finché un'espressione sia verificata;
  - L'esecuzione **ripetuta** di una riga o un blocco di codice per un numero definito di volte;
  - L'esecuzione di una modifica sugli elementi di **una struttura dati** (scorrimento);

} CICLI

La guerra dei Cicli

- **Un ciclo è** può essere definito come **una o più righe di codice eseguite** finché non si **presenta una condizione d'uscita;**

- Sono utili **perché consentono di creare una rappresentazione compressa** di interi blocchi di codice;

La guerra dei Cicli episodio 1: Il costrutto While

```
while True/expression:
        blocco di codice
        if expression:
                break
        blocco di codice
```

- **Un ciclo while** esegue **una o più righe di codice** finché una **espressione booleana** rimane True;
- **Di base potremmo costruire un ciclo infinito**, creando un programma che possa durare ipoteticamente fino alla fine dei tempi … sapreste dirmi come?
- Nel ciclo while è consentito usare il costrutto condizionale **if** per creare una condizione d'uscita tramite il costrutto break;

La guerra dei Cicli episodio 2: Il costrutto for

```
For element in sequence:
        blocco di codice
        if expression:
                break
```

- **Un ciclo for** esegue **una o più righe di codice per ogni** elemento di una sequenza (es. stringhe, liste)**;**
- Tipicamente utilizzato per effettuare **manipolazioni** sugli elementi della sequenza o **azioni** in base al valore dell'elemento;
- Anche in questo caso possiamo usare l'operatore di controllo condizionale e uscire dal ciclo (non succede spesso);

La guerra dei Cicli episodio 3: Iterare per un certo numero di volte

- La funzione **range(start, stop, step)** consente di iterare su una sequenza di interi, che partono da **start** e finiscono in **stop** effettuando ogni volta un salto pari a **step**;
- **Stop** è però considerato non inclusivo!
- Ad esempio, la chiamata alla funzione **range(2,10,2)**, creerebbe la seguente sequenza:

[2,4,6,8]

- La chiamata alla funzione **range(2,11,2)**, creerebbe la seguente sequenza:

[2,4,6,8,10]

Come fossero i programmatori senza cicli (cit.)

Fonte della cit

Gli oggetti

Python è un linguaggio orientato agli oggetti

- Python è un linguaggio che non solo permette di definire dei software definiti tramite oggetti!
- La programmazione ad oggetti è intensivamente usata… per ragioni che non spiegheremo;
- Non siamo qui per introdurre l'intero paradigma di programmazione ad oggetti ma solamente farvi capire cos'è un oggetto;
- Questo perché utilizzeremo durante il corso delle prossime lezioni **Classi, oggetti e metodi** senza però conoscere del tutto la loro implementazione!

Cos'è una classe?

- Una classe è composta da una serie di **attributi** e una serie di **metodi**;
- Gli attributi sono tipicamente detti «dati di una classe» cioè dati che in qualche modo assieme rappresentano un'entità;

```
Class persona:
        nome;
        cognome;
        età;
        codice fiscale;
```

Cos'è un oggetto

- Un'oggetto non è altro che una istanza di una classe;
- Una istanza di una classe non è altro che la classe stessa a cui però vengono forniti dei valori per gli attributi!

persona_lorenzo = persona(lorenzo, stacchio, 24, …)

Attributi passati come se
fossero argomenti di funzioni!

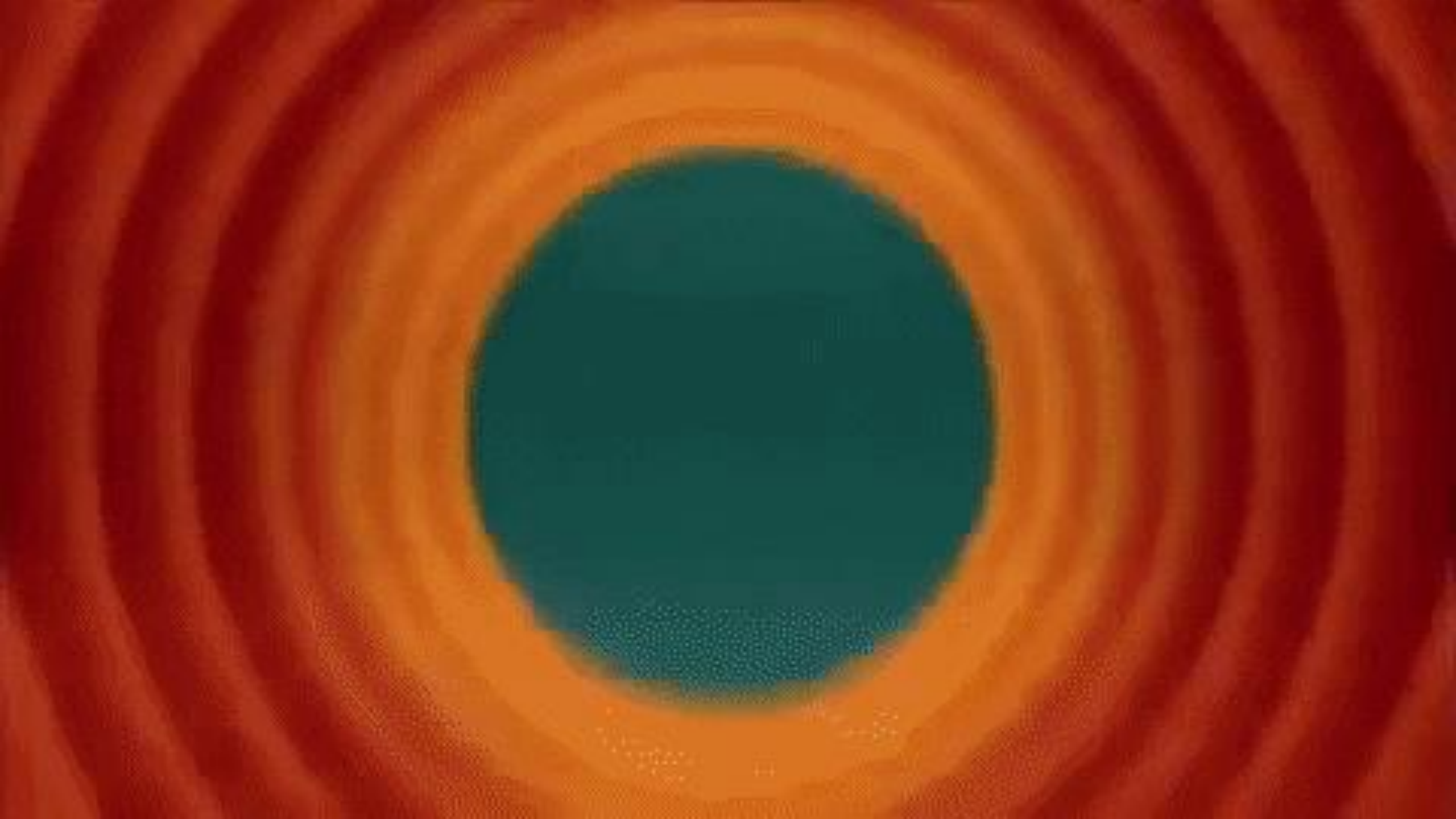
Cos'è un metodo?

- Inserire dei valori all'interno degli attributi definisce un'oggetto con un certo stato!
- Tuttavia, abbiamo bisogno di un costrutto che definisca un modo per manipolare questo stato!
- Questo costrutto sono proprio i metodi!
- **I metodi** possono essere semplicisticamente definiti come delle funzioni di un oggetto che ne modificano lo stato!

```
Class persona:
        nome;
        cognome;
        età;
        codice fiscale;

        def incrementa_età():
            if oggi == compleanno:
                età +=1
```

Abbiamo tutti gli strumenti per creare una Eliza in miniatura!



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Lorenzo Stacchio

Dipartimento di Scienze per la Qualità della Vita

lorenzo.stacchio2@unibo.it