Il linguaggio Python

Liceo G.B. Brocchi – Bassano del Grappa (VI)
Liceo Scientifico – opzione scienze applicate
Giovanni Mazzocchin

Il vostro futuro linguaggio preferito

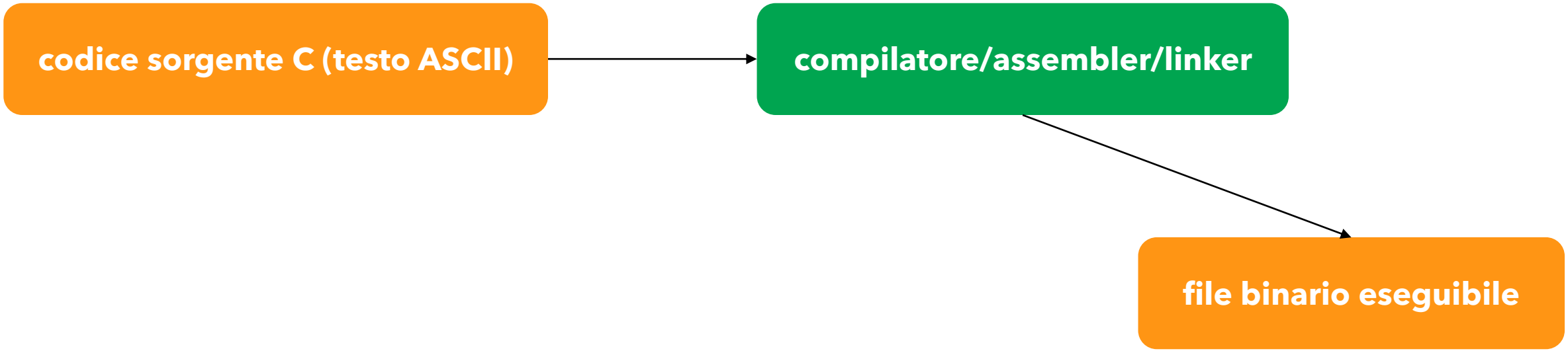
Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Il vostro futuro linguaggio preferito

- Creato da **Guido van Rossum** a fine anni '80
- La sintassi è piuttosto semplice: ricorda uno *pseudocodice con parole chiave in inglese*
- È un linguaggio **ad alto livello**: anche il C e il C++ sono ad alto livello, ma sono molto più vicini all'hardware rispetto a Python
- Python può essere utilizzato su qualsiasi sistema
- Le librerie disponibili per Python sono moltissime e facili da usare: potete usare questo linguaggio per fare machine learning, sviluppo web, crittografia, calcolo scientifico etc..

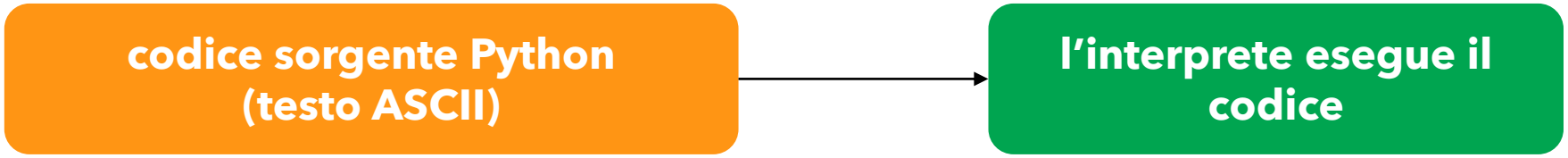
I linguaggi compilati

Il C e il C++ sono linguaggi compilati. Il codice sorgente viene convertito in codice macchina da un compilatore. Il risultato della compilazione (e del linkaggio), ossia un file eseguibile, viene caricato in memoria RAM ed eseguito dalla CPU, ossia dall'hardware del computer



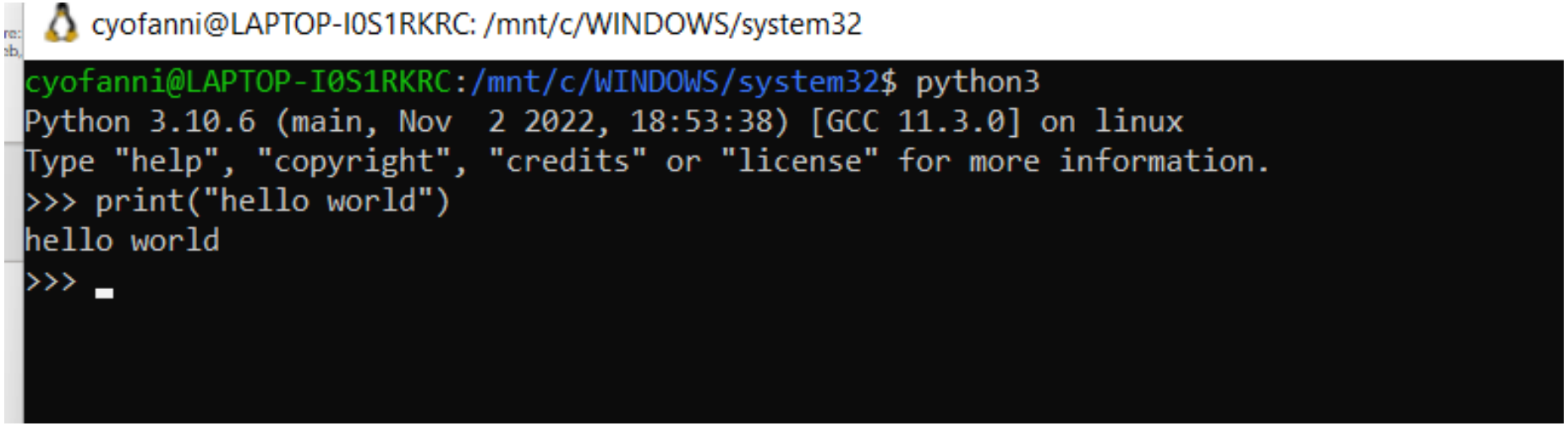
I linguaggi interpretati

Python, JavaScript, PHP (e altri) sono linguaggi interpretati.
Nel caso di Python, l'interprete (`python`) esegue direttamente il codice sorgente.
In realtà esiste un processo di traduzione in codice macchina interno a Python, ma al programmatore non interessa.
Il programmatore deve solo scrivere il codice e chiedere a Python di eseguirlo.
È abbastanza simile a quello che succede quando la shell interpreta i vostri comandi



Interprete interattivo vs file sorgente

- Potete utilizzare Python dall'interprete interattivo
  - lanciate semplicemente il comando **`python`**, o **`python3`**, oppure **`ipython3`** da una shell Linux (o dal prompt di Windows)
  - c'è anche l'opzione Google Colab
  - si aprirà una shell in grado eseguire istruzioni Python
  - l'istruzione **`quit()`** chiude l'interprete interattivo



Interprete interattivo vs file sorgente

- Non avrebbe senso riscrivere programmi interi ogni volta nell'interprete interattivo
- Per scrivere un programma una volta sola ed eseguirlo tutte le volte che si vuole bisogna salvarlo in un file:
  - scrivere il codice in un file (*script*) con estensione **`.py`**
  - per eseguire lo script: **`python filename.py`**

Variabili, stringhe, `format()`

```
#no need to write a main function
year = 1756    #no need to write the type
name = 'Wolfgang Amadeus Mozart'    #no explicit type, notice the single quote
work = 'The Magic Flute'    #notice the single quote

print('{0} was born in {1}'.format(name, year))
print('his most famous opera is {0}'.format(work))
```

Operatori matematici

- `2 * 3` restituisce 6
- `'hello' * 4` restituisce `'hellohellohellohello'`
- `2 ** 5` restituisce $32 = 2^5$
- `15 // 2` divisione intera, restituisce 7

Un piccolo script

```
import math

radius = 5

sphere_area = 4 * math.pi * radius**2
sphere_volume = (4/3) * math.pi * radius**3

print('Area of sphere of radius: {0} cm, is: {1} cm^2'.format(radius, sphere_area))
print('Volume of sphere of radius: {0} cm, is: {1} cm^3'.format(radius, sphere_volume))
```

```
Usage:  python3 sphere.py
```

Un piccolo script

```
#!/usr/bin/python3
import math

radius = 5

sphere_area = 4 * math.pi * radius**2
sphere_volume = (4/3) * math.pi * radius**3

print('Area of sphere of radius: {0} cm, is: {1} cm^2'.format(radius, sphere_area))
print('Volume of sphere of radius: {0} cm, is: {1} cm^3'.format(radius, sphere_volume))
```

- si specifica il percorso in cui si trova l'interprete Python installato sulla vostra macchina
- si utilizza lo **shebang #!**
- prima bisogna capire dove è posizionato l'eseguibile dell'interprete Python: lanciate **`which python3`** dalla shell Bash

```
make the file executable: chmod 755 sphere.py
now you can run it with: ./sphere.py
```

Le strutture di controllo

```
number_str = input('enter a number: ')
number_int = int(number_str)

if number_int % 2 == 0:
    print('you entered an even number')
else:
    print('you entered an odd number')
```

l'input da tastiera (standard input) è di tipo stringa (`str`). Se vogliamo trattarlo come numero, dobbiamo effettuare una conversione di tipo

conversione da stringa a intero (da `str` a `int`)

Le strutture di controllo

```
while True:
    print('python is great')


i = 1
while i <= 10:
    i += 1
print(i)
```

il costrutto `while` è molto simile a quello dei linguaggi C-like

Le strutture di controllo

```
for i in [2, 5, 3, 1, -12, 3, 9]:
    print(i, '\t', end='')
print()
```

il costrutto `for` di Python è un po' particolare

```
output:     2       5       3       1       -12     3       9
```

Le strutture di controllo

```
for item in ['3AQSA', '2AQSA', '3BSA', '4BSA', '1ASA', '2ASA', '3ASA']:
    print(item, '\t', end='')
print()
```

```
output:    3AQSA    2AQSA    3BSA    4BSA    1ASA    2ASA    3ASA
```

sembra proprio che il `for` non lavori sugli indici, ma direttamente sugli elementi dell'oggetto specificato dopo la keyword `in`

di fatto, si tratta di un foreach

Le liste

```
l1 = [1, 2, 3, 6, 1, 2]
l2 = [[2, 6, 3], [1, 6, 4]]
l3 = ['alan turing', 'john von neumann', 'gottfried leibniz']
l4 = ['banana', 1, 5, 3.4, l1, l2, 'strawberry']
```

le liste di Python sono molto più comode e flessibili degli array dei linguaggi di livello più basso

come potete notare, le liste possono contenere elementi di tipo diverso