



# Introduzione a Python

Corso Programmazione Python 2024  
Modulo 1

*Luca Di Pietro Martinelli*

Parte del materiale deriva dai corsi dei proff. Paolo Caressa e Raffaele Nicolussi (Sapienza Università di Roma) e Giorgio Fumera (Università degli Studi di Cagliari)

# Python

È un linguaggio di programmazione:



- Interpretato
- Ad alto livello di astrazione
- Orientato agli oggetti
- Semplice da imparare e usare
- Potente e produttivo
- Ottimo anche come primo linguaggio (poiché molto simile allo pseudocodice)
- Estensibile

Inoltre è:

- Open source ([www.python.org](http://www.python.org))
- Multiplatforma
- Facilmente integrabile con C/C++ e Java.

# Un po' di storia



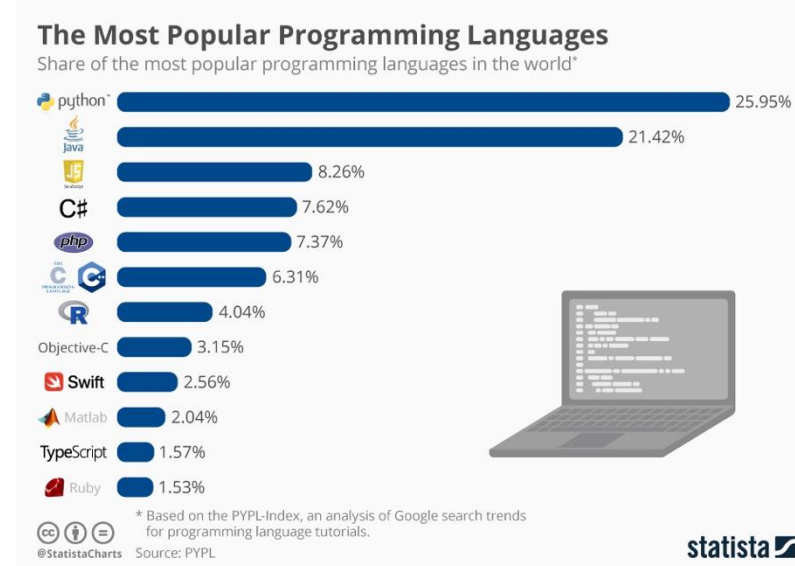
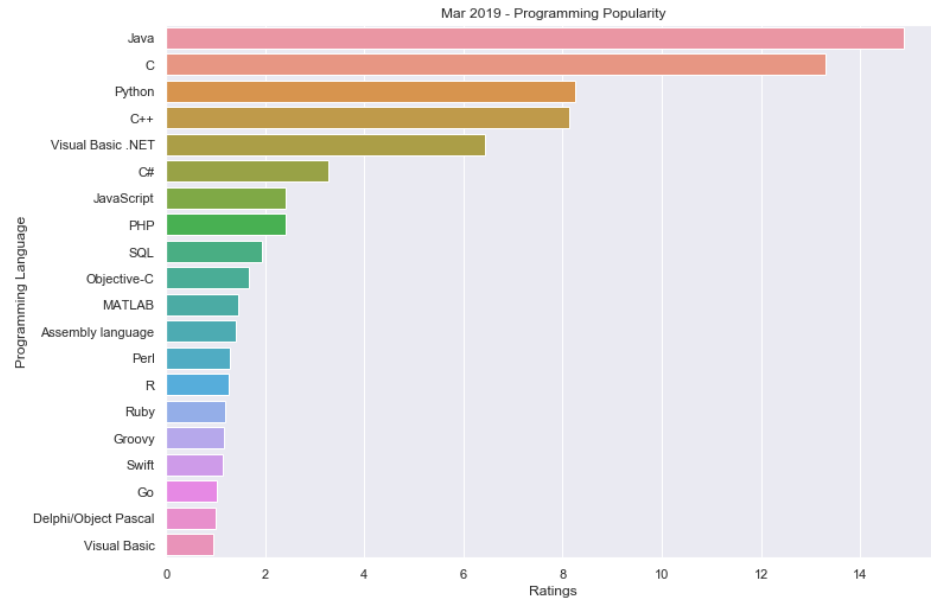
Python è stato creato nel 1989 dall'informatico olandese Guido van Rossum.

Ideato originariamente come linguaggio di scripting (linguaggio non compilato, destinato ad automazione o esecuzione su browser), Python si è poi evoluto come linguaggio completo.

Il nome fu scelto per via della passione di van Rossum per i Monty Python (<http://www.montypython.com>) e per la loro serie televisiva *Monty Python's Flying Circus*.

Attualmente l'ultima versione stabile è la 3.13, rilasciata a Ottobre 2024.

# Diffusione di Python



Python è ormai diventato uno tra i linguaggi di programmazione più famosi e utilizzati

# Iniziare a lavorare con Python

Python può essere installato su macchine Windows, Linux/UNIX e macOS scaricando la versione desiderata dalla pagina ufficiale dei download:

<https://www.python.org/downloads/>.

Assieme a questo viene fornito un IDE (Integrated Development Environment) denominato *IDLE* (Python's *Integrated Development and Learning Environment*) con cui si può iniziare fin da subito a lavorare.

In alternativa è possibile scaricare un altro tra gli IDE che supportano Python tramite cui poter cominciare a scrivere programmi. In questo corso utilizzeremo il software *Visual Studio Code*: <https://code.visualstudio.com/>

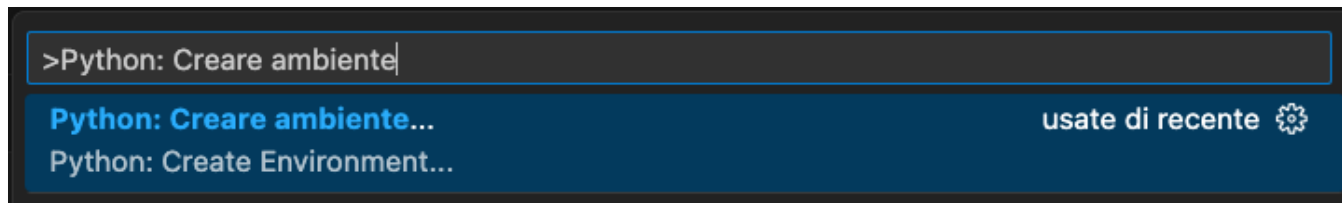


# Iniziare a lavorare con Python

Nella schermata iniziale di *Visual Studio Code* cliccate su **Apri Cartella**, dopodiché selezionate una cartella esistente o createne una nuova dove volete salvare i file del progetto.

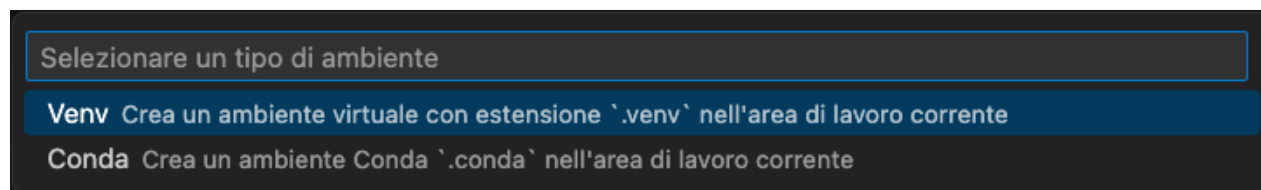
È ora possibile creare un *virtual environment*, ovvero un ambiente di lavoro specifico configurato per poter realizzare un programma scritto in Python.

Per fare questo è sufficiente aprire la palette dei comandi premendo contemporaneamente **Ctrl+Shift+P** su Windows e Linux o **Cmd+Shift+P** su Mac OS e digitare "*Python: Creare ambiente*" o "*Python: Create environment*".

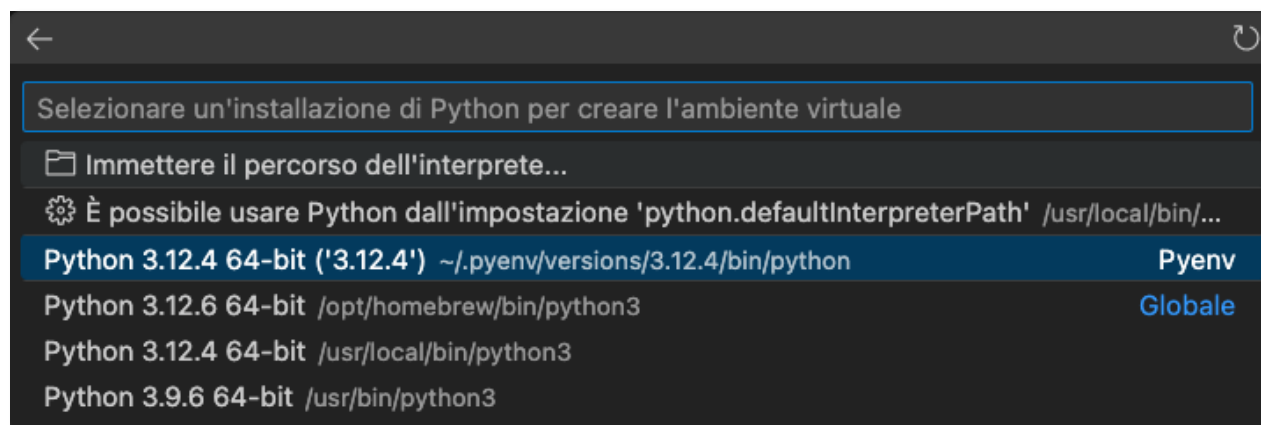


# Iniziare a lavorare con Python

È necessario ora selezionare il tipo di ambiente virtuale tra *Venv* e *Conda*: procediamo pure con il primo.

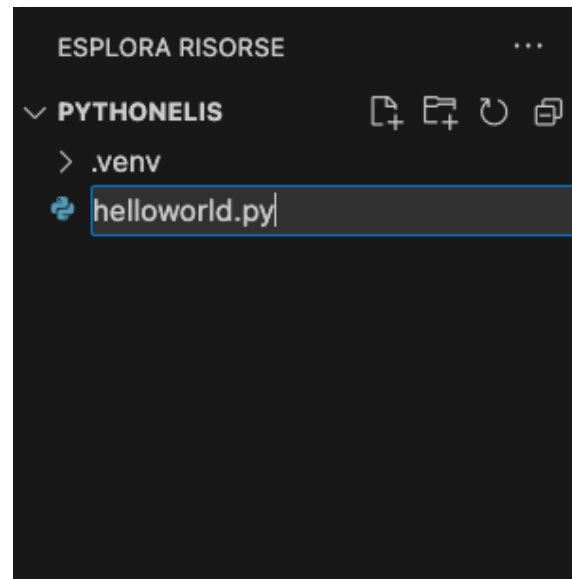


A questo punto viene fatto scegliere l'interprete che permetterà l'esecuzione dei programmi.



# Iniziare a lavorare con Python

Nella colonna relativa alla cartella del progetto fate tasto destro e selezionate **Nuovo File**, digitate un nome qualsiasi compreso di estensione `.py` e date Invio. *Visual Studio Code* crea un nuovo file Python e lo apre automaticamente per lavorarci.

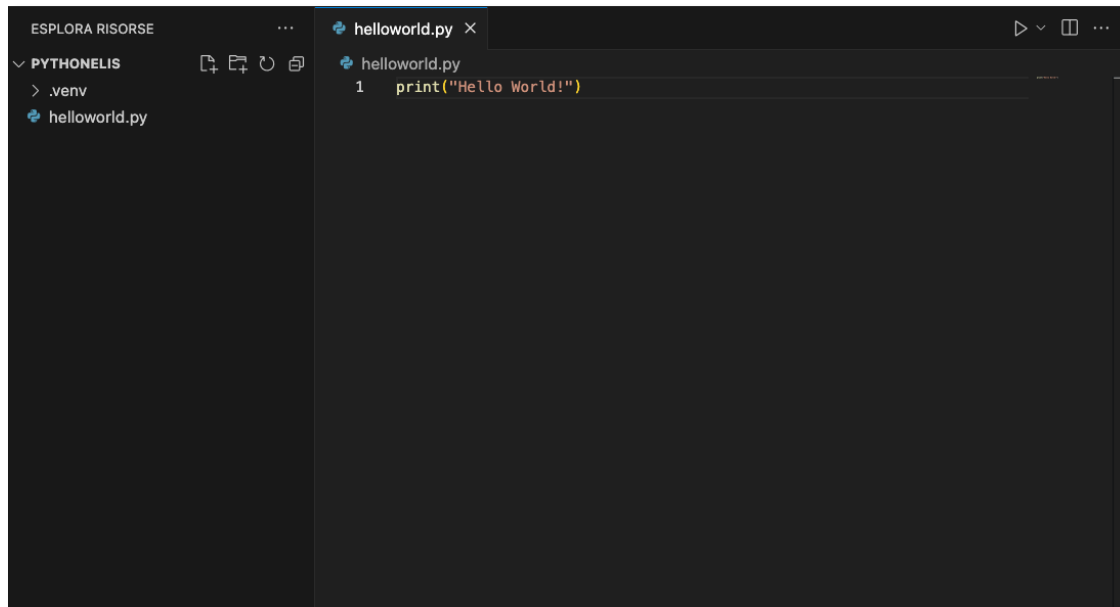




# Primo programma: Hello World

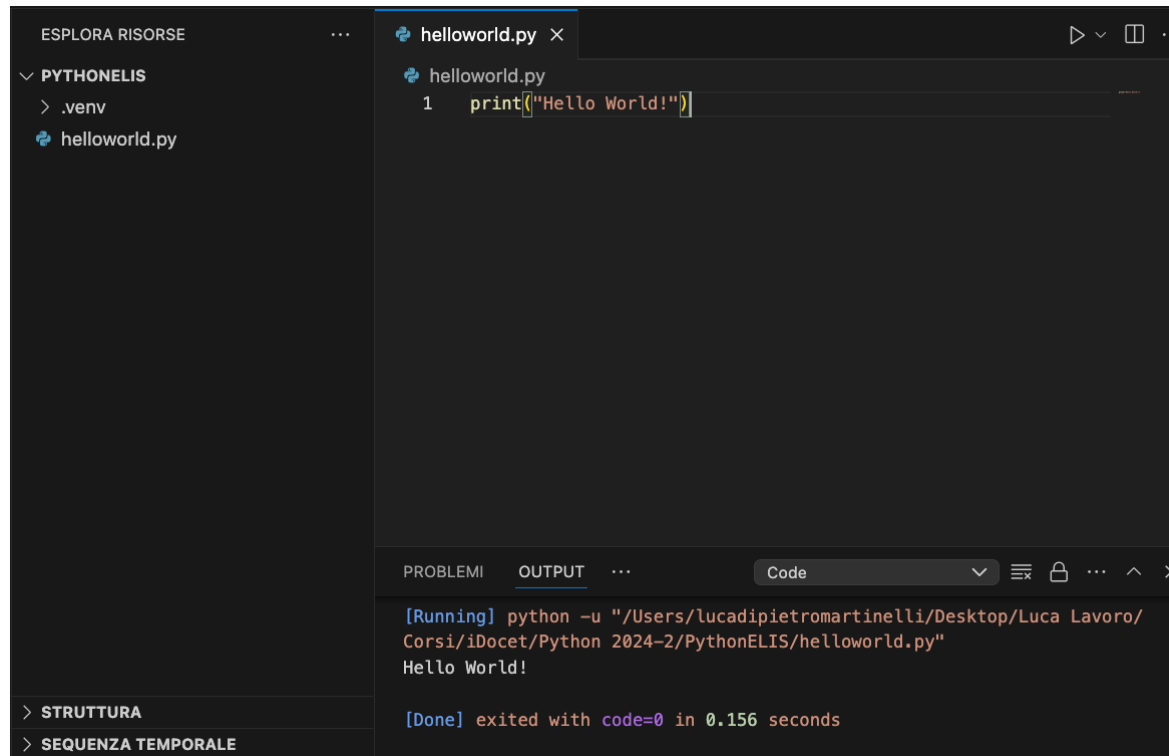
Il nostro obiettivo è quello di scrivere un programma che stampi nella console (la barra in basso) la scritta "Hello World". L'istruzione che permette di scrivere sulla console è `print()`.

Digitate l'istruzione e inserite la stringa di testo "Hello World!" (compresa di doppie virgolette) all'interno delle parentesi tonde e premete il tasto in alto a destra con il simbolo Play per avviare il programma.

A screenshot of a Python IDE interface. On the left, a sidebar titled 'ESPLORA RISORSE' shows a file explorer with 'PYTHONELIS' expanded, containing '.venv' and 'helloworld.py'. The main editor area has a tab for 'helloworld.py' with a single line of code: `1 print("Hello World!")`. The code is written in a dark theme with syntax highlighting. In the top right corner of the editor, there are icons for running (a play button), saving, and other functions.

# Primo programma: Hello World

Se il vostro programma ha stampato "Hello World" come in questo esempio avete completato l'esercizio e scritto il vostro primo programma in Python!



The screenshot shows a Python IDE interface. On the left, a file explorer under 'PYTHONELIS' shows a '.venv' folder and a 'helloworld.py' file. The main editor window displays the code for 'helloworld.py':

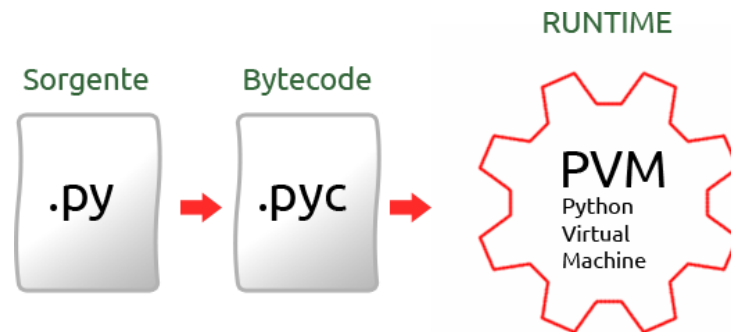
```
1 print("Hello World!")
```

At the bottom, the 'OUTPUT' panel shows the execution results:

```
[Running] python -u "/Users/lucadipietromartinelli/Desktop/Luca Lavoro/Corsi/iDocet/Python 2024-2/PythonELIS/helloworld.py"  
Hello World!  
[Done] exited with code=0 in 0.156 seconds
```

# Linguaggio interpretato

Come detto precedentemente Python è un linguaggio **interpretato**, il che significa che un interprete legge ed esegue il codice direttamente, senza compilazione.



Ogni volta che viene eseguito, il codice scritto viene scansionato per token, ognuno dei quali viene analizzato dentro una struttura logica ad albero che rappresenta il programma.

Tale struttura viene, infine, trasformata in bytecode (file con estensione .pyc). Per poter eseguire questi bytecode si utilizza un apposito interprete noto come macchina virtuale Python (PVM).

# Variabili

La **variabile** è un identificatore, ovvero un nome, che fa riferimento a un oggetto memorizzato in memoria.

Sintassi:

```
identificatore = espressione
```

**Identificatore:** nome che il programmatore assegna a una risorsa nel codice.

Regole per la definizione di una variabile:

- non devono esserci spazi né prima né all'interno dell'identificatore
- l'identificatore deve essere un nome simbolico scelto dal programmatore (con alcune limitazioni descritte più avanti)
- l'espressione indica il valore (per esempio un numero) che si vuole associare alla variabile.

Esempi:

```
x = -3.2
```

```
messaggio = "Buongiorno"
```

```
y = x + 1
```

# Variabili

Regole sugli identificatori:

- possono essere composti da uno o più dei seguenti caratteri:
  - lettere minuscole e maiuscole (NB: «A» e «a» sono considerate lettere diverse poiché Python è un linguaggio *case sensitive*)
  - cifre
  - il carattere `_` (underscore)

Esempi: `x`, `somma`, `Massimo_Comune_Divisore`, `y_1`

Limitazioni sugli identificatori:

- non devono iniziare con una cifra (esempio: `12abc` non è un identificatore ammesso)
- non devono contenere simboli matematici (+, -, /, \*, ecc.)
- non devono coincidere con le keyword del linguaggio:

`and`  
`continue`  
`except`  
`is`  
`print`  
`yield`

`as`  
`def`  
`exec`  
`lambda`  
`raise`  
`True`

`assert`  
`del`  
`if`  
`not`  
`return`  
`False`

`break`  
`elif`  
`import`  
`or`  
`try`  
`None`

`class`  
`else`  
`in`  
`pass`  
`while`

# Memory management

In Python la gestione della memoria è un processo che ha il compito di allocare e rilasciare lo spazio di memoria necessario agli oggetti e alle strutture dati.

All'interno della memoria RAM è infatti presente uno spazio *heap* privato destinato a Python che contiene tutti gli oggetti e le strutture dati. Lo spazio *heap* è una zona di memoria dinamica che viene assegnata a un programma in esecuzione all'interno della quale gli oggetti Python vengono creati, modificati e distrutti.

Lo spazio *heap* privato è gestito da un **gestore della memoria**, che è un componente del codice Python responsabile di allocare e rilasciare lo spazio di memoria per gli oggetti.

# Memory management

Il gestore della memoria in Python è composto da diversi componenti che si occupano di aspetti diversi della gestione della memoria. I principali sono:

- ▶ **Allocatore dell'interprete:** è il componente che interagisce direttamente con il sistema operativo per richiedere e rilasciare lo spazio heap.
- ▶ **Allocatore degli oggetti:** è il componente che si occupa di creare e distruggere gli oggetti Python nello spazio heap.
- ▶ **Garbage Collector:** è il componente che si occupa di rilevare e rimuovere gli oggetti Python che non sono più usati dal programma. Il raccoglitore dei rifiuti usa diversi algoritmi per identificare gli oggetti non raggiungibili, tra cui in particolare l'algoritmo di *Reference Counting* che ha come scopo quello di identificare gli oggetti non più utilizzati ed eliminarli dallo spazio heap della memoria. Tale algoritmo ha alcuni vantaggi evidenti come la semplicità e l'immediatezza, ma ha anche alcuni svantaggi come il costo aggiuntivo per mantenere i contatori e l'incapacità di gestire riferimenti ciclici.

# Commenti

È sempre utile documentare i programmi inserendo commenti che indichino quale operazione viene svolta dal programma, quali sono i dati di ingresso e i risultati, qual è il significato delle variabili o di alcune sequenze di istruzioni.

Nei programmi Python i commenti possono essere inseriti in qualsiasi riga, preceduti dal carattere # (“cancellino”).

Tutti i caratteri che seguono il cancellino, fino al termine della riga, saranno considerati commenti e verranno trascurati dall’interprete.



# Indentazione

L'**indentazione** è una tecnica di buona programmazione che ha come principale scopo il miglioramento della leggibilità del codice sorgente. Talvolta è chiamata anche indentatura o *typesetting*.

Consiste sostanzialmente nel cambiare la posizione delle istruzioni nelle righe del codice sorgente, anteponendogli degli spazi bianchi.

```
print("stampa iniziale")
if condizione:
    print("stampa condizionale")
print("stampa finale")
```

Nel caso dell'indentazione tradizionale l'utilità è data dalla maggiore leggibilità del codice, poiché con essa vengono evidenziati i blocchi delle strutture logiche e i vari annidamenti del programma.

# Indentazione

L'indentazione in Python è però definita **significativa**, poiché richiesta dalle specifiche del linguaggio in quanto influente nell'esecuzione del programma.

Nel linguaggio Python l'indentazione sostituisce infatti a tutti gli effetti le parentesi graffe presenti in altri linguaggi di programmazione.

```
print("stampa iniziale")
if condizione:
    print("stampa condizionale")
print("stampa finale")
```

Modificando il caso visto poco fa rimuovendo l'indentazione, ciò che accadrebbe è che verrebbe restituito un errore nell'esecuzione.

# Standard PEP 8 e convenzioni

Tra tutti i documenti ufficiali che definiscono il linguaggio Python ne esiste uno, chiamato PEP 8, che viene costantemente aggiornato e che ha il compito di raccogliere e riportare tutte le principali convenzioni per l'implementazione corretta e facilmente leggibile di un codice scritto in Python. I principali punti trattati nel documento PEP 8 sono:

- *Coerenza*: lo stile con cui viene scritto il codice deve essere mantenuto coerente all'interno di tutto il programma
- *Lunghezza massima delle righe*: non superare gli 80 caratteri per riga di codice
- *Indentazione*: utilizzare 4 spazi per ogni livello di indentazione
- *Righe vuote*: separare le definizioni di funzioni e classi con due righe vuote
- *Spazi nelle espressioni e nelle istruzioni*: utilizzare spazio solo dopo la punteggiatura e prima e dopo gli operatori
- *Commenti*: ne viene consigliato l'utilizzo, in particolare prima della definizione di funzioni e/o classi o subito dopo specifiche istruzioni
- *Convenzioni sui nomi*: i nomi delle funzioni devono essere scritti preferibilmente tutti minuscoli, quelli delle variabili anche con notazione Camel Case (`nomeVariabile`) o Snake Case (`nome_variabile`), mentre quelli della classe in Camel Case con iniziale maiuscola (`MyClass`).

# Tipi di dato ed espressioni

I **tipi di dato** base che possono essere rappresentati ed elaborati dai programmi Python sono:

- numeri interi
- numeri frazionari (*floating point*)
- stringhe di caratteri
- booleani.

Le espressioni Python che producono valori appartenenti a tali tipi di dati, e che possono contenere opportuni operatori, sono le seguenti:

- espressioni aritmetiche
- espressioni di stringhe.

# Espressioni aritmetiche

La più semplice espressione aritmetica è un singolo numero.

I numeri vengono rappresentati nelle istruzioni Python con diverse notazioni, simili a quelle matematiche, espressi in base dieci.

Python distingue tra due tipi di dati numerici:

- numeri interi, codificati nei calcolatori in complemento a due
  - esempi: 12, -9
- numeri frazionari (*floating point*), codificati in virgola mobile, e rappresentati nei programmi:
  - come parte intera e frazionaria, separate da un punto (notazione anglosassone)
    - Esempi: 3.14, -45.2, 1.0
  - in notazione esponenziale,  $m \times b^e$ , con base b pari a dieci ed esponente introdotto dal carattere E oppure e
    - Esempi: 1.99E33 ( $1,99 \times 10^{33}$ ), -42.3e-4 ( $-42,3 \times 10^{-4}$ ), 2E3 ( $2 \times 10^3$ )

# Operatori aritmetici

Espressioni aritmetiche più complesse si ottengono combinando numeri attraverso operatori (addizione, divisione, ecc.), e usando le parentesi tonde per definire la precedenza tra gli operatori.

Gli operatori disponibili nel linguaggio Python sono i seguenti:

<b>Simbolo</b>	<b>Operatore</b>
+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione
//	Divisione (quoziente intero)
%	Modulo (resto di una divisione)
**	Elevamento a potenza

# Espressioni aritmetiche: esempi

Alcuni esempi di istruzioni di assegnazione contenenti espressioni aritmetiche. Il valore per ogni espressione è indicato in grassetto sulla destra.

<code>x = -5</code>	<b>-5</b>
<code>y = 1 + 1</code>	<b>2</b>
<code>z = (1 + 2) * 3</code>	<b>9</b>
<code>circonferenza = 2 * 3.14 * 3</code>	<b>18.84</b>
<code>q1 = 6 / 2</code>	<b>3.0</b>
<code>q2 = 7.5 / 3</code>	<b>2.5</b>
<code>q3 = 5 // 2</code>	<b>2</b>
<code>resto = 9 % 2</code>	<b>1</b>

# Espressioni aritmetiche: risultati

Se entrambi gli operandi di +, - e \* sono interi, il risultato è rappresentato come intero (senza parte frazionaria), altrimenti è rappresentato come numero frazionario.

➤ Esempi:  $1 + 1 \rightarrow 2$ ,  $2 - 3.1 \rightarrow -1.1$ ,  $3.2 * 5 \rightarrow 16.0$

Anche se entrambi gli operandi di / sono interi, il risultato è invece sempre frazionario, come del resto accade se uno o entrambi gli operandi non sono interi.

➤ Esempi:  $6 / 2 \rightarrow 3.0$ ,  $6.0 / 2 \rightarrow 3.0$ ,  $2 / 5 \rightarrow 0.4$ ,  
 $2 / 5.0 \rightarrow 0.4$ ,  $-2 / 3 \rightarrow -0.6666666666666666$

L'operatore // produce sempre il più grande intero non maggiore del quoziente, rappresentato come intero se entrambi gli operandi sono interi, altrimenti come numero frazionario.

➤ Esempi:  $6 // 2 \rightarrow 3$ ,  $6.0 // 2 \rightarrow 3.0$ ,  $2 // 5 \rightarrow 0$ ,  
 $2 // 5.0 \rightarrow 0.0$ ,  $3 // 2 \rightarrow 1$



# Espressioni di stringhe

I programmi Python possono elaborare testi rappresentati come sequenze di caratteri (lettere, numeri, segni di punteggiatura) racchiuse tra apici singoli o doppi, dette stringhe.

Esempi:

- `"Esempio di stringa"`
- `'Esempio di stringa'`
- `'qwerty, 123456'`
- `"` (stringa vuota)

È possibile poi assegnare una stringa a una variabile

Esempi:

- `testo = "Esempio di stringa"`
- `messaggio = "Premere un tasto per continuare."`
- `t = ""`

# Espressioni di stringhe: concatenazione

Il linguaggio Python prevede alcuni operatori anche per il tipo di dato stringa. Uno di questi è l'operatore di concatenazione, che si rappresenta con il simbolo `+` (lo stesso dell'addizione tra numeri) e produce come risultato una nuova stringa ottenuta concatenando due altre stringhe.

Esempi:

- `parola = "mappa" + "mondo"`
  - assegna alla variabile `parola` la stringa `"mappamondo"`
- `testo = "Due" + " " + "parole"`
  - assegna alla variabile `testo` la stringa `"Due parole"`
  - (si noti che la stringa `" "` contiene solo un carattere di spaziatura)

# Espressioni contenenti variabili

È possibile inserire nomi di variabili all'interno delle espressioni, in modo che a quel nome venga poi sostituito in fase di esecuzione del programma il valore ad esso associato.

Esempio:

```
a = 1  
b = a + 1
```

Un'espressione contenente il nome di una variabile alla quale non sia stato ancora assegnato alcun valore è **sintatticamente errata**. Per esempio, assumendo che alla variabile `h` non sia ancora stato assegnato alcun valore, l'istruzione

```
x = h + 1
```

genererà il seguente messaggio di errore:

```
NameError: name 'h' is not defined
```

# Espressioni booleane

Il sistema di Boole, creato dal matematico britannico George Boole, è basato su un approccio binario, in grado di differenziare gli argomenti in base alla presenza (valore pari a «True») o all'assenza (valore pari a «False») di una determinata proprietà.

I valori booleani possono essere prodotti da:

- Le costanti True o False.
- Una variabile a cui sia stato assegnato un valore booleano.
- Una relazione fra due espressioni per mezzo degli operatori relazionali come ==, !=, <, >, <=, >=

# Operatore booleano AND

Se  $x$  e  $y$  sono booleani, possiamo completamente determinare i valori possibili di  $x \text{ and } y$  usando la seguente «tavola di verità»:

<b><math>x</math></b>	<b><math>y</math></b>	<b><math>x \text{ and } y</math></b>
False	False	False
False	True	False
True	False	False
True	True	True

In sintesi:  $x \text{ and } y$  è sempre False a meno che le variabili  $x$  e  $y$  non siano entrambe True

# Operatore booleano OR

Se  $x$  e  $y$  sono booleani, possiamo completamente determinare i valori possibili di  $x \text{ or } y$  usando la seguente «tavola di verità»:

<b><math>x</math></b>	<b><math>y</math></b>	<b><math>x \text{ or } y</math></b>
False	False	False
False	True	True
True	False	True
True	True	True

In sintesi:  $x \text{ or } y$  è sempre True a meno che le variabili  $x$  e  $y$  non siano entrambe False

# Operatore booleano NOT

Se  $x$  è booleano, possiamo completamente determinare i valori possibili di `not x` usando la seguente «tavola di verità»:

<b><math>x</math></b>	<b><code>not x</code></b>
False	True
True	False

In sintesi: `not x` è True se  $x$  è False ed è False se  $x$  è True

# Errori a runtime

Un *exception error* si verifica se si utilizza un operatore con tipi di dato non compatibili.

```
x = 5 + 'abc'  
print('x =', x)
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-4-8c20a3377a59> in <module>()  
----> 1 x = 5 + 'abc'  
      2 print('x =', x)
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```