



Python

Introduzione al linguaggio e primi passi

Argomenti della lezione

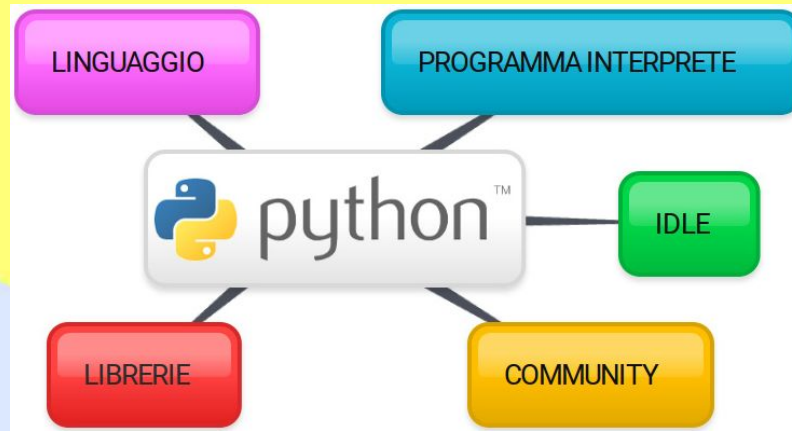
- Cos'è Python
- Installazione dell'ambiente
- IDLE e l'interprete Python
- Variabili e commenti
- Introduzione ai tipi di dato
- Operatori matematici di base
- Le funzioni predefinite
- Esecuzione di semplici applicazioni Python
- Esercitazioni

Cos'è Python

Python è un linguaggio di programmazione avanzato che si caratterizza per essere:

- **di alto livello ed interpretato:** il programmatore non si deve preoccupare di aspetti di “basso livello” legati all'hardware su cui verranno eseguite le applicazioni. L'interprete si occupa di tradurre il codice scritto in Python in linguaggio macchina;
- **multiplatforma:** le applicazioni scritte in Python sono portabili, cioè possono essere eseguite su differenti piattaforme come Windows, MacOS e Linux;

- **open source e gratuito:** chiunque può utilizzarlo per scrivere applicazioni e vedere il codice sorgente, con cui è stato scritto l'interprete;
- **semplice e conciso:** la sintassi del linguaggio è, in generale, facile da imparare ed utilizzare;
- **general purpose:** Python non è legato allo sviluppo di applicazioni specifiche, ma si presta alla realizzazione di progetti di varia natura.



L'ecosistema di Python, oltre al linguaggio, è costituito da:

- un'applicazione che interpreta ed esegue il codice scritto in Python;
- un ambiente di sviluppo (IDLE), per la scrittura ed esecuzione delle nostre applicazioni;
- un insieme di librerie utilizzabili all'interno del nostro applicativo;
- una community che si occupa dello sviluppo e della divulgazione del linguaggio.

Per cosa si utilizza Python

Come anticipato si tratta di un linguaggio *general purpose*, quindi non legato ad uno specifico ambito. Probabilmente uno degli scopi per cui è utilizzato di meno è lo sviluppo di videogames professionali anche se esistono alcune librerie di supporto. Per quasi tutti gli altri ambiti esistono delle librerie per scrivere applicazioni in Python.

Di seguito sono elencati alcuni ambiti con le relative librerie di supporto:

- applicazioni con GUI: Tkinter, PyQt/PySide, PyGtk, wxPython;
- applicazioni WEB: Django, Flask, Web2py
- applicazioni scientifiche: SciPy
- videogiochi: PyGame

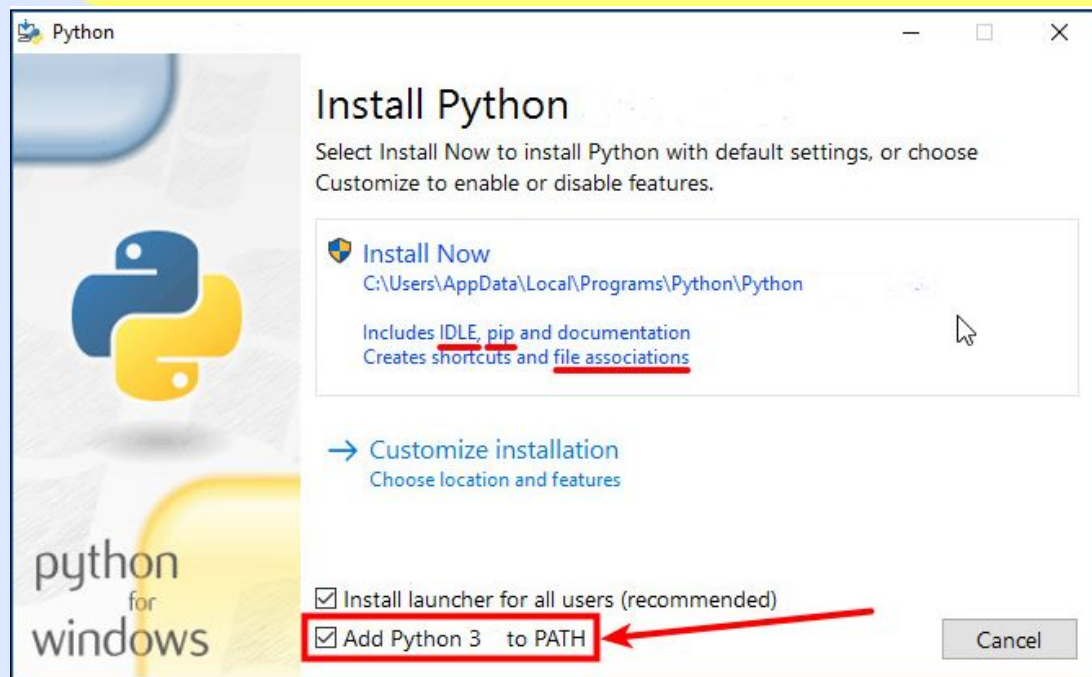
Installazione dell'ambiente

Per poter lavorare con Python in maniera ottimale è necessario installarlo sul proprio computer, esistono degli interpreti online per eseguire il codice scritto ma se ne sconsiglia l'utilizzo se non per testare piccole porzioni di codice.

L'installer dell'ultima versione della release 3 di Python è disponibile, per diverse piattaforme, sul sito ufficiale <https://www.python.org/downloads/>

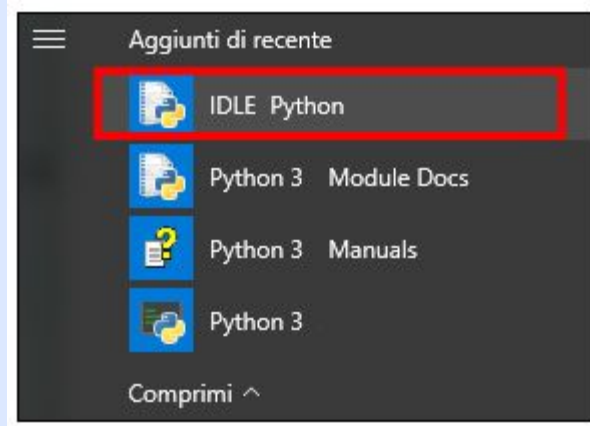


Nel caso di Windows, dopo aver scaricato e lanciato il file di installazione dell'ultima versione. E' importante selezionare la casella "Add Python to PATH" in modo che gli eseguibili di Python siano aggiunti alla variabile d'ambiente PATH.



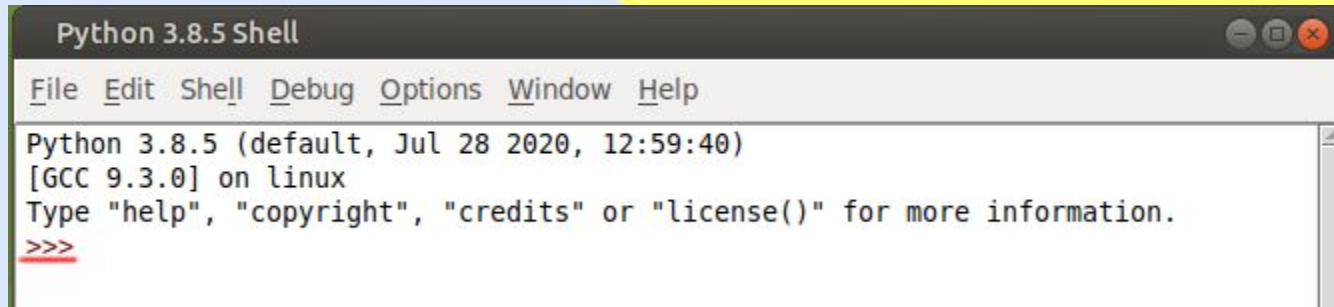
Nei sistemi operativi Linux e Mac OS X è generalmente installata la versione 2 del linguaggio, ma, sempre sul sito ufficiale di Python, si possono trovare i passaggi per installare in maniera semplice Python 3 e IDLE (in Ubuntu trovi Python e IDLE nell'Ubuntu Software Center).

Per iniziare a scrivere codice Python basta lanciare il programma IDLE.



Nella finestra che si apre appare la *Python interactive shell*, l'interfaccia a carattere, o a linea di comando, per interagire con l'interprete Python attraverso comandi testuali.

Scrivendo un comando e premendo invio, l'interprete lo esegue oppure segnala che non è in grado di interpretarlo.

A screenshot of a terminal window titled "Python 3.8.5 Shell". The window has a dark title bar with standard Linux window controls (minimize, maximize, close). Below the title bar is a menu bar with the following items: File, Edit, Shell, Debug, Options, Window, and Help. The main area of the window displays the following text: "Python 3.8.5 (default, Jul 28 2020, 12:59:40)", "[GCC 9.3.0] on linux", and "Type 'help', 'copyright', 'credits' or 'license()' for more information.". At the bottom of the main area, the prompt ">>>" is shown, with the first two characters ">>" underlined in red. The background of the terminal window is white, and the text is in a monospaced font.

```
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

IDLE e l'interprete Python

Se lanciamo il comando `python` da riga di comando, sia in ambiente Windows che Linux, viene mostrato un nuovo prompt caratterizzato da 3 caratteri di maggiore (`>>>`), che da adesso chiameremo **interprete interattivo**.

L'interprete interattivo è in grado di leggere e valutare man mano le espressioni inserite dall'utente, ma è anche possibile eseguire script contenenti sequenze di istruzioni Python, digitando il comando:

```
>>> python nome_script.py
```

IDLE è un ambiente visuale che permette di modificare, eseguire, e fare il debug dei programmi Python.

Variabili e commenti

I commenti sono delle informazioni aggiuntive, completamente ignorate dall'interprete Python, e si utilizzano generalmente per dare informazioni aggiuntive o spiegare lo scopo/funzionamento di un blocco di codice.

In Python è possibile inserire due tipologie di commento:

- **multilinea:** il testo dovrà essere compreso tra tre virgolette consecutive (es. `''' testo commento '''`)
- **su linea singola:** basterà far precedere il testo dal carattere `#`.

La definizione di una variabile in Python non richiede che essa venga definita prima dell'utilizzo né che ne venga specificato il tipo di valore.

L'unica operazione da compiere è l'assegnazione di un valore all'identificatore con l'apposito operatore (=):

```
>>> variabile = 10  
>>> variabile = "Hello Python"  
>>> variabile = [1, 2, 3]
```

Nella definizione del nome di una variabile dobbiamo fare attenzione alle seguenti regole:

- una variabile non può avere lo stesso identificatore di una delle *keywords* del linguaggio;
- Python è *case-sensitive*, ovvero distingue tra maiuscole e minuscole;
- anche se in Python 3 è possibile, nella definizione del nome di una variabile è altamente sconsigliato l'utilizzo di caratteri accentati (es. **nazionalità**="");
- il nome di una variabile deve iniziare con un carattere o underscore (`_`) e può essere seguito unicamente da caratteri, numeri o underscore;

Python permette l'assegnamento multiplo delle variabili, cioè la possibilità di inizializzare più variabili sulla stessa riga di codice:

```
>>> a, b = 10, 20
```

L'assegnazione multipla permette anche di scambiare (swap) i valori di due variabili in maniera molto semplice:

```
>>> a = 3
```

```
>>> b = 8
```

```
>>> a, b = b, a #dopo questa istruzione a vale 8 e b 3
```

Spesso capita di dover incrementare il valore di una variabile. Un primo modo per farlo è quello di utilizzare gli operatori di somma e assegnamento.

Ad esempio, per incrementare di 1 il valore della variabile “a” basta scrivere:

```
>>> a = a + 1 # Aggiunge 1 ad a
```

un modo equivalente è quello di ricorrere all'operatore di incremento, rappresentato dall'operatore di assegnamento composto **+=**, che significa “aggiungi” alla variabile di sinistra il valore dell'operatore di destra:

```
>>>a += 1 # Aggiunge 1 ad a
```

Python mette a disposizione l'operatore di assegnamento composto per tutte le operazioni matematiche di base.

Introduzione ai tipi di dato

Python gestisce sia i tipi di dato comuni alla maggior parte dei linguaggi di programmazione che tipi più flessibili. Di seguito è riportato il set di tipi di dato più comuni in Python:

Tipo di dati	Nome	Descrizione
Intero	int	Intero di dimensione arbitraria
Reale	float	Numero a virgola mobile
Booleano	bool	Per valori veri o falsi
Complesso	complex	Numeri complessi con parte reale e immaginaria
Stringhe	str	Usata per rappresentare testo
Bytes	bytes	Usata per rappresentare bytes

Tipo di dati	Nome	Descrizione
Liste	list	Una sequenza mutabile di oggetti
Tuple	tuple	Una sequenza immutabile di oggetti
Insiemi	set/frozenset	Un'insieme di oggetti unici
Dizionari	dict	Una struttura che associa chiavi a valori

Diversamente dai linguaggi fortemente tipizzati (come Java o C) in Python non deve essere necessariamente indicato il tipo di dato, inoltre anche la modalità di allocazione della memoria è differente. Mentre in Java/C viene allocato uno spazio in memoria atto a contenere il tipo di valore dichiarato, in Python viene prima salvato il valore in memoria e poi gli viene assegnato l'identificatore scelto.

Operatori matematici di base

In Python sono presenti 4 tipologie di dato per definire valori numerici: intero (**int**), razionale (**float**), complesso (**complex**) e booleano (**bool**). Inoltre è possibile definire i valori interi con notazione ottale, binaria ed esadecimale.

```
>>> 7 + 5 * 2
```

```
17
```

```
>>> int(5.3) # Conversione esplicita di un float in int
```

```
5
```

```
>>> 0b11111111 # Intero in notazione binaria
```

```
255
```

```
>>> 0o377 # Intero in notazione ottale
```

```
255
```

```
>>> 0xFF # Intero in notazione esadecimale
```

```
255
```

Le operazioni ed i relativi operatori sui numeri sono le seguenti:

Operatore	Descrizione	Esempi
+	addizione	$15 + 17 = 32$
-	sottrazione	$32 - 15 = 17$
*	moltiplicazione	$5 * 10 = 50$
/	divisione	$9 / 2 = 4.5$
//	divisione intera	$9 // 2 = 4$
%	modulo	$9 \% 2 = 1$
**	elevamento a potenza	$4 ** 3 = 64$

Se all'interno di un'espressione, oltre ad operazioni di somma e sottrazione, sono presenti delle operazioni di moltiplicazione e/o divisione queste hanno, naturalmente, la precedenza. Per cambiare l'ordine di precedenza è possibile utilizzare le parentesi tonde (in generale, nella programmazione, si utilizzano sempre le parentesi tonde anche se sono presenti più livelli).

```
>>> 2 + 4 * 3
```

```
14
```

```
>>> (2 + 4) * 3
```

```
18
```

Python supporta gli operatori di confronto, che restituiscono i valori booleani **True** e **False**

Operatore	Descrizione	Esempi
==	uguale	(10 == 10) = True
!=	diverso	(10 != 10) = False
>	maggiore	(15 > 15) = False
>=	maggiore o uguale	(15 >= 15) = False
<	minore	(3 < 8) = True
<=	minore o uguale	(3 <= 3) = True

Sono inoltre supportati gli operatori booleani **and**, **or** e **not**:

Operatore	Descrizione
and	Ritorna True se entrambi gli operandi sono veri, altrimenti False
or	Ritorna True se almeno uno degli operandi è vero, altrimenti False
not	Ritorna False se l'operando è vero, True se l'operando è falso

In Python ogni oggetto può essere:

- **vero** (numeri diversi da 0, la costante True, o contenitori che contengono almeno un elemento)
- **falso** (ovvero il numero 0, le costanti False e None, contenitori vuoti).

È possibile verificare se un oggetto è vero o falso usando `bool (oggetto)`. Così come avviene in altri linguaggi di programmazione, anche in Python gli operatori **and** e **or** funzionano in modo che, se il primo operando è sufficiente a determinare il risultato, il secondo non viene valutato.

Le funzioni predefinite

Python mette a disposizione un set (oltre 60) di funzioni predefinite che non necessitano dell'import di librerie esterne.

Per la lista completa è possibile visitare il seguente indirizzo:
<https://docs.python.org/3/library/functions.html>

Di seguito una breve introduzione delle principali funzioni Built-in:

- `print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`
Stampa sullo stream di testo identificato da *file* la lista di parametri di input *objects* separati dalla stringa indicata dal parametro *sep* e seguiti dal valore presente in *end*.

- **input([prompt])**

questa funzione viene usata per consentire all'utente di immettere dati da tastiera, che verranno poi utilizzati dal programma. Se l'argomento `prompt` è presente sarà scritto nello standard output. L'input inserito dall'utente viene convertito in una stringa e restituito dalla funzione.

- **int([x])**

converte il valore `x` passato in input in un intero

- **float([x])**

converte il valore `x` passato in input in un numero in virgola mobile

- **eval(expression)**

parserizza e valuta il valore presente nella stringa `expression` come una espressione Python, ad es:

```
>>> x = 1
```

```
>>> eval("x + 1")
```

```
2
```

- **max(iterable), min(iterable)**

restituiscono il massimo ed il minimo valore presente in una lista passata in input, ad es:

```
>>> x = max(3, 2, 7, 8, 4) #x varrà 8
```

```
>>> y = min(3, 2, 7, 8, 4) #y varrà 2
```

- **abs(x)**

restituisce il valore assoluto di x

- **type(object)**

con un solo argomento restituisce il tipo di dato contenuto in object, generalmente è lo stesso valore contenuto in `object.__class__`

```
>>> type(123) # Ritorna il tipo associato al valore 123
```

```
<class "int">
```

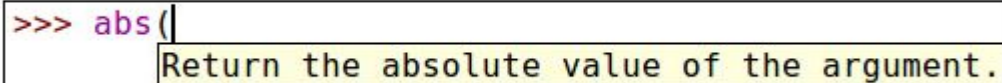
- **round(number[, ndigits])**

restituisce il valore di *number* arrotondato con *ndigits* cifre decimali. Se il parametro *ndigits* non viene passato round restituirà il valore intero più vicino all'input

- **help([object])**

da utilizzare quando non ricordi qual è la sintassi di un'istruzione, di una funzione, di una libreria, di un oggetto o di un valore. Stampa una descrizione più esaustiva di quella data dal call tip di IDLE

In IDLE, sia nell'interprete Python che nella finestra dell'editor, dopo aver digitato il nome di una funzione e la parentesi aperta (, appare un riquadro con un suggerimento (chiamato *call tip*) che aiuta a capire cosa fa la funzione e come può essere usata.

A screenshot of a Python call tip. It shows the text '>>> abs(' in a monospaced font. Below it, a tooltip box contains the text 'Return the absolute value of the argument.' in a standard sans-serif font.

```
>>> abs(  
Return the absolute value of the argument.
```

Esecuzione di semplici applicazioni Python

Problema n°1:

Proviamo a risolvere un semplice problema di economia: “Se depositiamo in banca 100 € a un tasso di interesse annuo del 3% quanti soldi avremo dopo 10 anni?” All’inizio abbiamo 100 €, dopo un anno si aggiungono i 3 € di interesse portando il nostro capitale a 103 €, dopo 2 anni maturano altri 3,09 € di interesse (infatti, $103 \cdot 0,03 = 3,09$) e il capitale diventa 106,09 € e così via.

I dati di ingresso con cui abbiamo a che fare sono:

- capitale iniziale (C_i);
- tasso di interesse annuo (tasso), è la percentuale di interesse che la banca riconosce dopo un anno di deposito del capitale;
- numero di anni (n) di deposito.

A partire da questi valori di input utilizziamo la seguente formula per il calcolo dell'*interesse composto* (si chiama composto perché l'interesse si calcola sia sul capitale iniziale che sull'interesse accumulato fino all'anno precedente):

$$C_f = C_i(1 + \text{tasso})^n$$

Problema n°2:

Oggi ho fatto 4,1 km di corsa in 21 minuti 34 secondi.
Qual è stata la mia velocità oraria e in metri al secondo?

Per calcolare la velocità in metri al secondo (m/s) trasformiamo il tempo impiegato in secondi, salvandolo nella variabile intermedia secondi_totali, mentre per avere i chilometri all'ora (km/h) sfruttiamo questa proporzione:

<km percorsi> : <km all'ora> = <secondi impiegati> : <secondi in un'ora>

Esercitazioni

1. Scrivi il programma **quadrato_e_cubo.py** che definisce la variabile `n`, assegnandole il valore 7, e stampa il suo quadrato e il suo cubo.

L'esecuzione fornisce il seguente output:

```
Il quadrato di 7 è: 49
```

```
Il cubo di 7 è: 343
```

2. Crea il programma **conversione_temperature.py** per la conversione di valori di temperatura tra gradi Celsius (o centigradi) e gradi Fahrenheit e Kelvin. Le formule di equivalenza sono:

$$T(^{\circ}\text{F}) = T(^{\circ}\text{C}) \cdot 1.8 + 32$$

$$T(^{\circ}\text{K}) = T(^{\circ}\text{C}) + 273.15$$

Eseguendo il programma e inserendo il valore 30 otteniamo:

```
Temperatura in °C: 30
```

```
Temperatura in °F: 86.0
```

```
Temperatura in °K: 303.15
```

3. Scrivi il programma **interesse_semplice.py** che, dopo avere definito le variabili con gli stessi dati iniziali del programma **calcolo_interesse_composto.py**, calcola e stampa l'interesse semplice, che non considera l'interesse sull'interesse ma solo sul capitale iniziale ed è definito dalla formula:

```
capitale_finale = capitale_iniziale * (1 + tasso * n_anni)
```

L'esecuzione fornisce il seguente output (in questo caso, essendo nulla la parte frazionaria, la funzione round() ritorna un solo zero anche specificando due o più cifre decimali):

```
Dopo 10 anni il capitale è: 130.0
```


4. Il miglio terrestre è un'unità di misura di lunghezza del sistema britannico che corrisponde a 1.609,344 metri. Scrivi il programma **velocita_in_mph.py** che, dato un tempo di percorrenza e una distanza, calcola la corrispondente velocità in m/s, in km/h e in mph (miles per hour o miglia all'ora).

Il risultato, con i dati del programma visto poco sopra è il seguente:

Velocità: 3.17 m/s

Velocità: 11.41 km/h

Velocità: 7.09 mph