

Python

Parte 1: le basi

Mattia Cozzi
cozzimattia@gmail.com

a.s. 2024/2025

Contenuti

Introduzione

Markdown

Variabili

Stringhe

bool()

Operatori

I/O

try/except

Perché Python

Python nasce nel 1991 ed è un **linguaggio di programmazione ad alto livello orientato agli oggetti**.

La sua sintassi è tra le più semplici e versatili, poiché somiglia molto alla pseudocodifica.

Python, a differenza di altri linguaggi, può essere considerato un linguaggio interpretato e non richiede compilazione: questo significa che il codice, salvato in file con **estensione .py** può essere eseguito direttamente.



IDE per Windows, MacOS e Linux (1)

Per modificare ed eseguire il codice in Python, installiamo **Visual Studio Code**, stando attenti a spuntare, durante l'installazione, la casella per aprire una cartella con VS Code.



Visual Studio Code

<https://code.visualstudio.com/>

Aggiungiamo anche alcune estensioni, che andranno a costituire il nostro IDE (Integrated Development Environment).

IDE per Windows, MacOS e Linux (2)

Installiamo, dal menu a sinistra con i quattro quadratini, le seguenti estensioni:

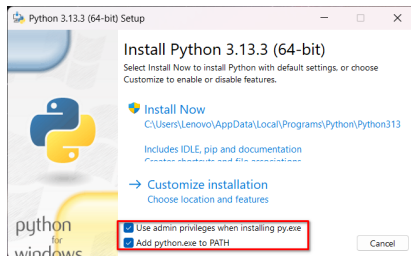
- Italian Language Pack for Visual Studio Code;
- Code Runner (per eseguire agevolmente il codice);
- Code Spell Checker;
- Italian - Code Spell Checker (ricordiamoci di abilitare la lingua italiana dopo averlo installato);
- Python;
- Pinned Projects (per avere a portata di mano le cartelle che usiamo più spesso, come quella per gli esercizi di Python).

Installazione su Windows (1)

Python può essere scaricato dal sito ufficiale:

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

Ricordiamoci, durante l'installazione, di **spuntare la casella per aggiungere Python a PATH**, in modo che Windows sappia dove trovare l'eseguibile di Python.



Installazione su Windows (2)

Per verificare che Python sia installato, apriamo un terminale con [clic destro sul logo Windows](#) della barra e selezioniamo “Terminale”.

Diamo il comando:

```
python --version
```

Se tutto è corretto vedremo un output del genere:

```
PS C:\Users\Lenovo> python --version
Python 3.13.3
PS C:\Users\Lenovo> |
```

Installazione su MacOS

MacOS ha già Python installato, ma dobbiamo controllare quale versione abbiamo (la comunità usa in maniera totale Python versione 3).

Apriamo il terminale e digitiamo semplicemente

```
python
```

Se dobbiamo aggiornare a Python 3, lo scarichiamo da:

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

e seguiamo questa guida:

<https://docs.python.org/3/using/mac.html>

Installazione su Linux

Python è preinstallato in tutte le distribuzioni Linux moderne. L'eseguibile è di solito chiamato “python3”.

L'unica cosa da fare è impostare Code Runner per eseguire correttamente un file Python. Apriamo le impostazioni di Code Runner e cerchiamo l'opzione “Executor Map” e poi “Modifica in settings.json”.

La riga relativa a Python dovrà essere:

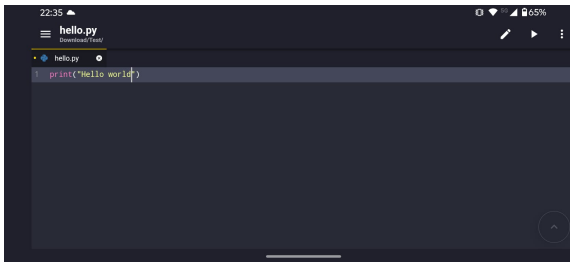
```
"python": "python3 -u",
```

Ricordiamoci infine di salvare il file di impostazioni con CTRL+S.

Installazione su Android

Su Android, possiamo scrivere codice in Python utilizzando l'applicazione [Acode](#), reperibile dal Play Store.

Una volta che la abbiamo installata, installiamo anche il plugin chiamato semplicemente [Python](#), che ci permetterà di eseguire il codice creato.



Editor online

Se non riusciamo a installare Python sulla nostra macchina, possiamo sempre ricorrere ad un editor online, come ad esempio:

<https://www.online-python.com/>

Ricordiamoci di installare un adblocker sul nostro browser per non essere disturbati dalle pubblicità mentre creiamo il nostro codice.

Primo programma

Creiamo prima di tutto una cartella che contenga i nostri esercizi (apriremo sempre questa cartella con VS Code o Acode), e all'interno della cartella un file chiamato `01-helloworld.py`.

All'interno del file scriviamo la seguente riga di codice:

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

Proviamo ora ad eseguire il codice con l'estensione `Code Runner` usando la scorciatoia CTRL+ALT+N e, nel terminale di VS Code, vediamo eseguire il nostro codice.

```
[Running] python3 -u "/home/mattia/Documents/Lavoro/Fondazione Biotecnologie/esempi/01-helloworld.py"
Hello world!

[Done] exited with code=0 in 0.108 seconds
```

Commenti

I commenti, che permettono gli umani di comprendere meglio il codice e vengono ignorati dal computer in fase di esecuzione, iniziano con `#` e finiscono a fine riga.

Ecco un esempio di codice commentato:

```
1 print("Hello world!") #così mostro una riga a schermo
```

I commenti sono utili per prendere appunti sul codice che stiamo scrivendo (anche durante le lezioni).

Su VS Code, la scorciatoia CTRL+SHIFT+7 ci permette di commentare una riga.

Indentazione

Python, a differenza di altri linguaggi che usano le parentesi, utilizza l'**indentazione** per indicare un blocco di codice.

L'indentazione è lo spostamento a destra di alcuni spazi (di solito quattro). L'indentazione viene ottenuta velocemente con il tasto TAB sulla tastiera (le due frecce sopra al tasto per le maiuscole).

```
1  if 5>2:
2      print("Cinque è maggiore di due")
3      #il codice della riga precedente è INDENTATO:
4      #se non indentassi il codice, Python non riuscirebbe
5      #a interpretare correttamente le istruzioni fornite
```

L'indentazione sarà fondamentale più avanti quando studieremo le **strutture** che possiamo utilizzare in Python.

Il linguaggio Markdown

Il linguaggio Markdown serve ad ottenere **testo formattato** a partire da file di **testo semplice**, come quelli che possiamo modificare con il blocco note o con il nostro editor di codice.

È molto comodo per **prendere appunti all'interno del nostro IDE**, riportando anche esempi di codice.

Il file di Markdown sono file di testo salvati **con estensione .md**

Titoli

Scrivendo il codice a sinistra, lo visualizzeremo poi come mostrato a destra.

```
1 # Titolo principale
2
3 ## Sottotitolo
4
5 ### Sotto-sottotitolo
6
7 #### Sotto-sotto-sottotitolo
8
9 Questo invece è testo semplice.
```

Titolo principale

Sottotitolo

Sotto-sottotitolo

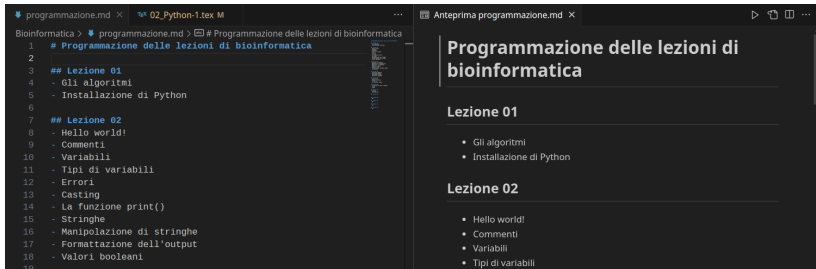
Sotto-sotto-sottotitolo

Questo invece è testo semplice.

Visualizzazione

Per visualizzare la versione formattata (chiamata “anteprima”), su VS Code usiamo la scorciatoia CTRL+ALT+V.

Molto comodo è [dividere la finestra di VS Code in pannelli](#) trascinando la scheda dell’anteprima fuori dall’elenco delle altre schede.



Formattazione del testo

```
1  *testo in corsivo*, oppure _testo in corsivo_
2
3  **testo in grassetto**, oppure __testo in grassetto__
4
5  ~~testo sbarrato~~
6
7  porzione `di codice informatico`
```

testo in corsivo, oppure testo in corsivo

testo in grassetto, oppure testo in grassetto

testo sbarrato

porzione *`di codice informatico`*

Per scrivere il carattere ~, usiamo la scorciatoia ALT GR+Ì.

Interruzioni

Il testo rimane su una sola riga anche se andiamo a capo. Per forzare una nuova linea, usare un **doppio spazio a fine riga**.

Per un nuovo paragrafo, lasciamo una alertriga vuota.

```
1 Il lonfo non vaterca né gluisce__
2 e molto raramente barigatta__
3 ma quando soffia il bego a bisce bisce__
4 sdilenca un poco, e gnagio s'archipatta.__
5
6 È frusco il lonfo! È pieno di lupigna__
```

```
Il lonfo non vaterca né gluisce
e molto raramente barigatta
ma quando soffia il bego a bisce bisce
sdilenca un poco, e gnagio s'archipatta.

È frusco il lonfo! È pieno di lupigna
```

Elenchi e sottoelenchi

Per ottenere un elenco dentro un altro elenco, usiamo l'**indentazione** (tasto TAB sulla tastiera).

```
1 - primo
2 - secondo
3   - secondo-1
4   - secondo-2
5 - terzo
6
7 1. primo
8 1. secondo
```

```
• primo
• secondo
  ◦ secondo-1
  ◦ secondo-2
• terzo

1. primo
2. secondo
```

Link e immagini

```
1 [Il mio link](https://www.il-mio-link.com)
2
3 ![Descrizione immagine](img/python.png)
```

Il mio link



Tabelle

```
1 | elemento | simbolo | numero atomico | massa atomica |
2 | :-----: | :-----: | :-----: | :-----: |
3 | ossigeno | O        | 8          | 16.00         |
4 | ferro    | Fe       | 26         | 55.85         |
5 | alluminio| Al       | 13         | 26.98         |
6 | selenio  | Se       | 34         | 78.97         |
7 | carbonio | C        | 6          | 12.01         |
```

elemento	simbolo	numero atomico	massa atomica
ossigeno	O	8	16.00
ferro	Fe	26	55.85
alluminio	Al	13	26.98
selenio	Se	34	78.97
carbonio	C	6	12.01

Realizzare diagrammi di flusso con Mermaid

Con una estensione di Markdown, chiamata [Mermaid](#), possiamo facilmente [realizzare diagrammi scrivendo codice](#).

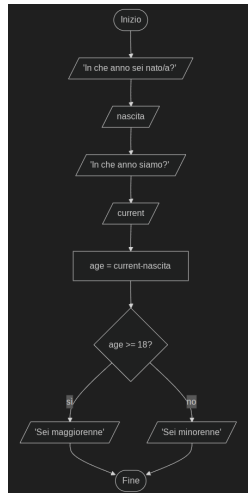
Cominciamo installando l'estensione [Markdown Preview Mermaid Support](#) e proviamo con il seguente codice:

```
1  ```mermaid
2  graph TB
3      A([Inizio]) --> B[/stampa: 'Hello world!'/]
4      B --> C([Fine])
5  ```
```

Per scrivere il carattere ` , usiamo la scorciatoia ALT GR+'. TB indica che il nostro grafico sarà verticale (*top to bottom*). Per un grafico orizzontale, usiamo LR (*left to right*).

Esempio

```
1  ```mermaid
2  graph TB
3      A([Inizio]) --> B[/In che anno sei nato/a?/]
4      B --> C[/nascita/]
5      C --> D[/In che anno siamo?/]
6      D --> E[/current/]
7      E --> F[age = current-nascita]
8      F --> G{age >= 18?}
9      G -->|si| H[/Sei maggiorenne'/]
10     G -->|no| I[/Sei minorenne'/]
11     H --> J([Fine]);
12     I --> J
13  ```
```

[► Mermaid cheat sheet](#)

Variabili e costanti

Torniamo ora a Python.

Una **variabile** è uno spazio in memoria in cui salvare un certo dato. Il valore della variabile può cambiare anche molte volte durante l'esecuzione dell'algoritmo. Ogni variabile deve avere un **nome**.

Una **costante** è un dato che non cambia mai durante l'esecuzione dell'algoritmo.

Attenzione! Python è **case sensitive**, quindi distingue tra maiuscole e minuscole.

Creazione di una variabile

La sintassi per la creazione di una variabile è molto semplice:

```
1 #creo la variabile chiamata "numerino" e le assegno il valore 5
2 #il simbolo di UGUALE è un operatore di assegnazione
3 numerino = 5
4 #stampo il valore della variabile
5 print(numerino)
```

In Python, **non possiamo creare variabili vuote**: alla creazione della variabile, deve esserle assegnato subito un valore.

È bene assegnare alle variabili dei nomi che ci aiutino a ricordare quale dato rappresentano.

Nei nomi di variabili, non sono ammessi spazi o trattini. I nomi delle variabili non devono iniziare con un numero.

Creazione di variabili multiple e utilizzo

Possiamo usare una sintassi rapida per creare in una sola riga più di una variabile:

```
1 x, y, z = 10, 20, 30
```

Per creare più variabili con lo stesso valore:

```
1 x = y = z = 10
```

Una volta creata una variabile, possiamo utilizzarla nel nostro algoritmo:

```
1 x = 10
2 y = 20
3 z = x + y
4 print(z)
```

Errori

Quando eseguiamo il nostro algoritmo in Python, capiterà spesso che il codice contenga qualche errore e non venga eseguito.

È utile [leggere il messaggio di errore](#) nel terminale, perché ci indica quale errore è avvenuto e dove. Ad esempio il codice:

```
1 x = 50
2 print(y)
```

restituisce il seguente errore:

```
File "../esempi/test.py", line 2, in <module>
    print(y)
      ^
NameError: name 'y' is not defined
```

Tipi di dati (1)

In altri linguaggi di programmazione, quando creo una variabile devo anche assegnarle un **tipo**: numero intero, numero con la virgola, carattere, stringa di testo, eccetera.

In Python, **il tipo viene assegnato automaticamente**. Per scoprirlo, usiamo la funzione **type()**:

```
1 x = 10
2 y = 3.14
3 print(type(x))
4 print(type(y))
```

Tipi di dati (2)

Tipo	Descrizione
<code>int</code>	numero intero
<code>float</code>	numero con virgola
<code>bool</code>	valore booleano (True/False)
<code>str</code>	stringa di testo
<code>list</code>	lista di valori
<code>tuple</code>	elenco di valori
<code>set</code>	insieme di valori
<code>range</code>	range di valori

Vedremo più avanti le caratteristiche dei tipi in fondo alla tabella, cioè i tipi relativi alle collezioni di dati.

Casting di variabili

Il casting è l'operazione con cui **modifichiamo il tipo di dato di una variabile**.

Ad esempio, il seguente codice mi darà un errore:

```
1 x = "10"  
2 y = 5  
3 print(x + y)
```

Il motivo dell'errore è che ho provato ad eseguire un'operazione tra una stringa ed un numero intero. Se eseguo il casting posso eseguire l'operazione:

```
1 x = "10"  
2 y = 5  
3 x = int(x) #trasforma x in un intero  
4 print(x + y)
```

Esercizi

1. Crea due variabili float, scegli per esse un nome e un valore e stampa la stringa "La somma di X e Y è Z". Ricorda di eseguire il casting.
2. Crea due variabili intere, scegli per esse un nome e un valore e stampa la stringa "La differenza di X e Y è Z".
3. Crea una variabile intera e una float, scegli per esse un valore e stampa la stringa "La somma di X e Y è Z".

Creazione e stampa di una stringa

Per creare una stringa, usiamo una di queste due sintassi:

```
1 x = "Chimica"      #possiamo scegliere quali
2 y = 'Matematica'   #virgolette usare (semplici o doppie)
```

Per stampare a schermo una stringa:

```
1 print("Informatica") #scrivo direttamente la stringa
2 print(x)              #scrivo il nome della variabile
```

Stringhe multilinea

Per creare una stringa che si estende su più linee:

```
1 poesia = """Il lonfo non vaterca né gluisce
2 e molto raramente barigatta,
3 ma quando soffia il bego a bisce bisce
4 sdilenca un poco, e gnagio s'archipatta."""
5 #uso tre virgolette (anche qui, semplici o doppie)
```

Matrici

Nel linguaggio informatico gli **insiemi strutturati di dati** si chiamano **array** (letteralmente “matrice”).

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,m} \\ a_{2,1} & a_{2,2} & \dots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,m} \end{pmatrix}$$

La matrice rappresentata ha **n righe** e **m colonne**.

Si indicano sempre **prima le righe e poi le colonne**. Una matrice (5, 3) ha 5 righe e 3 colonne.

Attenzione! I dati di una matrice devono essere **omogenei**, cioè dello **stesso tipo**.

Cosa sono i vettori in informatica

Quando una matrice ha una sola riga (è cioè monodimensionale) viene chiamata **vettore**.

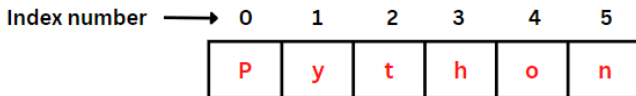
arr	4	7	9	4	-5	8	6	9
	0	1	2	3	4	5	6	7

Nota che gli indici iniziano sempre da zero. Un vettore di dimensione n ha quindi gli indici da 0 a $n - 1$.

In Python, le stringhe sono trattate come **array di caratteri**.

Riferimento ad un elemento di un vettore

```
language = "Python"
```



Nella figura precedente la variabile stringa si chiama `language`.

Per indicare l'elemento di indice 3 (cioè il quarto elemento, la "h") usiamo la sintassi:

```
language[3]
```

Lunghezza di una stringa

La funzione `len()` restituisce la lunghezza di una stringa:

```
1 animale = "giraffa"
2 print(len(animale))
3 #ottengo a schermo il valore 7, quindi
4 #gli indici della stringa andranno da 0 a 6
```

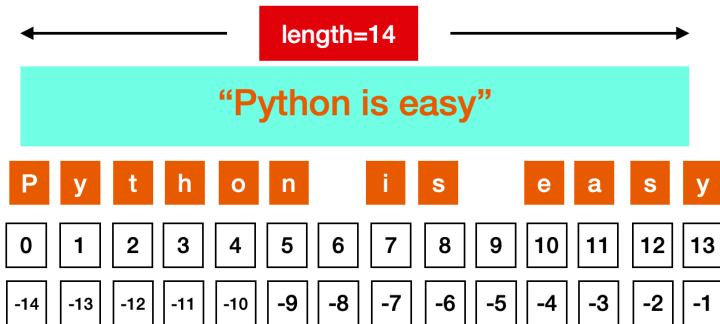
Utilizzare parti di una stringa

Mostriamo con degli esempi come far riferimento a porzioni di una stringa:

```
1 parola = "praticamente"  
2 #la stringa ha 12 lettere  
3 #quindi indici da 0 a 11  
4  
5 print(parola[:3])  
6 #stampa i caratteri con indice da 0 a 3 (escluso)  
7  
8 print(parola[7:])  
9 #stampa i caratteri con indice da 7 fino alla fine  
10  
11 print(parola[5:8])  
12 #stampa i caratteri con indice da 5 a 8 (escluso)
```

Indici negativi

Possiamo far riferimento agli elementi di una stringa **a partire dal fondo** utilizzando degli indici negativi, come spiegato in figura:



Modificare una stringa

I **metodi**, in programmazione a oggetti, sono le **attività** che possiamo eseguire con un certo oggetto. Vengono indicati con un punto dopo il nome dell'oggetto.

Alcuni metodi per le stringhe sono:

```
1 parola = "Fontanile "  
2 print(parola.upper())  
3 #stampa la stringa in maiuscolo  
4  
5 print(parola.lower())  
6 #stampa la stringa in minuscolo  
7  
8 print(parola.strip())  
9 #elimina gli spazi bianchi a inizio e fine stringa  
10  
11 print(parola.replace("a", "u"))  
12 #sostituisce tutte le "a" con delle "u"
```

Separare una stringa

Mostriamo questa funzione con un esempio:

```
1 string = "Il mio gatto si chiama Spooky"
2
3 string2 = string.split(" ")
4 #separa string ogni volta che vede uno spazio vuoto
5 #ovviamente al posto dello spazio posso mettere
6 #qualsiasi carattere mi serva
7
8 print(string2)
9 #stampa la stringa 'smontata'
```

► [Elenco completo dei metodi per le stringhe](#)

Concatenazione di stringhe e ricerca

Per unire tra loro stringhe:

```
1 x = "Il mio gatto si chiama "  
2 y = "Spooky"  
3 z = x + y  
4 print(z)  
5 #attenzione: posso concatenare solo stringhe con altre  
6 #stringhe, quindi all'occorrenza posso usare il casting!
```

► Concatenate strings

Per cercare quante volte una stringa compare in un'altra:

```
1 x = "Il mio gatto si chiama Spooky"  
2 print(x.count("o")) #quante volte compare "o"?
```

Il metodo format()

Per inserire valori all'interno di una stringa evitando casting complessi, possiamo ricorrere al metodo `format()`.

```
1 presentazione = "Ciao, mi chiamo {}, sono nato il {} gennaio,  
  ↳ sono alto {} m e peso {} kg"  
2 #le parentesi graffe fungono da 'segnaposto' per  
3 #i punti in cui voglio inserire i miei dati  
4 nome = "Carlo"  
5 data = 12  
6 alt = 1.75  
7 peso = 72  
8 print(presentazione.format(nome, data, alt, peso))  
9 #attenzione all'ordine con cui sono indicati i valori  
10 #da inserire dove abbiamo messo i nostri 'segnaposto'
```

Stringhe formattate

A partire da Python 3.6, esiste un modo più rapido e semplice per costruire stringhe che contengono il valore di certe variabili.

Queste stringhe formattate vengono chiamate **f-strings** e usano la seguente sintassi:

```
1 nome = "Carlo"
2 data = 12
3 alt = 1.75
4 peso = 72
5 #attenzione alla f davanti alla stringa!
6 presentazione = f"Ciao, mi chiamo {nome}, sono nato il {data}
   ↳ gennaio, sono alto {alt} m e peso {peso} kg"
7 #nelle graffe inserisco le variabili che voglio utilizzare
8 print(presentazione)
```

Formattazione avanzata

A volte avremo bisogno di esprimere il valore delle variabili numeriche in modo differente. Di seguito alcuni esempi.

```
1 age = 37
2 pi = 3.1415926535
3
4 #il primo numero dopo i due punti indica lo spazio
5 #minimo da utilizzare per scrivere il numero
6 string = f"Io ho {age:3} anni"
7
8 #il secondo numero dopo i due punti indica lo spazio
9 #minimo da utilizzare per scrivere il numero
10 #e, se vuoto, inserisce degli zeri
11 string2 = f"Io ho {age:04} anni"
12
13 #il numero dopo il punto indica quanti
14 #decimali (figures) voglio usare
15 string3 = f"Pi greco vale {pi:.4f}"
```

Prova a riscrivere questo codice e a stampare le tre stringhe.

Sequenze di escape

Se volessimo inserire un apostrofo in una stringa, Python lo interpreterebbe come la fine della stringa.

```
1 string = 'Mi passi l' acqua?'
```

Per ovviare a problemi di questo tipo, usiamo delle [sequenze di escape](#) (che iniziano tutte con un `\`), che ci permettono di digitare caratteri speciali nelle stringhe.

```
1 string= 'Mi passi l\'acqua?'
```

► [Elenco completo caratteri di escape](#)

Esercizi

4. Creare una stringa che dica "Hello world!", sostituire la parola "world" con "Python" e stampare la stringa modificata.
5. Creare una stringa casuale e stamparne solo i primi tre caratteri.
6. Creare una stringa casuale e stamparne gli ultimi tre caratteri usando la funzione `len()`.
7. Creare una stringa con una parola con un numero pari di lettere e stampare le due lettere centrali.

Esercizi

8. Con il metodo `format()` oppure usando le f-strings, prepara una presentazione di te stesso/a che inserisca in una stringa almeno cinque valori salvati in altrettante variabili.
9. Riscrivere una poesia di almeno 5 versi in una unica variabile stringa e stamparla a schermo.
10. Contare tutte le volte che la lettera "a" compare nei testi dei due esercizi precedenti.

Valori booleani

I valori booleani prendono il nome da George Boole, un logico e matematico del XIX secolo che creò un'algebra basata su **valori binari**.

I valori booleani sono quindi **True** e **False** (attenzione alla maiuscola in Python), oppure 1 e 0, ecc.



Ogni linguaggio di programmazione prevede la presenza di variabili di tipo booleano, per poter valutare se una certa condizione (semplice o complessa) risulta vera oppure falsa.

```
1 print(5 < 3)
2 #questa proposizione è indicata ovviamente come False
```

La funzione bool()

La funzione `bool()` restituisce **False** in tutti i casi indicati di seguito:

```
1 bool(False)
2 #variabile falsa
3 bool(None)
4 #variabile vuota
5 bool(0)
6 #variabile nulla
7 bool("")
8 #stringa vuota
9 #lo stesso discorso vale per liste, tuple e insiemi vuoti
```

In tutti gli altri casi ritorna **True**. Utile ad esempio per valutare se una certa lista è vuota oppure contiene qualcosa.

Operatori aritmetici

Operazioni aritmetiche di base, da eseguire su variabili numeriche:

Operatore	Significato	Operatore	Significato
+	somma	**	potenza
-	sottrazione	%	modulo
*	prodotto	//	<i>floor division</i>
/	divisione		

L'operatore modulo fornisce il **resto** di una divisione. Ad esempio $11 \% 3 = 2$, mentre $20 \% 4 = 0$.

La *floor division* restituisce il risultato di una divisione arrotondato all'**intero inferiore**, quindi $20 // 3 = 6$.

Operatori di assegnazione

Questi operatori servono ad assegnare (o ri-assegnare) un valore ad una variabile.

Operatore	Esempio	Corrisponde a
=	<code>x = 5</code>	<code>x = 5</code>
+=	<code>x += 5</code>	<code>x = x + 5</code>
-=	<code>x -= 5</code>	<code>x = x - 5</code>
*=	<code>x *= 5</code>	<code>x = x * 5</code>

Operatori di confronto

Questi operatori servono a confrontare due valori e ritornano un **valore booleano** (**True** o **False**).

Operatore	Esempio	Significato
<code>==</code>	<code>x == y</code>	uguale a
<code>!=</code>	<code>x != y</code>	diverso da
<code>></code> , <code><</code>	<code>x < y</code>	maggiore/minore di
<code>>=</code> , <code><=</code>	<code>x >= y</code>	maggiore/minore o uguale a

Operatori logici

Questi operatori ritornano un valore booleano a seconda del valore di verità delle proposizioni che collegano.

Operatore	Esempio	Descrizione
and	<code>x < 5 and x > 0</code>	ritorna True se entrambi i congiunti sono veri
or	<code>x < 5 or x > 10</code>	ritorna True se almeno uno dei disgiunti è vero
not	<code>not(x < 5 and x > 0)</code>	ritorna True se applicato ad un False e viceversa

Altre funzioni matematiche

Ci sono altre funzioni matematiche che possono risultare utili:

```
1 x = min(7, 12, 4, 30)
2 #assegna ad x il valore minimo della lista
3
4 y = max(7, 12, 4, 30)
5 #assegna ad x il valore massimo della lista
6
7 z = abs(-9)
8 #assegna a z il valore assoluto di -9
9
10 w = pow(3, 5)
11 #assegna a w il risultato di 3 alla quinta
```


Esercizi

11. Partendo dalla variabile "numero" con valore a piacere, creare una stringa che dica "Il quadrato di X è Y".
12. Creare le variabili intere "numero1" e "numero2" e calcolare il resto della divisione tra numero1 e numero2.
13. Creare le variabili intere "numero1" e "numero2" e mostrare a schermo la variabile booleana che valuta se il primo è maggiore del secondo.
14. Creare le variabili intere "numero1" e "numero2" (maggiore dell'altro) e mostrare a schermo la variabile booleana che valuta se il primo è multiplo del secondo.

Input di dati

Per richiedere all'utente di inserire dei dati, possiamo usare la funzione `input()`.

```
1 username = input("Inserisci il tuo nome utente:")
2 #la variabile assume il valore inserito dall'utente
3 print("Il tuo nome utente è " + username)
```

L'input dell'utente viene sempre **formattato come stringa**.

Se dobbiamo utilizzarlo come numero o altro, dobbiamo ricorrere al **casting**.

Esempio di input e casting

```
1  nascita = (input("In che anno sei nato/a? "))
2  #chiedo l'inserimento di qualcosa
3
4  nascita = int(nascita)
5  #trasformo l'inserimento in un numero
6
7  corrente = int(input("In che anno siamo? "))
8  #eseguo le due operazioni precedenti in una sola riga
9
10 age = corrente - nascita
11 #calcolo dell'età
12
13 risposta = f"Hai {age} anni"
14
15 print(risposta)
```

Output

L'output di valori e stringhe si ottiene con la funzione `print()`. Ricordiamoci che possiamo concatenare stringhe con:

```
1 messaggio = "Il mio gatto si chiama"  
2 nome = "Spooky"  
3 print(messaggio + " " + nome)
```

oppure con:

```
1 nome = "Spooky"  
2 animale = "gatto"  
3 presentazione = f"Il mio {animale} si chiama {nome}"  
4 print(presentazione)
```

Formattazione avanzata output

Usando le stringhe formattate posso fornire anche alcune [specifiche di formattazione avanzata](#) ai miei output, ad esempio usando la notazione scientifica.

```
1 txt1 = f"Il prezzo è {45:.2f} €."
2 print(txt1)
3 #l'opzione .2f mi dice quante cifre decimali (figures)
4 #deve avere il mio numero
5
6 txt2 = f"La velocità della luce è {300000000:.2E} m/s"
7 print(txt2)
8 #notazione scientifica con due decimali
```

Esercizi

15. Scrivere un programma che chieda un input numerico all'utente e ne calcoli il quadrato, mostrandolo a schermo.
16. Utilizzare la formattazione delle stringhe per ottenere "Il numero binario di 42 è 0b101010". Per il binario utilizzare la funzione `bin(numero)`.
17. Utilizzare la formattazione delle stringhe per chiedere all'utente un numero intero e mostrare "Il numero binario di X è Y".
18. Scrivere un algoritmo che contenga la ricetta per una torta per quattro persone. L'algoritmo deve poi chiedere all'utente per quante persone deve calcolare gli ingredienti della torta e deve poi mostrare a schermo gli ingredienti da utilizzare.

Esercizi

19. Scrivere un algoritmo che chieda all'utente le masse di due corpi in kg, la loro distanza in metri e calcoli l'intensità della forza di attrazione gravitazionale tra di essi. Il risultato deve essere mostrato in notazione scientifica con tre decimali e deve essere presentato con chiarezza da una stringa appropriatamente costruita.

La formula per l'attrazione gravitazionale tra due corpi è:

$$F = G \cdot \frac{m_1 \cdot m_2}{r^2}$$

con $G = 6.673E - 11$.

Gestire gli errori

Può (ovviamente) capitare che il nostro codice contenga un errore.

La funzione **try** ci permette di eseguire un blocco di codice in **modalità test**, in modo che il programma non si interrompa se si verifica un errore.

La funzione **except** ci permette di raccogliere l'errore che potrebbe essere generato dal blocco che abbiamo eseguito in modalità test.

La funzione **finally** ci permette di eseguire un altro blocco di codice una volta eseguite le due funzioni precedenti.

Esempio

Scriviamo un codice che genera un errore (si vuole stampare una variabile che non è mai stata creata) all'interno di un **try**:

```
1 try: #attenzione ai due punti e all'indentazione
2     print(x)
3 except:
4     print("Si è verificato un errore!")
5     #se non voglio fare nulla, scrivo "pass" nel mio except
6 finally: #non obbligatorio
7     print("Controllo eseguito")
8     #questo codice, se presente, viene eseguito sempre
9 print("Il programma è giunto al termine")
```

In questo modo il programma che contiene un errore lo “gestisce” e riesce a terminare.

“Creare” un errore

Possiamo fare in modo che Python, pur non avendo formalmente degli errori, ne “sollevi” uno al verificarsi di una certa condizione.

Ad esempio:

```
1 x = -1
2 if x < 0:
3     raise Exception("Il numero deve essere maggiore o uguale 0")
4 print("Programma terminato")
5 #poiché la riga 3 solleva un errore
6 #la riga 4 non viene mai eseguita
```