

python

introduction

python is

Python è un linguaggio di programmazione **high-level**, **interpreted**, **general-purpose**, **multi-paradigma** e **open-source**, *python* è **dinamicamente tipizzato**.

...cosa significa?

high-level → linguaggio con **forte astrazione** dai dettagli della macchina;

interpreted → un **interprete** esegue i comandi/programmi direttamente senza una compilazione;

general-purpose → tipico dei linguaggi di programmazione progettati per essere utilizzati in un'ampia gamma di domini di applicazione, ovvero non hanno uno scopo specifico;

multi-paradigma → supporta diversi paradigmi di programmazione, nel paradigma imperativo supporta sia il paradigma **procedurale** che quello **orientato agli oggetti**. Supporta anche il paradigma **funzionale**.

open-source → il codice è pubblico e modificabile da chiunque;

dinamicamente tipizzato → la "tipizzazione" avviene a runtime.



history

Guido van Rossum (BDFL - Benevolent dictator for life) è il creatore di Python.

Python nasce come il successore del linguaggio ABC, un linguaggio imperativo general-purpose della metà degli anni 80'.

Il nome è ispirato al "Monty Python's Flying Circus" [van Rossum era un appassionato]

più "vecchio" di Java → Java nasce nel 1995;

versione 2 → rilasciato nel 2000 (EOL prevista nel 2015 estesa al 2020)

versione 3 → rilasciata nel 2008 (non retrocompatibile)

filosofia → riassunta nella *The Zen of Python* [`import this`]



interpreti

Cpython

CPython è l'implementazione di riferimento di Python. Scritto in C e Python, CPython è l'implementazione predefinita e più diffusa del linguaggio.

Jython

Jython, successore di JPython, è un'implementazione del linguaggio di programmazione Python scritto in Java. I programmi Jython possono importare e usare qualsiasi classe Java.

Pypy

PyPy è un'implementazione alternativa di Python a CPython. PyPy viene spesso eseguito più velocemente di CPython, perché PyPy è un compilatore just-in-time, mentre CPython è un interprete.

Stackless Python

Stackless Python è un interprete di Python, così chiamato perché evita di dipendere dallo stack di chiamate C per il proprio stack. La caratteristica più importante di Stackless sono i microthread, che evitano gran parte del sovraccarico associato ai soliti thread del sistema operativo.



interpreti

MicroPython

Implementazione di Python 3 ad hoc per microcontrollori (schede da maker tipicamente inferiori prestazionalmente ad una raspberry). Non ha tutta la py3 standard library ma aggiunge alcuni moduli di controllo hardware.

Anaconda

Di gran successo soprattutto nel campo della data science. Aggiunge alla standard library un numero enorme di moduli aggiuntivi preconfigurati (tra cui i famosi numpy e pandas)

Ipython

Ipython potenzia CPython e lo rende più adatto all'utilizzo interattivo. Forte interazione con la shell di comando. Alla base del progetto Ipython notebook oggi evoluto in Jupyter Notebook.



il linguaggio

struttura lessicale

La fine di una linea delimita la fine di uno statement. Se lo statement è più lungo di 80 caratteri utilizzo backslash (\) per continuare in una nuova linea. L'apertura di parentesi ([{) determinano la fine dello statement solo alla loro chiusura (eventuali nuove linee vengono considerate un'unica linea logica senza l'utilizzo del backslash).

Python usa l'indentazione per esprimere la struttura dei blocchi di un programma. Quindi un blocco è un insieme di linee contigue tutte indentate della stessa quantità.

Nell'indentazione è bene non mischiare spazi e tab. **È raccomandato l'utilizzo di 4 spazi**, gli IDE di solito traducono tab in 4 spazi.



charset - tokens

In python 3 i file sorgenti sono “encodati” in UTF-8.

L’encoding è un problema tipico soprattutto quando si lavora su file in ambiente windows, dove l’encoding di default è iso-8859-1.

In python ogni linea logica è una sequenza di componenti lessicali noti come *tokens*, questi sono *identifiers*, *keywords*, *operators*, *delimiters*, *literals*.

identifiers → nome usato per specificare una variabile, funzione, classe, modulo o altro oggetto. Inizia con una lettera o un underscore(_) seguito da 0 o più lettere, underscore o cifre. **Case matter!**

keywords → identificatori riservati per uso sintattico (and, as, assert, break, class, continue, def, del, elif, else, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield)



tokens

operators → + - * / % ** // << >> & | ^ ~ < <= > >= <> != ==

delimiters → () [] { } , : . ` = ; @ += -= *= /= //= %= &= |= ^= >>= <<= **=
' e " circondano literals di tipo stringa
fuori da una stringa inizia un commento
\\ alla fine di una linea fisica unisce la successiva in un'unica linea logica

literals → in un programma si tratta della denotazione di un dato (numero, string, contenitore)

```
>>> 42 # Integer literal
42
>>> 1.0j # Imaginary literal
1j
>>> """ Good
... night""" # Triple-quoted string literal
' Good\\nnight'
>>> [1, 2, 3] # list
[1, 2, 3]
```



tipi di dato

python - variabili -

Python è object oriented e con tipizzazione dinamica delle variabili (queste sono però strettamente tipizzate). Non è necessario dichiarare le variabili prima di usarle né dichiarare il loro tipo. Ogni variabile in Python è un'istanza di una classe che si può assegnare a un identificatore.

```
# commento con il simbolo "#"
# == octothorpe

my_var = 42
```

commento (non eseguito)

istanza della classe **INT**

identificatore



I principali “tipi” di numero che Python supporta sono numeri interi e numeri in virgola mobile (supporta anche i numeri complessi). [La built-in function `type()` ritorna il tipo di oggetto]

<class 'int'>

```
>>> type(3)
<class 'int'>
>>> x=3
>>> type(x)
<class 'int'>
>>> y=int('3')
>>> type(y)
<class 'int'>
>>> █
```

check istanza

check istanza
assegnata a
identificatore

check istanza
creata con
costruttore

<class 'float'>

```
>>> type(1.5)
<class 'float'>
>>> x=1.5
>>> type(x)
<class 'float'>
>>> y=float('1.5')
>>> type(y)
<class 'float'>
>>> █
```



python - numeri -

L'interprete si comporta come una semplice calcolatrice: è possibile scrivere un'espressione e l'interprete restituirà il valore della stessa. La sintassi dell'espressione è semplice: gli operatori $+$, $-$, $*$ e $/$ funzionano esattamente come nella maggior parte degli altri linguaggi; le parentesi $()$ possono essere utilizzate per il raggruppamento.

```
>>> # python rispetta l'ordine convenzionale delle operazioni
>>> 2 + 2
4
>>> 50 - 5 * 6
20
>>> (50 - 5 * 6) / 6
3.3333333333333335
>>> 4 / 2 # la divisione torna sempre il numero in virgola mobile
2.0
>>> █
```



python - stringhe -

Le stringhe possono essere definite indifferentemente con single quote (' ') o con double quote (" "). Sono sequenze immutabili di caratteri testuali sulle quali è possibile iterare.

```
>>> "hello" == 'hello' # stessa stringa con single and double quote
True
>>> print('hello') # built-in function print() visualizza la stringa sul file del flusso di testo
hello
>>> █
```

```
>>> for letter in 'hello':
...     print(letter)
...
h
e
l
l
o
>>> █
```

iterazione su string con
for loop



python - None/bool -

None è l'unica istanza della classe NoneType, può essere utilizzato come valore nullo da assegnare ad un identificatore (in altri linguaggi si utilizza la keyword "null") ed è il valore restituito di default dalle funzioni

La classe bool viene utilizzata per manipolare i valori logici (booleani) e le uniche due istanze di quella classe sono True e False. True "equivale" a 1 e False a 0.

```
>>> x = None
>>> type(x)
<class 'NoneType'>
>>> def no_return():
...     pass
...
>>> print(no_return())
None
>>> █
```

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
>>> True == 1 and False == 0
True
>>> (True + 3) * False
0
>>> █
```



print

La funzione `print()` permette di stampare oggetti in output (tipicamente su schermo, ma anche su file o altri stream). Gli oggetti inseriti nella funzione vengono convertiti in stringhe e scritti sullo stream. È possibile inserire espressioni matematiche, variabili ed altro all'interno di `print` e il risultato sarà stampato.

```
>>> print("Python print function")
Python print function
>>> print(3+7)
10
>>> a = 'string'
>>> print(a)
string
>>>
```

stampa di stringa

stampa risultato
espressione algebrica

stampa valore variabile



python - stringhe -

Print ha diversi parametri. Tra i più comunemente usati si deve sicuramente menzionare *end*

end → il valore di default è “\n” (escape sequence: new line). Si può sostituire con qualsiasi stringa.

end=default(“\n”)

end=” (evita di creare una new line)

Terza funzione stampata sulla stessa riga della precedente (utilizzo di escape sequence: horizontal tab)

Nessun oggetto: stampa new line

INPUT

```
print("prima riga")  
print("seconda riga", end='')  
print("\tsempre riga", end='\ttab!\n')  
print()  
print("terza riga", end='\t...fine terza riga\n')
```

OUTPUT

```
prima riga  
seconda riga    sempre riga    tab!  
  
terza riga      ...fine terza riga
```



slicing

python - slicing -

Python offre una notazione chiamata slicing (to slice = affettare) che permette di accedere ad una serie di elementi di una sequenza ordinata (stringa, lista, etc.) attraverso il loro "index" (indice). Il primo elemento corrisponde ha index 0, il secondo 1 e così via).

La sintassi utilizza le parentesi quadre con i seguenti parametri:
[start: stop [: step]]

- **start** → indice di partenza [opzionale]. 0 di default
es. `"casa"[:3] == "cas"`
- **stop** → indice di arrivo NON compreso. Se non specificato, lo slicing arriva fino alla fine dell'oggetto
es. `"casa"[2:] == "sa"`
- **step** → differenza di index tra un elemento e il successivo [opzionale]. 1 di default (tutti gli elementi compresi tra start e stop [non compreso])
es. `"abcdefg"[0:5:2] == "ace"`



python - slicing -

Esempi:

```
>>> s = "slicing test"
```

```
>>> s[0]
'S'
>>> s[1]
'l'
>>> s[:4]
'slic'
>>> █
```

Accesso a
elemento
singolo

Slice dall'inizio
all'elemento con index
4 (non compreso)

Accesso a elemento
singolo (index egativo
per index inverso)

Slice di tutti gli
elementi con
index pari

step negativo per scorrere
l'oggetto in senso inverso

```
>>> s[-1]
't'
>>> s[::2]
'Siigts'
>>> s[::-1]
'tset gnicils'
>>> █
```



operatori

python - operatori -

- Gli operatori sono utilizzati per eseguire operazioni su valori e variabili.
- Gli operatori possono manipolare singoli oggetti e restituire un risultato.
- Gli elementi coinvolti sono indicati come operandi o arguments.
- Gli operatori sono rappresentati da parole chiave o caratteri speciali.



python - operatori aritmetici -

Gli operatori aritmetici sono usati con valori numerici per eseguire operazioni matematiche comuni.

Operatore	Nome	Esempio
+	Addizione	$3 + 2 = 5$
-	Sottrazione	$3 - 2 = 1$
*	Moltiplicazione	$3 * 2 = 6$
/	Divisione	$3 / 2 = 1.5$
%	Modulo (restituisce il resto)	$3 \% 2 = 1$
**	Potenza	$3 ** 2 = 9$
//	Floor Division	$3 // 2 = 1$



python - operatori di assegnamento -

Gli operatori di assegnamento (augmented assignment) sono utilizzati per assegnare istanze agli identificatori.

Operatore	Esempio	Uguale 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>
/=	<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>
**=	<code>x **= 5</code>	<code>x = x ** 5</code>



python - operatori di confronto -

Gli operatori di confronto sono utilizzati per comparare due valori.

Operatore	Nome	Esempio
==	Uguale	<code>x == y</code>
!=	Non uguale	<code>x != y</code>
>	Maggiore di	<code>x > y</code>
<	Minore di	<code>x < y</code>
>=	Maggiore/Uguale	<code>x >= y</code>
<=	Minore/Uguale	<code>x <= y</code>



python - operatori di confronto -

A differenza di altri linguaggi, Python offre la possibilità di concatenare gli operatori di confronto:

Prendiamo ad esempio questa condizione:
 $a < b < c$

La sintassi più comune è sicuramente:
 $a < b$ and $b < c$

Python offre la possibilità di scrivere:
 $a < b < c$

```
>>> 2 < 5 and 5 < 7
True
>>> 2 < 5 < 7
True
>>> 2 < 5 < 4
False
>>>
```



python - operatori logici -

Gli operatori logici sono utilizzati per combinare dichiarazioni condizionali.

Operatore	Descrizione	Esempio
and	Ritorna True se entrambe le dichiarazioni sono vere	$x < 5$ and $x < 10$
or	Ritorna True se una delle due dichiarazioni è vera	$x < 5$ or $x < 4$
not	Inverte il risultato, ritorna False se il risultato è vero.	not $x > 5$



python - operatori di identità -

Gli operatori di identità sono usati per confrontare gli oggetti, non per verificare se i valori sono uguali, ma se sono lo stesso oggetto (che occupa la stessa locazione nella memoria).

Operatore	Descrizione	Esempio
is	Ritorna True se gli oggetti confrontati SONO lo stesso oggetto.	x is y
is not	Ritorna True se gli operandi comparati NON SONO lo stesso oggetto.	x is not y



python - operatori di appartenenza -

Gli operatori di appartenenza controllano l'appartenenza di un oggetto ad una sequenza come una lista, una stringa o una tupla.

Operatore	Descrizione	Esempio
in	Ritorna True se un oggetto con un determinato valore è presente nella sequenza.	x in y
is not	Ritorna True se un oggetto con un determinato valore NON è presente nella sequenza.	x not in y



dichiarazioni condizionali

python - operatori di appartenenza -

```
if prima_condizione:  
    primo_blocco  
elif seconda_condizione:  
    secondo_blocco  
elif terza_condizione:  
    terzo_blocco  
else:  
    quarto_blocco
```

I costrutti condizionali forniscono un modo per eseguire un blocco di codice scelto in base alla valutazione in fase di esecuzione di una o più espressioni booleane.

N.B. A differenza di altri linguaggi, Python si basa su l'indentazione, usando 4 spazi, per definire i blocchi di codice.



loop

python - while loop -

Python utilizza due tipi di loop:

- while
- for

Il while loop esegue un blocco di codice fintanto che una condizione è vera.

IMPORTANTE!
Incrementare *i* evita di creare un loop infinito.

```
i = 1  
  
while i < 6:  
    print(i)  
  
    i += 1
```



python - for loop -

Il for loop è uno strumento utile per iterare su una serie di elementi. La sintassi del for-loop può essere usata su qualsiasi struttura iterabile (liste, tuple, stringhe, set, dizionari o file).

Struttura iterabile con identificatore "primes" (lista)

Il blocco di codice viene eseguito una volta per ciascun elemento dell'oggetto iterabile

prime → identificatore assegnato ad ogni passaggio al rispettivo elemento

```
primes = [2, 3, 5, 7]
```

```
for prime in primes:  
    print(prime)
```

```
out: 2  
     3  
     5  
     7
```



python - continue / break -

La keyword *break* viene usata per uscire da un for loop o un while loop anche se l'iterazione sugli elementi non è finita o anche se la condizione è rispettata

output: 0
1
2
3
4

```
i = 0
while i < 7:
    if i == 5:
        print(i)
        break
    print(i)
```

La keyword *continue* viene usata per saltare il blocco corrente di codice e tornare alla dichiarazione *for* o *while*

```
for l in "casba":
    if l == "b":
        continue
    print(l)
```

output: c
a
s
a



exception handling

python - exception handling -

Quando si verifica un errore (o “eccezione”, come viene chiamata in Python) l’esecuzione del codice si interrompe e Python genera un messaggio d’errore. Queste eccezioni possono essere gestite usando le clausole `try` ed `except`.

`TypeError`: errore generato quando un’operazione o una funzione sono applicate ad un oggetto di tipo non appropriato. `int + str` → `raise error`

```
try:  
    1 + 'a'  
except:  
    print("Si è verificata un'eccezione.")
```

La keyword `except` gestisce qualsiasi tipo di eccezione. Se si vuole trattare le possibili eccezioni diversamente, si fa seguire ad `except` il tipo di errore. Tra i più comuni ci sono `AttributeError`, `IndexError`, `KeyError`, `NameError`, `TypeError`, `ValueError`, etc.



python - exception handling - else

La dichiarazione `try/except` ha una clausola `else` opzionale, che, quando presente, deve seguire tutte le clausole `except`. È utile per inserire codice che deve essere eseguito se la clausola `try` non genera un'eccezione...

```
try:  
    f = open('test.txt')  
except FileNotFoundError:  
    print(f"{f.name} non trovato")  
else:  
    print("File aperto correttamente")
```

Tentativo di aprire
"test.txt"

Gestione eccezione

Se non viene riscontrata
nessuna eccezione, viene
eseguita la clausola `else`



python - exception handling - finally

...quando invece si vuole eseguire del codice a prescindere dal risultato di try/except, si aggiunge la clausola **finally**. Questo codice verrà SEMPRE eseguito in qualsiasi circostanza.

Tentativo di
scrittura

Clausola
else
eseguita se
non si
presentano
eccezioni

```
f = open('test.txt', 'x')  
  
try:  
    f.write("Lorem ipsum dolor sit amet")  
except:  
    print("Errore durante la scrittura di {}".format(f.name))  
else:  
    print("Scrittura in corso")  
finally:  
    print("Chiusura del file in corso")  
    f.close()
```

Creazione
nuovo file

Gestione
eccezioni

Chiusura
file (sempre
eseguito)



python - assert -

Assert è un controllo di integrità che spesso si inserisce all'inizio di una funzione per verificare l'input valido e dopo una chiamata di funzione per verificare la validità dell'output.

Quando incontra `assert`, Python valuta l'espressione dichiarata dopo la keyword che si spera sia vera. Se l'espressione è falsa, Python solleva un'eccezione `AssertionError`.

Sintassi → `assert Espressione [, Arguments]`

- controllo che l'input della funzione sia un int o un float con la built-in function `isinstance()`;
- messaggio personalizzato di errore passato come argument;

```
>>> def area_quadrato(l):  
...     assert isinstance(l, (int, float)), "Input Errato"  
...     return l ** 2  
...  
>>> area_quadrato(3.5)  
12.25  
>>> area_quadrato('a')  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
  File "<stdin>", line 2, in area_quadrato  
AssertionError: Input Errato
```

`AssertionError` sollevato solo con la seconda invocazione della funzione in quanto viene passata una stringa.



python - raise -

La dichiarazione raise permette di sollevare un'eccezione specifica.

Se si vuole determinare se è stata sollevata un'eccezione ma non si intende gestirla, la semplice dichiarazione della keyword **raise** consente di sollevare nuovamente l'eccezione.



```
>>> raise NameError("Sollevo un eccezione 'Name Error'")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: Sollevo un eccezione 'Name Error'
```

```
>>> try:
...     raise NameError("Eccezione sollevata")
... except:
...     print("Eccezione gestita")
...     raise
...
Eccezione gestita
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: Eccezione sollevata
```

esercizi

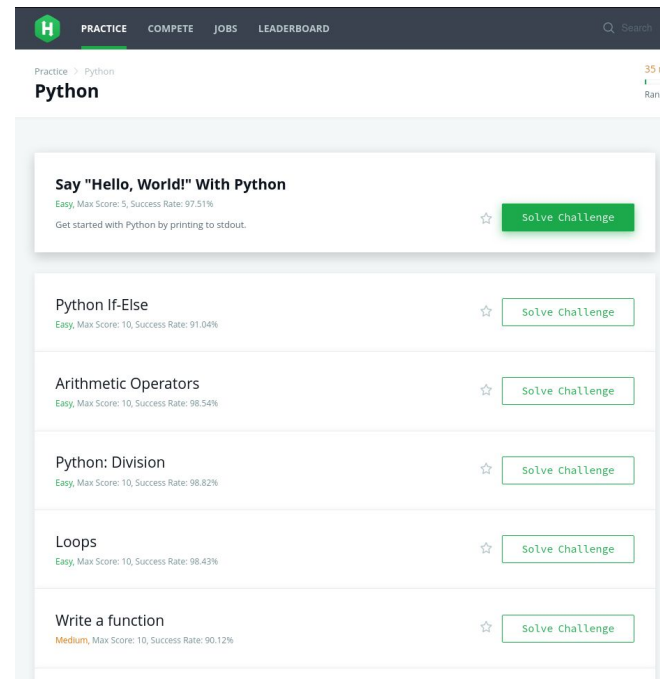
esercizi

“Hard work beats talent when talent doesn’t work hard”

Svariati siti online su cui allenarsi:

<https://www.w3schools.com/python/>

<https://www.hackerrank.com/domains/python>

A screenshot of the HackerRank website's Python practice section. The page has a dark header with navigation links: PRACTICE, COMPETE, JOBS, and LEADERBOARD. Below the header, the page title is "Python". The main content area lists several challenges with their difficulty levels, max scores, and success rates. Each challenge has a "Solve Challenge" button. The challenges listed are: "Say 'Hello, World!' With Python" (Easy, Max Score: 5, Success Rate: 97.51%), "Python If-Else" (Easy, Max Score: 10, Success Rate: 91.04%), "Arithmetic Operators" (Easy, Max Score: 10, Success Rate: 98.54%), "Python: Division" (Easy, Max Score: 10, Success Rate: 98.82%), "Loops" (Easy, Max Score: 10, Success Rate: 98.43%), and "Write a function" (Medium, Max Score: 10, Success Rate: 90.12%).

1.parte - esercizi

esercizi - parte 1.1 -

`input()` → funzione che permette ad un utente di immettere una stringa durante l'esecuzione di uno script [N.B. i caratteri inseriti attraverso `input()` vengono SEMPRE convertiti da Python in stringa].

1. Scrivi uno script che accetta un input consistente in un numero intero rappresentante un raggio, calcola l'area del rispettivo cerchio e stampa il risultato.
2. Scrivi uno script che accetta tre numeri interi. Se sono tutti diversi, stampa la somma dei tre; se due sono uguali, stampa il risultato della somma di quelli uguali divisi per il terzo; se tutti e tre sono uguali, stampa il risultato di $(n + n)^n$.
3. Scrivi uno script che accetta due numeri interi. Se valore di entrambi, la loro somma o la loro differenza è 5, stampa "True", altrimenti stampa "False". Se gli input inseriti non sono numerici, stampa "L'input deve essere un numero."



esercizi - parte 1.2 -

4. Scrivi uno script che accetta un input con nome e cognome e stampa la stringa con i caratteri di "*index*" dispari in ordine inverso.
5. Scrivi uno script che accetta un intero n , calcola e stampa il valore di $n + n*n + n*n*n$
6. Scrivi uno script che accetta una stringa e aggiunge "Sono" davanti alla stessa. Se la stringa comincia già con "Sono", stampa la stringa inalterata.
7. Scrivi uno script che calcoli la differenza tra un numero dato e 17. Se il numero è minore di 17, stampa il doppio della differenza assoluta (Python possiede la funzione `abs(n)` che ritorna il valore assoluto di un *int* o *float*).
8. Dato un numero intero n , esegui le seguenti operazioni condizionali:
 - se n è dispari, stampa "bizzarro";
 - se n è pari e tra 2 e 5 (compresi), stampa "non bizzarro";
 - se n è pari e tra 6 e 20 (compresi), stampa "bizzarro";
 - se n è pari e più grande di 20, stampa "non bizzarro".

