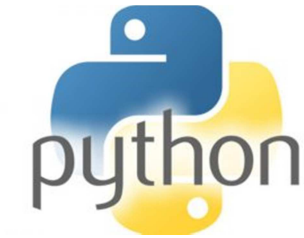
Python



Python è un linguaggio di programmazione creato nel 1991 da Guido Van Rossum.

E' un linguaggio semplice, adatto a tutti, ed usato in moltissimi ambiti programmazione che sta crescendo rapidamente in tutto il mondo :

- sviluppo web
- Raspberry
- data science
- machine learning
- scripting
- database
- data mining

Il Linguaggio

Il **Python** è un **linguaggio del tipo interpretato**, significa che il codice viene letto al da un "traduttore" e trasformato dopo qualche passaggio in linguaggio "macchina".

Il passaggio da codice ad esecuzione in ogni caso è piuttosto rapido**.** Un altro esempio di linguaggio interpretato è il JavaScript. Il Python è anche orientato agli oggetti (Object Oriented);

Quando prendiamo un'entità come "albero" e la fissiamo nel codice Python come Tree, non stiamo facendo altro che **creare un nuovo oggetto**.

Si dice che Python è orientato agli oggetti perché è molto facile creare nuovi oggetti e ragionare su di essi in modo molto comodo.

Ma passiamo al pratico con un esempio base :

```
    print("Hello Python!")
```

Possiamo notare come sia spostata di quattro spazi verso destra (tabulazione).

Se faccio girare questo codice Python avrò un errore (detto anche eccezione) : **IndentationError**.

Che cosa significa? In Python è importante rispettare la cosiddetta **indentazione**, ovvero la tabulazione del codice.

Il motore Python legge il codice in base a come è indentato.

L'esempio di prima non girerà, ma rispettando invece la tabulazione :

```
print("Hello Python!")
```

vedrete "Hello Python!" .

Ambiente di funzionamento di Python

Per usare Python potete scaricarlo ed installarlo sul computer oppure utilizzare alcune piattaforme di sviluppo on-line presenti nel web.

Inoltre Python è già installato su alcuni sistemi operativi come Linux (Fedora, Ubuntu), mentre per altri (Windows, Mac) dovrete scaricarlo ed installarlo.

Ci sono due modi per usare Python sul computer.

La prima opzione è creare dei file in codice Python ed estensione **.py**, per poi lanciarli con l'eseguibile Python.

Per stampare un semplice messaggio sul terminale crea un file di nome **hello.py** con il seguente codice:

```
print("Hello Python!")
```

eseguilo quindi con :

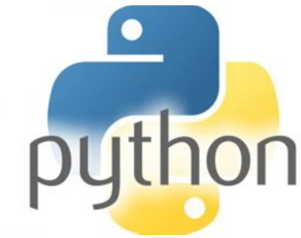
```
python3 hello.py
```

La seconda opzione per far girare codice Python: **la console Python**. Dopo aver installato Python, apri un terminale e lancia l'eseguibile python3:

```
python3
```

Ti ritroverai nella console di Python (dovresti vedere il simbolo >>>) dove potrete eseguire piccole istruzioni e scrivere codice per rapidi test:

```
>>> print("Hello Python!")
```



Dati di base

Quando parliamo di **tipi di dato** in programmazione ci riferiamo a dei blocchi fondamentali su cui è costruito tutto il resto.

In Python un **dato** può essere di questi tipi :

TIPO	ESEMPIO	NOME ESTESO
float	41.77	floating point
int	88	integer
str	"Cate"	string
bool	True	boolean
None	None	None

None è un valore nullo, una sorta di segnaposto.

Ogni tipo di dato corrisponde ad un valore che può essere associato ad un contenitore, le cosiddette **variabili**.

Le variabili

Nota: da questo momento per fare pratica esegui gli esempi che ti propongo nella console di Python (o salvali in un file e lanciali con l'eseguibile).

In "Python è dinamico" ho parlato di variabili come di scatole in cui mettere "roba".
Le variabili in Python sono ovunque e servono per contenere dei dati.

Quando creiamo una nuova variabile la stiamo **dichiarando** e per dichiarare variabili in Python basta fissarne il nome, seguito da un valore.

```
name = "Cate"
age = 31
city = "Rotterdam"
married = False
count = 105.00
```

Quando ad una variabile viene assegnato un particolare **tipo**, quella variabile diventa "di quel tipo". Qui sopra ad esempio "name" è una stringa, "age" è un numero, e così via.

Vi ricordo che se una variabile nasce stringa non è detto che debba restare tale, infatti è possibile poter cambiare il tipo:

```
name = "Cate"
name = 81
# Prima name è una stringa
# e poi diventa un numero
```

Vale la pena chiarire anche che Python considera come variabili tutti i nomi che nel programma non sono contenuti tra virgolette. Questo significa che il seguente esempio:

```
name = "Liz"
```

si legge come stringa Liz assegnata alla variabile name.

Il Casting

Il casting è quell'operazione che restituisce il valore di un'espressione (o una variabile) convertito in un altro tipo. Difatti la variabile resta comunque del tipo originale.

Le funzioni per il casting sono:

– **int()**: converte una variabile al tipo di dato intero;
– **str()**: converte una variabile al tipo di dato stringa;
– **float()**: converte una variabile al tipo di dato float, cioè un numero decimale.

Ricordiamo che le stringhe vanno scritte tra apici singoli o doppi apici.

a='13' # a viene vista come stringa e non come numero. Allo stesso modo se scrivo a="13".

Esempio :

```
>>>a='13'    #in questo caso a è una stringa in quanto ci sono gli apici.
>>>a*2       #moltiplichiamo la variabile a per 2.
'1313'       #otterremo la ripetizione della stringa 13 due volte.
```

```
>>>a=int(a)  #convertiamo ad intero, facendo il casting.
>>>a*2       #facciamo nuovamente il prodotto di a per 2.
26            #in questo caso otterremo il prodotto di 13 per 2.
```

Quello che abbiamo fatto in questo punto: **a=int(a)** rappresenta un'operazione di casting.

La variabile **a** inizialmente contiene una stringa ma, con l'operazione di casting **int**(), successiva viene convertita in intero.

Esempio :

```
>>> b=3       #in questo caso la variabile b è un intero.
>>> b
3
>>> b=str(b)
>>> b
'3'
```

In questo caso **b** da intero viene convertito in stringa utilizzando la funzione **str**().

Scrivere variabili alla maniera di Python

Non solo Python si aspetta di trovare il codice ben allineato, devi fare attenzione ad alcune regole che il mondo Python si è dato.

Facciamo qualche esempio. In alcuni linguaggi di programmazione le variabili vengono scritte con la forma cosiddetta **camel case**:

```
# Non farlo in Python
myVariable = "Liz"
```

In Python questa forma è da evitare perché accettata dalla community.

La forma corretta invece è:

```
# ok!
my_variable = "Liz"
```

Separa quindi con un underscore i nomi lunghi.

Un'altra cosa da evitare è il maiuscolo (a parte per un caso particolare):

```
# Non farlo in Python
MYVARIABLE = "Liz"
```

Non è raro vedere cose del genere :

```
# NON CI SIAMO
Nome = "Cate"
ETà = 31
CITTA = "Rotterdam"
count = 105.00
```

E' buona norma in Python scrivere le variabili in minuscolo.

Questi piccoli accorgimenti migliorano la **leggibilità del codice** e lo rendono facile anche agli altri sviluppatori.

Operatori aritmetici

OPERAZIONE	**OPERATORE**
addizione	+
sottrazione	-
moltiplicazione	*
esponente	**
divisione	/
divisione intera	//
modulo	%

Il simbolo della percentuale è detto modulo e restituisce il resto di una divisione:

```
>>> 80 % 2
0
```

Python ha anche un operatore di divisione intera detto **floor division**.

La divisione intera ritorna la parte intera del risultato nel caso in cui lo stesso sia decimale (float):

```
# divisione normale
>>> 89 / 4
22.25
# divisione intera
>>> 89 // 4
22
```

I commenti

Il simbolo **#** è un **commento**, e serve per lasciare indicazioni nel codice, sia per gli altri sviluppatori, sia per noi stessi.

I commenti vengono ignorati da Python e sono molto utili per **annotare e spiegare il codice**, eliminare temporaneamente istruzioni, e così via.

Ci sono due tipi di commento, il commento a **linea singola**:

```
# ciao, sono un commento
name = "Amy"
```

ed il commento a **linea multipla**:

```
"""
This is a
multi line
comment
"""
name = "Andrea"
```

Come avrai intuito i **commenti multilinea** sono utili per illustrare l'intento del nostro codice e possono fare anche da **documentazione**.

Le stringhe

Nella programmazione le stringhe sono ovunque e vale la pena spendere qualche parola su come usarle.

Alcuni operatori aritmetici funzionano anche sulle **stringhe**, come l'**addizione** e la **moltiplicazione** :

```
>>> "a" + "b"
"ab"


>>> "bb" * 4
"bbbbbbbb"
```

L'addizione può **concatenare più stringhe insieme** mentre la moltiplicazione **le ripete**.

Ad esempio per **concatenare più parole intervallate da spazi** potreste fare qualcosa del genere:

```
>>> "ciao" + " " + "il mio nome è" + " " + "Vale"
'ciao il mio nome è Vale'
```

L'esempio sopra può essere riscritto con **f-strings** (funzione string) in questo modo:

```
>>> f"ciao il mio nome è Vale"
'ciao il mio nome è Vale'
```

Basta richiamare **f** seguito dalle virgolette.

Le stringe possono essere anche salvate all'interno di variabili:

```
name = "Cate"
```

Quindi combinando **f-strings** e le variabili è possibile **concatenare variabili all'interno di stringhe**:

```
>>> name = "Cate"
>>> f"ciao il mio nome è {name}"
'ciao il mio nome è Cate'
```

Esistono anche altri modi di concatenare ed interpolare stringhe ma **f-strings** è la forma più chiara.

Operatore in (sulle stringhe)

Oltre agli operatori aritmetici, Python ha anche altri tipi di **operatori** non aritmetici, **l'operatore in**, chiamato in gergo **membership operator**.

L'operatore **in** in particolare è molto utile sulle per **controllare se un carattere oppure una sotto-stringa compare nella stringa** bersaglio:

```
>>> "guest" in "1 - insert a new guest"
True

>>> "g" in "1 - insert a new guest"
True

>>> "Gastgeber" in "1 - insert a new guest"
False
```

L'operatore **in** risponde sempre alla seguente domanda: x contiene y ? La risposta è un valore booleano, che è True se l'elemento da controllare esiste o meno nella stringa bersaglio.

Scambio di variabili

Esempio:

Dati in input i valori di due variabili **a** e **b**, vogliamo scambiarne i valori, così che al termine **a** contiene il valore di **b** e **b** quello di **a**.

a=5 e b=4
vogliamo che al termine del nostro algoritmo sia a=4 e b=5.
Per scambiare questi valori, possiamo utilizzare una variabile temporanea di appoggio.
Chiamo questa terza variabile **temp**, e procediamo così:

```
temp=a      #nella variabile temp memorizzo il valore di a
a=b         #in a memorizzo il valore della variabile b
b=temp      #in b memorizzo il valore della variabile temp che contiene il valore di a
```

avrò alla fine : a=4 e b=5.

NOTA : C'è però un altro metodo per scambiare le variabili in python.

Assegnazione multipla

Esempio:

Con l'**assegnazione multipla** in python si possono assegnare più variabili alla volta.

a=b=5
In questo caso sia a che b assumono il valore di 5.

Oppure un'altra assegnazione che si può operare è la seguente :
a, b= 5, 4

In questo modo assegno ad **a** il valore 5 e a **b** il valore 4.

Scrivendo : **a, b=b, a** otterrò lo scambio dei due valori.

Esempio completo qui sotto :

```
a=int(input('Inserisci a:'))
b=int(input('Inserisci b:'))
print('I valori inseriti sono a:', a, ' e b: ', b)
print('Adesso scambio i valori')
a,b=b,a
print('I valori scambiati sono a:', a, ' e b: ', b)
```

Gli Operatori

Operatori di confronto

Gli operatori di confronto vengono utilizzati nelle istruzioni condizionali (if else) e non solo e possono restituire solo due valori: **True** o **False**.

Facciamo dunque un esempio, assumendo che la variabile **a** sia uguale a 5 e **b** sia uguale a 6.

== uguale Es: a==b restituisce False

!= diverso Es: a!=b restituisce True

> maggiore Es: a>b restituisce False

< minore Es: a<b restituisce True

>= maggiore o uguale Es: a>=b restituisce False

<= minore o uguale Es: a<=b restituisce True

Provate pure in modalità interattiva ad eseguire questi esempi.