

Ch. 1 ~ Python intro

Perchè scrivere una guida di python?

Ce ne sono migliaia sul Web, ma la mia idea è quella di creare uno strumento semplice ed immediato, che si concentra più sulla sostanza che sulla forma.

E' bene precisare che questo tutorial è indirizzato a chi già possiede una conoscenza più o meno base di un linguaggio di programmazione quale R, Matlab, Java, C++ etc.

Mi scuso per le 'espressioni gergali' che verranno inserite di tanto in tanto, ma vorrei sottolineare che voglio essere il più diretto possibile! (Poi diciamocelo, sarà mica una guida seria..)

Nota Benissimo Visto e considerato che ci sarebbero milioni di cose da scrivere, ed io mi vorrei limiare all'essenziale, spesso e volentieri lascerò la scritta *S.O.* seguita da una stringa di parole apparentemente senza senso; in questo caso dovreste skillarvi nell'utilizzo del mood **Stack Overflow**, ovvero COPIA + INCOLLA + QUERY SU GOOGLE.



Figure 1: La tastiera che vi servirà

Nota Benissimo 2.0 Una delle skill più importanti se già non lo sapete non è tanto saper programmare quanto imparare a fare *query su google* o più semplicemente *sgooglare*. Ricordatevi che se avete un problema ci sarà qualche anima che ci ha già sbattuto la testa ed ha scritto qualcosa in merito, oppure il vostro problema è talmente complicato che siete voi i primi a doverlo risolvere (ma per le cose che verranno trattate dubito fortemente). Su internet c'è tutto, il problema è trovarlo.

Tendenzialmente il metodo più efficace per cercare qualunque cosa su google:

Installare Python

Per evitare ogni sorta di dubbio: ci sono due versioni di python disponibili. Quella *fortemente consigliata* è la 2.7.14.

S.O. PYTHON DOWNLOAD + seguire istruzioni

Notebook o Script, questo è il dilemma!?

Python è un linguaggio di programmazione, fin qui ci siamo. Può essere utilizzato tramite diversi *canali*, senza dilungarsi troppo, cercate il più comodo per voi. Gli strumenti più comodi per imparare a programmare sono senza dubbio i notebook. Offrono la possibilità di scrivere piccole porzioni di codice e computare quella parte. Il vantaggio è l'esecuzione sequenziale delle operazioni e la possibilità di trovare abbastanza velocemente eventuali errori. Personalmente mi trovo molto bene con Jupyter.

S.O. DOWNLOAD ANACONDA → dopodichè aprite Jupyter e iniziate a smanettare come se non ci fosse un domani.

Gli script invece sono file di testo salvati con l'estensione .py che vengono eseguiti integralmente, dalla prima all'ultima riga di codice. Si perde in questo caso la visione d'insieme dei notebook, ma in contenisti più specifici del semplice apprendimento trovano la loro naturale collocazione. **S.O** DOWNLOAD Pycharm → è una procedura un po' tediosa, ma basta sgooglare e ce la farete!

Identazione

Python funziona SE E SOLO SE il codice è indentato nella maniera corretta! Ovvero le righe di codice devono essere 'allineate' bene e non da bestie. Per chi proviene da linguaggio come R o C++ all'inizio sarà particolarmente complicato, ma posso garantire che ci si abitua in fretta.

```
a = 5
b = [3, 2, 'ciao']
if a > 3:
    print a, '\n'
    for i in range(len(b)):
        print i, b[i]
```

5

0 3

1 2

2 ciao

Ch. 2 ~ Dati

Ch. 2.1 Variabili & affini

Commenti

Inizialmente farete una fatica del diavolo a migrare da qualunque linguaggio di programmazione a python, quindi inizierete a commentare ogni riga di codice per capire di cosa si tratta. Per scrivere i commenti risulta molto semplice, basta utilizzare il `#` commento. Nel caso in cui dobbiate commentare più righe, è sufficiente “aprire” il commento con tre apici `'''` commento riga 1, commento riga 2, commento riga 3 `'''` e chiuderlo con tre apici.

```
#questo è un commento
#questo è un altro commento

'''
se è necessario commentare generose porzioni di codice,
è sufficiente racchiudere fra tre apici
'''
```

Variabili

Esempio di come assegnare i valori alle variabili. L'assegnamento delle variabili è banale, utilizzando l'operatore `=`.

Nomi: Python è case-sensitive, pertanto `->QUESTO = 7` e `->questo = 7`, sono due variabili differenti!

Ci sono alcuni nomi da evitare: *False, None, True, and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield*

```
numero_intero = 5 #assegnamento standard
numero_reale = 0.543021
stringa = 'stringa'

x = 5 #ora x vale 5
x=7 #viene sovrascritto il 5
x='quanto vale x?' #viene sovrascritta la stringa 'quanto vale la x?'
```

Assegnamento multiplo

```
primo, secondo, terzo = 6, 'ciao' , 9.12

print primo, '\n'
print secondo, '\n'
print terzo
```

6

ciao

9.12

Numeri

Premessa A differenza di altri linguaggi di programmazione per la gestione di vettori, matrici etc è necessario importare un apposito pacchetto.

In questa sezione vengono trattati i numeri e le operazioni da 'calcolatrice'. So che fremete per smanettare su prodotti vettoriali, scomposizioni spettrali e quant altro. Abbiate pazienza, ogni cosa a tempo debito.

I dati numerici si distinguono in quattro categorie: **interi** (*int*), **razionali** (*float*), **complessi** (*complex*) e **booleani** (*bool*)

Operatori aritmetici

```
a, b = 5, 7

a + b #somma

a - b #sottrazione

a * b #moltiplicazione

a ** b #elevamento a potenza

a / b #divisione classica

a // b #parte intera

a % b #resto della divisione
```

Operatori di confronto, operatori logici & friends

Operatori di confronto (= > ! <): danno la possibilità di confrontare due elementi e vengono utilizzati per Restituiscono un valore *True* o *False* Operatori logici (and or not): si utilizzano per concatenare delle condizioni Friends(in): operatore utile che permette di verificare se un elemento è presente all'interno di una struttura dati, viene restituito un valore *True* o *False*.

```
a, b = 5, 7

#Operatori Logici
a == b #uguale

a != b #diverso

a > b #maggiore, >= per maggiore stretto

a < b #minore , <= per minore stretto

#Operatori di Confronto
if (a != b) and (a < b):
    print 'esegui qualche istruzione'

#Comando in
lst = [1, 2, 3, 7]

risp_1 = b in lst
risp_2 = a in lst
print '\n', risp_1, '\n'
print risp_2
```

esegui qualche istruzione

True

False

Ch 2.2 Stringhe

Variabili “di testo” sono definite come stringhe. In python sono viste come una sequenza di caratteri, e ciò fa di loro uno strumento molto versatile. E' possibile accedere ad un singolo elemento, ad una porzione (substring), verificarne la lunghezza o altre operazioni.

```
stringa = 'questa è una stringa'

seconda_stringa = "anche questa è una stringa"

#verifichiamo il tipo con la funzione type(oggetto) --> NOTA BENE, si può usare su qualunque oggetto
print type(stringa), '\n', type(seconda_stringa)

#verifichiamo la lunghezza della stringa con la funzione len(oggetto) --> NOTA BENE, si può usare su (q
print len(stringa), '\n', len(seconda_stringa)

<type 'str'>
<type 'str'>
21
27
```

Indexing

Python prevede una duplice indicizzazione in ciascun elemento: una prima classica che parte da zero fino all'ultimo elemento, ed una che parte dal ‘fondo’ con l'indice -1 fino all'inizio della stringa -(len(stringa)). Per selezionare un particolare elemento all'interno della stringa è sufficiente utilizzare le parentesi quadre accanto all'elemento desiderato, ed indicare l'indice corrispondente alla posizione desiderata. s[0] o s[-1]

```
s = 'python'
print s[0], ' --> primo elemento della stringa \'python\' '
print '\n'
print s[-1], ' --> ultimo elemento della stringa \'python\' '
```

p --> primo elemento della stringa 'python'

n --> ultimo elemento della stringa 'python'

Substring

Per selezionare porzioni di una stringa è sufficiente selezionare due i due indici che fanno riferimento alla porzione desiderata. Da ricordare che il primo elemento è incluso, mentre il secondo no!

```
s = 'python'

print s, '--> stringa iniziale'
print '\n'
print s[0:3], '--> dall\' indice 0-3'
print '\n'
print s[-3:-1], '--> dall\' indice -3 a -1'
print '\n'
```

```
print s[3:], '--> dall\' indice 3 alla fine della stringa'
print '\n'
print s[:3], '--> dall\' inizio fino all\' indice tre'
```

python --> stringa iniziale

pyt --> dall' indice 0-3

ho --> dall' indice -3 a -1

hon --> dall' indice 3 alla fine della stringa

pyt --> dall' inizio fino all' indice tre

Concatenamento ed utilizzo di in - not in

Per verificare se un elemento fa parte della stringa è sufficiente scrivere (tra virgolette) * 'th in 'python' *; ovviamente la risposta sarà true! Vale la stessa regola per numeri o sottostringhe.

Due stringhe si concatenano con l'operatore + .

```
uno = 'ba'
risultato = uno + 'na' * 2

print risultato
```

banana

METODI Trattandosi python di un linguaggio ad oggetti, è possibile utilizzare i 'metodi' associati a ciascun oggetto.

oggetto.metodo(argomenti)

esempio : stringa.lower()

```
s = 'python'

print s.upper(), '--> metodo upper'
print s.lower(), '--> metodo lower'
```

PYTHON --> metodo upper

python --> metodo lower

NOTA BENE Per vedere i metodi associati ad una stringa ci sono diverse soluzioni: -1 cercare su google, dopo il punto cliccare il 'tab' e comparirà il menu a tendina con tutti i metodi associati, oppure utilizzare l'help di python con il comando help(stringa.lower()), oppure con il comando dir(str) viene aperta la lista dei metodi associati.

S.O. METODI STRINGA PYTHON

Ch 2.3 Oggetti: Liste

Le liste sono fra gli oggetti più versatili nell'arsenale python. La natura mutabile e la possibilità di contenere oggetti eterogenei ne fa uno strumento adatto ad ogni occasione. Infatti le liste possono contenere al loro interno altre liste (nested-list), oppure dei dizionari, o delle tuple.

In generale la lista è definita fra parentesi quadre, e gli elementi sono separati al loro interno dalla virgola.

```
prima_lista = [1, 3, 4, 5]
print type(prima_lista), ' --> tipo di oggetto'
print len(prima_lista), ' --> lunghezza della lista'

print prima_lista, '\n'

print 'come visualizzare gli elementi in colonna'
for i in prima_lista:
    print i
```

```
<type 'list'> --> tipo di oggetto
4 --> lunghezza della lista
[1, 3, 4, 5]
```

```
come visualizzare gli elementi in colonna
1
3
4
5
```

```
seconda_lista = ['elemento uno', 1936.27, (19, 12)]

nested_list = [[1, 2, 3], ['a', 'b', 'z'], [1, 'ciao']]

print nested_list
```

```
[[1, 2, 3], ['a', 'b', 'z'], [1, 'ciao']]
```

Accedere agli elementi di una lista Per accedere agli elementi di una lista è sufficiente utilizzare le parentesi quadre, e selezionare la posizione dell'indice desiderato.

Per accedere ad un elemento della nested-list sono necessarie due coppie di parentesi quadre, la prima indicherà la posizione della nested-list all'interno della lista, la seconda indicherà la posizione dell'elemento all'interno della nested list

```
nst_list = [['gabriele', 'luca', 'daniel'], 'valerio', ['roberto', 'claudio']]

#per accedere a 'daniel'
print nst_list[0][2], '\n'

tup_lst = [(10, 12), (666, 999), (77, 23)]
#per accedere all'elemento 77

print tup_lst[2][0], '--> quello che ci interessa'

daniel
```

```
77 --> quello che ci interessa
```

Sub-list

Esattamente come le substring, per selezionare una porzione di lista è necessario utilizzare le parentesi quadre

ed inserire gli indici di riferimento.

```
lst = [1, 1, 3, 5, 8, 13, 21, 34, 55, 89]

print lst[:3], '--> dall\' elemento di posizione zero, fino a quello di posizione tre', '\n'
print lst[3:], '----> dall\'elemento di posizione tre, fino alla fine', '\n'
print lst[2:4], '--> dall\' elemento di posizione 2 a quello di posizione 4', '\n'

#per assegnare una porzione di lista utilizzare semplicemente l'uguale

a = lst[1:3]
print a, '--> nuova variabile', '\n'

#concatenazione di due liste
b = a + a
c = a *2
print b, ' == ', c, 'si tratta della stessa lista'
```

[1, 1, 3] --> dall' elemento di posizione zero, fino a quello di posizione tre

[5, 8, 13, 21, 34, 55, 89] ----> dall'elemento di posizione tre, fino alla fine

[3, 5] --> dall' elemento di posizione 2 a quello di posizione 4

[1, 3] --> nuova variabile

[1, 3, 1, 3] == [1, 3, 1, 3] si tratta della stessa lista

Nota Bene Le liste supportano le funzioni/metodi comuni alle sequenze di elementi, come min(lista), max(lista), len(lista), lista.index(indice), lista.count(elemento_di_interesse)

Metodi Di seguito sono riportati i metodi più comuni alle liste.

- lista.append(elem): aggiunge elem alla fine della lista;
- lista.extend(seq): estende la lista aggiungendo alla fine gli elementi di seq;
- lista.insert(indice, elem): aggiunge elem alla lista in posizione indice;
- lista.pop(): rimuove e restituisce l'ultimo elemento della lista;
- lista.remove(elem): trova e rimuove elem dalla lista;
- lista.sort(): ordina gli elementi della lista dal più piccolo al più grande;
- lista.reverse(): inverte l'ordine degli elementi della lista;
- lista.copy(): crea e restituisce una copia della lista;
- lista.clear(): rimuove tutti gli elementi della lista;

L'unico modo per prendere confidenza con questi oggetti è aprire un notebook e fare tante, ma tante (ma davvero tante) prove da soli.

Per qualunque problema, query su google ***

Ch 2.4 Oggetti: Tuple

Le tuple sono sequenze immutabili di oggetti racchiusi fra parentesi tonde.


```

prima_tupla = (10, 11, 23) #
seconda_tupla = ('io', 'tu', 'egli' )

print prima_tupla
print '\n'
print type(prima_tupla), '--> tipo di oggetto ' , '\n'
print len(prima_tupla), '--> lunghezza della prima_tupla', '\n'
print seconda_tupla

```

```
(10, 11, 23)
```

```
<type 'tuple'> --> tipo di oggetto
```

```
3 --> lunghezza della prima_tupla
```

```
('io', 'tu', 'egli')
```

Accedere agli oggetti della tupla

Così come nelle stringhe è necessario selezionare l'indice corrispondente all'elemento della posizione desiderata. `tupla[1]` -> restituisce il secondo elemento della tupla!

```

tup = ('gatto', 'topo', 'gallina')
print tup[1]

```

#verificare la natura degli oggetti della tupla.

```
print type(tup[1]), '--> tipo di oggetto nella prima (seconda) posizione delle tupla'
```

```
topo
```

```
<type 'str'> --> tipo di oggetto nella prima (seconda) posizione delle tupla
```

Metodi

L'oggetto tupla può essere utilizzato come argomento di funzioni, come per esempio `min(tupla)`, `max(tupla)`, `len(tupla)`, `type(tupla)`, `my_function(tupla)`.

Si possono applicare i metodi, tra i più utilizzati ci sono `'count()'` e `'index()'`

```
tup = ('Luca', 'Ilaria', 'Paolo', 'Claudio', 'Andrea', 'Tommaso', 'Luca', 'Giacomo', 'Luca')
```

```
print tup.index('Tommaso'), '--> restituisce l'indice associato a tommaso', '\n'
```

```
print tup.count('Luca'), '--> restituisce il numero di occorrenze di \' Luca \''
```

```
5 --> restituisce l'indice associato a tommaso
```

```
3 --> restituisce il numero di occorrenze di ' Luca '
```

Ch 2.5 Oggetti: Dizionari

BLA BLA BLA ***