2-Operazioni e variabili

March 27, 2021

1 Operazioni numeriche e variabili in python

Benvenuti alla seconda lezione di sfclub torino sull'introduzione di python e dell'applicazione di esso su Jupytero JupyterLab suo successore, partiamo dalle operazioni che sono possibile effettuare con esso e definiamo cosa sono le variabili. Iniziamo!

1.1 Operazioni numeriche

Python grazie alla possibilità di essere interpretato permette di eseguire anche singole righe di codice e operazioni come somma, prodotto, divisione, differenza e molto altro. Per eseguire il codice presente nella cella schiacciare ▶ in alto a sinistra o usare `Shift+Enter` da tastiera.

NOTA : il codice scritto dopo il carattere `#` non verrà interpretato poiché verrà classificato come commento e ha solo funzione esplicativa per il programmatore o colui che sta leggendo il codice per comprendere il codice

[1]:
```
# Io sono un commento e non produco nessun output
```

[2]:
```
1 + 1 #somma
```

[2]:
```
2
```

[3]:
```
1 - 5 #differenza
```

[3]:
```
-4
```

[4]:
```
3 * 5 # prodotto
```

[4]:
```
15
```

[5]:
```
5 / 6 #divisione
```

[5]:
```
0.8333333333333334
```

[6]:
```
6 % 5 #questo è il modulo ovvero il resto della divisione tra due numeri
```

[6]:
```
1
```

[7]:
```
2 ** 3 #elevazione a potenza
```

[7]:
```
8
```

[8]:
```
7 // 2 # questo è il floor division ovvero la divisione tra i due numeri␣
↪considerando solo il valore intero
```

[8]:
```
3
```

[9]:
```
3.6 * 2.8 #python accetta anche numeri non interi
```

[9]:
```
10.08
```

Python ovviamente supporta molte altre tipi di operazioni che sarebbe troppo lungo da elencare, qualora foste interessati è possibile consultare questo **link**. Inoltre è possibile definire nella stessa cella o riga operazioni multiple ricordando però che ci sono precedenze come i prodotti e divisioni vengono prima delle somme e le parentesi vengono valutate precedentemente, inoltre qualore ci fossero operzioni con la stessa precedenza l'ordine è da sinistra verso destra:

[10]:
```
1 + 5 * 5
```

[10]:
```
26
```

[11]:
```
(5 + 5) * 6
```

[11]:
```
60
```

[12]:
```
(6 + 6) * 4 / 6
```

[12]:
```
8.0
```

1.1.1 Note utili

Ovviamente è sempre meglio scrivere un codice pulito e semplice con l'utilizzo di parentesi o altre convezioni per i seguenti motivi:

Facilità di lettura

Facilità di modifica

Riduzione della difficoltà di risoluzioni di problemi

Quindi cercate di scrivere il codice nella maniera quanto più chiara possibile per evitare ad altri e a voi grossi problemi nella risoluzione di problemi futuri.

1.2 Variabili

Le variabili in informatica sono un riferimento logico che permettono di accedere a una parte della memoria destinata alla lettura o scrittura di un dato e in molti casi vengono usate per far eseguire delle operazioni da parte del computer, in python questa operazioni è molto semplice, vediamo ora come ripetere le operazioni fatte in precedenza con le variabili. Definiamo quindi due variabili, per semplicità `A` e `B`, associamole a dei valori e facciamo delle operazioni:

[13]:
```
A = 3 #la variabile A ora avrà come valore 3
B = 5 #la variabile B ora avrà come valore 5
```

[14]:
```
A + B #somma di A e B
```

[14]:
```
8
```

[15]:
```
A - B #differeneza di A e B
```

[15]:
```
-2
```

[16]:
```
A / B #divisione di A e B
```

[16]:
```
0.6
```

[17]:
```
A * B #prodotto tra A e B
```

[17]:
```
15
```

Nel caso in cui volessimo memorizzare i risultati ottenuti facendo le operazioni sulle variabile è possibile scrivere la seguente riga:

[18]:
```
somma = A * B # la nuova variabile somma conterrà il valore somma delle due␣
↪variabili
print(somma)
```

```
15
```

1.2.1 Regole sulle variabili

Ci sono delle precise regole su come chiamare una variabile in python, esse sono:

Il nome di una variabile deve iniziare con una lettera o il carattere _

Il nome di una variabile non può iniziare con un numero

Il nome di una variabile può solo contenere carartteri alfa-numerici(A-z, 0-9) e _

Il nome di una variabile è case sensitive ovvero io, Io, iO, IO sono variabili diverse

Ricordate anche di non chiamare due variabili con lo stesso nome, poiche questo potrebbe portare ad errori! Inoltre alcuni nomi sono riservati a delle operazioni e quindi non possono essere usati come sum, add, del ecc..

E' buona norma usare nomi di variabili significativi ai valori associati o alle operazioni che verranno eseguite su di esso, questa facilita ulteriormente la comprensione della logica del codice.

1.2.2 Eliminare una variabile

In alcuni casi potrebbe essere necessario eliminare una variabile per vari motivi, python contiene già in sè un meccaniso automatico attraverso cui libera la memoria dalle variabili non più usate, in alcuni casi però potrebbe essere necessario non rendere la variabile non più accessibile, è in questo caso che entra in gioco il comando **del**:

[19]:
```
del somma
print(somma)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-19-1ceb8c4380b2> in <module>
      1 del somma
----> 2 print(somma)

NameError: name 'somma' is not defined
```

Come possiamo vedere ora la variabile "somma" ha creato un errore che ci dice che la variabile non è più disponibile e questo significa che non possiamo più utilizzarla, ma attenzione la memoria allocata per "somma" potrebbe ancora essere occupata quindi fate molta attenzione a usare questo comando poiché **non libera la memoria**, python infatti ha un suo sistema di liberare memoria automatica contando le allocazioni e le deallocazioni nel codice, in particolare lui libera la memoria poco. Qualora si volesse **liberare memoria manualmente** allora si dovrebbe importare il **garbage compiler(gc)** importando il modulo, in tal caso il **gc** automaticamente capirà quale oggetto è stato svuotato con **del** e procederà al liberamento della sua memoria.

[20]:
```
import gc
print("Oggetti eliminati automaticamente o manualmente che sono stati liberati␣
 ↪dalla memoria: {}".format(gc.collect()))
```

```
Oggetti eliminati automaticamente o manualmente che sono stati liberati dalla
memoria: 244
```

1.2.3 Tipi di variabili

Le variabili possono contenere anche valori non numerici come in questo esempio:

[21]:
```
x = "anche io sono una variabile"
```

Questa variabile contiene una stringa, ovvero una sequenza di caratteri, che python riesce a riconoscere attraverso il carattere " o ' **all'inizio e alla fine**.

[22]:
```
stringa1 = "Questa è una stringa"
stringa2 = 'Anche questa è una stringa'
```

Python per fare ciò associa un particolare **tipo** ad ogni variabile e questo ci fornisce informazioni su come sia il valore associato a questa variabile, per sapere di quale tipo sia la nostra variabile python permette l'uso del comando `type()`

[23]:
```
print("Tipo della variabile x:", type(x))
print("Tipo della variabile A:", type(A))
print("Tipo del valore 3.5:", type(3.5))
```

```
Tipo della variabile x: <class 'str'>
Tipo della variabile A: <class 'int'>
```

```
Tipo del valore 3.5: <class 'float'>
```

Come possiamo vedere le variabile `x` è della classe 'str' che sta per stringa, `A` è della classe intero, mentre il valore `3.5` è della classe floating che indica i numeri con la virgola.

Esiste anche un tipo di variabili **booleane** ovvero variabili che contengono solo **vero** o **falso**:

[24]:
```
vero = True
falso = False
print('Tipo di vero: ', type(vero))
print('Tipo di falso: ', type(falso))
```

```
Tipo di vero:  <class 'bool'>
Tipo di falso:  <class 'bool'>
```

Operazioni sulle stringhe Sulle variabili stringa è possibile definire delle operazioni come:

[25]:
```
print(stringa1 + ',' +stringa2) # unione delle stringhe
```

```
Questa è una stringa,Anche questa è una stringa
```

[26]:
```
print(x * 3) #ripetizione della stringa 3 volte
```

```
anche io sono una variabileanche io sono una variabileanche io sono una
variabile
```

[27]:
```
print(x.split()) #divisione da un carattere, di default spazio
```

```
['anche', 'io', 'sono', 'una', 'variabile']
```

Ovviamente ce ne sono molte altre, fate pure riferimento a questo **link** per ulteriori informazioni.

Attenzione però che le alcuni operazioni con diversi tipi non sono concesse, come ad esempio:

[28]:
```
stringa1 + 3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-ece236a18fcf> in <module>
----> 1 stringa1 + 3

TypeError: can only concatenate str (not "int") to str
```

Il motivo è abbastanza semplice quanto fa un carattere più un numer, questo non è facile nemmeno per noi da rispondere e questo è segnalato
da python con TypeError e un messaggio che ci dice cosa non funziona.

1.2.4 Variabili globali e locali

Le variabil, siccome sono racchiuse all'interno di blocchi di codice posoono essere di due tipi in python **globali** o **locali**, globali vuol dire che possono essere accessibili dappertutto all'interno del

codice, mentre locali vuol dire che sono accessibili solo all'interno di un blocco, per essere più chiari facciamo un esempio:

[29]:
```
k = "bravo" # global k

#nuovo blocco
#questa è una funzione, la vedremo in dettaglio dopo
def myfunc():
    #definisco la stessa variabile con un diverso valore
    k = "grande" #local k
    print("variabile k dentro myfunc è: ", k)
    return None

myfunc()
print("La variabile k è: ", k)
```

```
variabile k dentro myfunc è:  grande
La variabile k è:  bravo
```

Cosa è successo? La variabile `k` nella prima riga era globale con un certo valore, ma non viene ridefinita dentro la funzione perché quella variabile non è globale, ma locale e pertanto non riscriverà la k definita prima, ma ne creerà una diversa che sarà accessibile solo dentro la funzione. Nel caso in cui noi volessimo rendere una variabile globale, possiamo usare la parola chiave **global**, come nel seguente modo:

[30]:
```
k = "bravo" # global k

#nuovo blocco
#questa è una funzione, la vedremo in dettaglio dopo
def myfunc():
    #definisco la stessa variabile con un diverso valore
    global k #ora k è globale
    k = "grande" #k viene riscritta
    print("variabile k dentro myfunc è: ", k)
    return None

myfunc()
print("La variabile k è: ", k)
```

```
variabile k dentro myfunc è:  grande
La variabile k è:  grande
```

Un'altra possibile applicazione è la seguente:

[31]:
```
#nuovo blocco
#questa è una funzione, la vedremo in dettaglio dopo
def myfunc():
    #definisco una nuova variabile
    global k #nuova variabile k globale
```

```
    k = "grande" #k viene definita
    return None

myfunc()
print("La variabile k è: ", k)
```

```
La variabile k è:  grande
```

Questo nuovo modo di usare global permette di definire una nuova variabile che sarà utilizzabile anche al di fuori della funzione e questo può essere molto utile in certi contesti.

COMPLIMENTI AVETE ORA COMPRESO COME FARE OPERAZIONI NUMERICHE E USARE LE VARIABILI!