Funzioni

November 9, 2020

1 Funzioni

Una funzione in informatica è un blocco di codice organizzato attraverso cui vengono compiute delle azioni. I motivi per utilizzare una funzione sono: - maggiore chiarezza del codice - unica definizione per una specifica azione - riduzione della lunghezza del codice - possibilità di ripetere la stessa azione con facilità

1.1 Creazione di una funzione

In python una funzione è definita attraverso la parola chiave `def` seguito dal nome della funzione che vogliamo usare e delle parentesi `()` che in genere contengono dei parametri, subito dopo di esso è presente il carattere `:` e il codice **indentato** contenente la **docstring** che fornisce informazioni sul suo utilizzo e le sue azioni seguito dalle istruzioni da eseguire e una riga finale contenente `return` e il valore che vogliamo ritornare anche in forma di variabile, la struttura logica è la seguente:

[ ]:
```
def nome_funzione(parametri):
    '''funzione docstring'''
    codice_funzione
    return valore
```

Ora che abbiamo capiro la sintassi da utilizzare vediamo degli esempi.

1.1.1 Required arguments - valori dei parametri richiesti

[1]:
```
def somma(x, y): #due parametri sono forniti in input
    '''somma due numeri ''' #informazioni sulla funzione
    tot = x + y #esegui la somma
    return tot #ritorna il valore della somma
```

Ora se noi scriviamo `somma()` e con il cursore all'interno delle parentesi schiacciamo `Shift + Tab` vedremo apparire una mini finestra con la **docstring** che abbiamo scritto nella funzione e altri informazioni. Vediamo ora di usare la funzione definita.

Stiamo attenti che qualora ci dimenticassimo di definire i valori dei parametri un errore verrà stampato dicendoci che ci siamo dimenticati di definire i parametri.

[2]:
```
somma() # non ho specificato valori
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-96de7c92fd92> in <module>
----> 1 somma() # non ho specificato valori

TypeError: somma() missing 2 required positional arguments: 'x' and 'y'
```

1.1.2 Keyword arguments - valori dei parametri specificati

[3]:
```
tot = somma(x = 3, y = 2) #tot avrà il valore della funzione return
print('La somma di 3 e 2 è:',tot)
```

```
La somma di 3 e 2 è: 5
```

Come possiamo vedere abbiamo passato dei valori effettivi alla funzione ponendo i parametri a specifici valori, è possibile anche non usare la sintassi `parametro = valore` e scrivere solo i valori, stando però attenti di mantenere l'ordine, a meno che specifici tipi delle variabili siano diversi, vedremo questo in seguito.

[4]:
```
tot = somma(3,2) # sottointeso x = 3, y = 2
print('La somma di 3 e 2 è:',tot)
```

```
La somma di 3 e 2 è: 5
```

1.1.3 Funzioni senza return o parametri

Le funzioni possono anche essere **prive di parametri o ritornare nessun valore** e comunque funzionare, vediamo degli esempi:

[5]:
```
def valore_nullo(): #nessun parametro è richiesto
    ''' ritorna il valore nullo'''
    return 0 # ritorno il valore senza eseguire alcuna azione
```

[6]:
```
zero = valore_nullo() #notare che non ho passato nessun parametro
print('Il valore ritornato dalla funzione valore_nullo è:', zero)
```

```
Il valore ritornato dalla funzione valore_nullo è: 0
```

[7]:
```
def trattini(numero):
    ''' stampera di seguito un numero di trattini
       definito dal valore del parametro passato'''
    print('-'*numero) #notare che non metto un return
```

[8]:
```
trattini(80)
```

```
--------------------------------------------------------------------------------
```

Dobbiamo però notare la seguente cosa, nel caso in cui si associ una variable al risultato di trattini otteniamo il risultato `None`.

[9]:
```
valore = trattini(80)
print('Valore è:', valore)
```

```
--------------------------------------------------------------------------------
Valore è: None
```

Ovviamente l'azione della funzione verrà eseguita e quindi verrano stampati 80 trattini, ma il valore che sarà ritornato sarà `None`, il motivo è dovuto al fatto che se non è presente il **`return`** nella funzione python definirà la funzione con **`return None`** come riga finale. **Nota:**`None` è un tipo di variabile che non contiene nessun valore e può essere tradotto come **nulla**, il suo comportamento è molto importante!

[10]:
```
#funzione senza parametri o return
def saluta(): #nesssun parametro
    ''' stampa Ciao'''
    print('Ciao') #no return
```

[11]:
```
saluta()
```

```
Ciao
```

1.1.4 Default Arguments - Argomenti di default dei parametri

Nelle funzioni in genere potrebbe essere richiesto che qualora non sia specificato nulla si passi un valore di default al suo interno e si eseguino comunque le azioni.

[12]:
```
def default_somma(x = 0, y = 0):
    '''somma di due numeri, qualora
       non siano passati valori i due numeri
       saranno assunti come nulli'''
    tot = x + y
    return tot
```

[13]:
```
#confrontiamo il comportamento delle due funzioni
#caso in cui i valori sono definiti
tot_somma = somma(x = 3, y = 2)
tot_somma_default = default_somma(x = 3, y = 2)
print('Somma di valori definiti')
print('funzione somma:', tot_somma)
print('funzione default somma:', tot_somma_default)
```

```
Somma di valori definiti
funzione somma: 5
funzione default somma: 5
```

[14]:
```
print('Somma senza valori definiti')
print('funzione default somma:', default_somma())
print('funzione somma:',somma())
```

```
Somma senza valori definiti
funzione default somma: 0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-242990a038b2> in <module>
      1 print('Somma senza valori definiti')
      2 print('funzione default somma:', default_somma())
----> 3 print('funzione somma:',somma())

TypeError: somma() missing 2 required positional arguments: 'x' and 'y'
```

Come possiamo notare nel caso della funzione con default argument non ci sono problemi e viene ritornato la somma dei valori di default 0, mentre la funzione senza default argument produce un errore.

1.1.5 Variable length arguments - argomenti di lunghezza variabile

Nel caso in cui noi non fossimo in grado di sapere quanti parametri o argomenti siano passati possiamo passare un parametro che contenga i valori in eccesso. #### Arbitrary arguments - argomenti arbitrari #### In questo caso un arbitrary arguments è passato come un parametro preceduto da * che convertirà i valori in una **tupla**, vediamo il suo utilizzo:

[15]:
```
#*args permette di definire un numero arbitrario di valori
def somma_arbitrary(*args):
    ''' somma su due o più valori'''
    tot = 0 #inizializza tot a 0
    for arg in args: #loop sulla tupla
        #somma sui valori ulteriori
        tot += arg # tot = tot + arg
    return tot
```

[16]:
```
tot_arbitrary = somma_arbitrary(3, 2, 5)
print('Somma di 3, 2 e 5:',tot_arbitrary)
```

```
Somma di 3, 2 e 5: 10
```

Possiamo notare che in questo caso non servono i keyward arguments (*parametro* = *valore*), ma solo una serie di valori in input il che semplifica di molto le cose. #### Arbitrary Keyword arguments - argomenti arbitrari con chiave #### Nel caso in cui noi vogliamo specificare il valore associato a una particolare variabile senza sapere quanti argomenti ci servano è possibile usare un parametro(di solito **kwargs**) preceduto da ** questo ci convertirà gli argomenti passati in **dizionario**.

[17]:
```
def stampa_nomi(**kwargs):
    ''' stampa i nomi delle persone,
       scrivere i nomi in formato Nome:Cognome'''
    #ricorda che kwards è un dizionario
    for name, surname in kwargs.items():
        print('Nome:', name, ' Cognome:', surname)
    #return non necessario
```

[18]:
```
stampa_nomi(Matteo = 'Conterno', Simona = 'Ferrero', Marco = 'Coluccio', Flavio␣
↪= 'Pirazzi')
```

```
Nome: Matteo  Cognome: Conterno
Nome: Simona  Cognome: Ferrero
Nome: Marco  Cognome: Coluccio
Nome: Flavio  Cognome: Pirazzi
```

1.2 Specificare il tipo dei parametri e del return

Poiché le funzioni in python possono accettare **qualsiasi tipo** di variabile come input dei parametri questo potrebbe portare a strani comportamenti delle funzioni, se per esempio alla prima funzione somma passiamo due strighe ecco cosa succede:

[19]:
```
tot = somma(x = 'Ehi', y = 'Ciao')
print('Risultato della somma di Ehi e Ciao:',tot)
```

```
Risultato della somma di Ehi e Ciao: EhiCiao
```

Per specificare il tipo dei parametri si utilizza il carattere `:` subito dopo il nome del parametro seguito dal tipo, mentre per specificare il tipo dell'output si usa la combinazione `->` dopo le parentesi delle funzioni specificando subito dopo di essa il tipo.

[20]:
```
# x:int y:int convertono i parametri in intero
# -> int specifica che l'output deve essere convertito in intero
def somma_tipo(x:int = 0, y:int = 0) -> (int): # notare x:int, y:int
    ''' somma due numeri interi, ritorna un numero intero'''
    tot = x + y
    return tot
```

[21]:
```
tot = somma_tipo(x = 2, y = 3)
print('Somma di 2 e 3:', tot, '\n il totale è di tipo:', type(tot))
```

```
Somma di 2 e 3: 5
 il totale è di tipo: <class 'int'>
```

Notiamo però che questo non impedisce alle variabili di assumere altri tipi di valori ed eseguire il codice restituisce non il valore specificato.

[22]:
```
somma_tipo(x = 'ciao', y = 'dopo')
```

[22]:
```
'ciaodopo'
```

Questo è dovuto al fatto che le funzione secondo la filosofia python devono funzionare con qualsiasi tipo di variabile, ma rimane comunque utile specificare il tipo per capire quali sono le richieste in input, qualora volessimo impedire alcuni operazioni possiamo far creare un **errore** che saranno spiegati in futuro. ## Comando Pass ## Le funzioni non possono essere vuote, ma è possibile crearne senza occorrere in errori usando il comando `pass`, il comando può essere usato anche nel caso in cui particolari condizioni sono soddisfatte e non sia necessario eseguire il blocco di codice contenuto in una parte della funzione.

[23]:
```
def vuota():
    ''' non fa nulla!'''
    pass
```

[24]:
```
vuota()
```

[25]:
```
def positivo(numero):
    ''' non ritorna nulla se il numero è positivo,
       altrimenti dice se è nullo o negativo'''
    if numero > 0: #il numero è positivo
        pass #non fare nulla!
    elif numero < 0:
        print('è negativo!')
    else:
        print('nullo')
```

[26]:
```
positivo(2) # non fa nulla
```

[27]:
```
positivo(-3)
```

```
è negativo!
```

1.3 Funzioni Anonime o Lamba

In certi contesti potrebbe non essere necessario definire delle lunghe funzioni, ma delle brevi funzioni che compiano poche operazioni, in tal caso python permette l'utilizzo di **funzioni lamba o anonime**.

[28]:
```
#questa funzione prenderà come argomento x e restituirà il quadrato di esso
#è equivalente a
#def square(x):
#   return x * x
square = lambda x : x * x #notare la dichiarazione di lambda
```

Per richiamare la funzione basta scrivere il suo nome e mettere tra parentesi un valore ed essa restituirà il valore delle corrispondenti operazioni dopo `:`.

[29]:
```
five = 5
print('Il quadrato di', five, 'è', square(five))
```

```
Il quadrato di 5 è 25
```

In genere le funzioni anonime sono usate in contemporanea alle funzioni `filter` e `map` delle liste per ripulire velocemente i dati e saranno usate anche nei dataframes successivamente. ### Filter e Map ### **Filter** è una funzione che ha come **argomenti un'altra funzione e una lista** per cui il risultato sarà una nuova lista dove i risultati saranno gli elementi che sooddisfano la condizione della funzione.

[32]:
```
lista = [1,2,4,5,6,22, 43, 55]
nuova_lista = list(filter(lambda x: (x%2)!=0, lista))
print('Lista dei soli numeri dispari:',nuova_lista)
```

```
Lista dei soli numeri dispari: [1, 5, 43, 55]
```

Map è una funzione che ha come **argomenti un'altra funzione e una lista** per cui il risultato sarà la nuova lista con gli elementi ottenuti applicando la funzione senza la necessità che ci sia una condizione.

[33]:
```
lista_quadrati = list(map(lambda x: x*x, lista)) #quadrato di ogni elemento␣
 ↪nella lista
print('Lista dei quadrati:',lista_quadrati)
```

```
Lista dei quadrati: [1, 4, 16, 25, 36, 484, 1849, 3025]
```

COMPLIMENTI AVETE FINITO LA LEZIONE SULLE FUNZIONI!