scriviamo-le-prime-funzioni

June 5, 2023

0.1 Scriviamo le prime funzioni

Esercizio: scrivete un programma che stampi l'area di un cerchio per cinque volte, un per ogni raggio nella lista che segue. - 10.5 - 2.4 - 7.5 - 1.0 - 3.24

[1]:
```
raggio = 10.5
area = 3.14 * raggio**2
print("Il cerchio di raggio",raggio,"ha area",area)

# Proseguite voi...
```

```
Il cerchio di raggio 10.5 ha area 346.185
```

Esercizio: adesso modificate il programma per usare l'approssimazione migliore di pi greco, importando il modulo `math` e adoperando `math.pi` al posto di `3.14`.

Se ora vi dicessi che voglio la stampa per altri 5 valori? Altri 10? La prendereste bene? Immagino che abbiate notato subito quanto sia sgradevole dover scrivere lo stesso codice ripetutamente. - È noioso e ripetitivo. - Se si deve cambiare una cosa, bisogna essere certi di farlo su tutte le copie. - Ci sono molte possibilità che una o più copie contengano errori.

Non sarebbe bello poter inventare **una nuova istruzione** che stampa l'area del cerchio, così da poterla utilizzare a piacimento?

0.1.1 Definizione di funzioni

Tutti i linguaggi di programmazione prevedono modi per definire nuove funzioni simili a `len`, `cos`, `sin`, ecc… In questo modo possiamo scrivere del codice **una sola volta** e riutilizzarlo **molte volte**. Oltretutto se vogliamo correggerlo e/o cambiarlo dobbiamo modificare solo **una parte del programma**.

Definiamo una funzione che stampa dell'area del cerchio di raggio 10.5.

[2]:
```
def stampaarea():
    raggio = 10.5
    area = 3.14 * raggio**2
    print("Il cerchio di raggio",raggio,"ha area",area)

print("Questa instruzione non fa parte della funzione.")
stampaarea()
```

```
Questa instruzione non fa parte della funzione.
Il cerchio di raggio 10.5 ha area 346.185
```

Prima di discutere il codice, eseguitelo e osservate cosa succede.
- Se cambiate il raggio cosa succede? - Se alla fine del programma aggiungete una seconda riga `stampaarea()`? - Se invece di `stampaarea()` scrivere `stampaarea`, che succede?

0.1.2 Descrizione dell'esempio precedente

Abbiamo usato l'istruzione `def` per creare una nuova funzione di nome `stampaarea`, che sostanzialmente esegue le tre righe di codice successive ogni volta che viene richiamata. Per richiamarla, basta scrivere `stampaarea()`.

In tutti i programmi che abbiamo visto fino ad ora Python eseguiva le istruzioni dalla prima all'ultima. Eventualmente alcune righe venivano ignorate in caso di costrutti `if elif else`.

In questo caso Python ha trova `def` alla riga 1 e capisce che deve creare una nuova funzione `stampaarea`. Le parentesi dopo il nome della nuova funzione sono necessarie, ma per ora lasciamo perdere il loro significato. L'istruzione `def` termina con `:`, e se ricordate la sintassi di `if elif else`, questo vuol dire che dalla riga successiva deve iniziare un blocco di codice, indentato verso destra. Questo blocco contiene le **istruzioni associate alla funzione**.

Alla riga 7, Python vede che l'indentazione è tornata a sinistra, quindi capisce che il blocco di codice associato alla `def`, e quindi alla funzione `stampaarea`, è terminato. Dalla riga 7 in poi, sarà possibile utilizzare la funzione `stampaarea`.

Per **usare la funzione** dovete usare l'istruzione `stampaarea()`. Fate attenzione, le parentesi sono necessarie.

0.1.3 Funzioni con parametri

Qual è il difetto più ovvio della funzione `stampaarea`? Per me quello più fastidioso è che stampa solo l'area di un cerchio di raggio `10.5`. Se ricordate l'esercizio all'inizio di questa sezione, o semplicemente avete buon senso, vorrete scrivere una funzione che permetta di specificare il raggio ad ogni utilizzo. In effetti è possibile, nel momento in cui si definisce la funzione, specificare i suoi **parametri**.

[4]:
```
def stampaarea(R):
    area = 3.14 * R**2
    print("Il cerchio di raggio", R, "ha area", area)

print("Questa instruzione non fa parte della funzione.")
```

```
Questa instruzione non fa parte della funzione.
```

[5]:
```
stampaarea(10.5)
stampaarea(2.4)
stampaarea(7.5)
stampaarea(1.0)
stampaarea(3.24)
```

```
stampaarea( 1.4 - 0.2 )
```

```
Il cerchio di raggio 10.5 ha area 346.185
Il cerchio di raggio 2.4 ha area 18.0864
Il cerchio di raggio 7.5 ha area 176.625
Il cerchio di raggio 1.0 ha area 3.14
Il cerchio di raggio 3.24 ha area 32.962464000000004
Il cerchio di raggio 1.2 ha area 4.5216
```

Ecco a che servono le parentesi nella `def`. Nelle parentesi possiamo specificare la lista dei parametri della funzione. Quando la funzione viene richiamata, deve ricevere tra parentesi i valori corrispondenti ai paramentri, nello stesso ordine. Nel programma precedente ci sono sei chiamate alla stessa funzione, ogni volta con un parametro diverso.

Esercizio: modificare questa funzione per usare `math.pi`. Non e più semplice farlo, adesso?

0.1.4 Definiziona di funzioni in generale

In generale la definizione di una funzione ha questo aspetto:

```
def nomefunzione(par1,par2,par3):
    istruzione1
    istruzione2
    istruzione3
```

e una volta definita, la stessa funzione può essere richiamata con

[ ]:
```
nomefunzione(expr1,expr2,expr3)
```

In questa descrizione ho considerato tre parametri ma il numero di parametri può essere **zero** o un numero **arbitrario ma fisso**. Nel senso che se definite una funzione con 4 parametri allora tutte le sue chiamate devono avere quattro espressioni tra parentesi altrimenti si ha un errore, come nell'esempio successivo nel quale calcoliamo il massimo di 4 numeri.

[8]:
```
def massimo(n0,n1,n2,n3):
    t = n0

    if t < n1:
        t = n1

    if t < n2:
        t = n2

    if t < n3:
        t = n3

    print("Il massimo è",t)
```

[9]:
```
massimo(0,-4,7,-3)
massimo(4,2,1,3)
```

```
massimo(0,-4,7)
```

```
Il massimo è 7
Il massimo è 4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [9], in <cell line: 3>()
      1 massimo(0,-4,7,-3)
      2 massimo(4,2,1,3)
----> 3 massimo(0,-4,7)

TypeError: massimo() missing 1 required positional argument: 'n3'
```

0.1.5 Funzione che restituisce un valore

Le funzioni viste fino ad ora non producono nessun valore, come succede ad esempio con `print`. Tuttavia funzioni come `len`, `abs` e `sin` producono un valore e possono essere utilizzate all'interno di espressioni.

Con il comando `return` una funzione **restituisce** un valore alla parte di programma nella quale la funzione è stata chiamata. Vediamo un esempio:

[10]:
```
import math

def areacerchio(R):
    area = R*R*math.pi
    return area
```

[11]:
```
totale = areacerchio(2) + areacerchio(5)

print("La superficie dei due dischi di raggio 2 e 5 è in totale", totale)
```

```
La superficie dei due dischi di raggio 2 e 5 è in totale 91.106186954104
```

- Il comando `return` può essere usato solo all'interno di una funzione
- Appena la funzione esegue `return qualcosa`, la sua esecuzione si interrompe immediatamente, e il valore dell'espressione `qualcosa` viene restituita al programma chiamante.
- Ci possono essere molteplici istruzioni `return` in una funzione, ma solo una di esse verrà eseguita quando la funzione viene chiamata.

Vediamo un altro esempio.

[13]:
```
def mezzotesto(testo, inizio):
    """Metà del testo

Restituisce metà del testo passato come parametro. Se
'inizio' è True allora prende la prima metà, altrimenti la seconda."""
```

```
    metà = len(testo) // 2
    if inizio:
        return testo[:metà]
    else:
        return testo[metà:]
```

[14]:
```
print( mezzotesto( 'Programmazione python', True))
print( mezzotesto( 'Programmazione python', False))
```

```
Programmaz
ione python
```

[15]:
```
print( mezzotesto( 'ABCDEF', True))
print( mezzotesto( 'ABCDEF', False))
```

```
ABC
DEF
```

[16]:
```
print( mezzotesto( 'ABCDE', True))
print( mezzotesto( 'ABCDE', False))
```

```
AB
CDE
```

Vedete che in questo caso la funzione è documentata. La documentazione di una funzione deve essere inserita immediatamente dopo la riga che contiene la `def`, viene aperta e chiusa da tre apici `"""`, e tipicamente è formata - una riga con una breve descrizione - una riga vuota - descrizone più ricca - il testo **DENTRO** la documentazione non deve essere necessariamente indentato.

Esercizio: alla fine dell'esempio precedente aggiungere l'istruzione `help(mezzotesto)`.

0.1.6 Il non-valore `None`

Le funzioni che non restituiscono valori in realtà vengono viste da Python come funzioni che restituiscono un valore convenzionale `None`.

[17]:
```
x = print("ciao")
print(x)
print( x is None)
x + 5
```

```
ciao
None
True
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [17], in <cell line: 4>()
      2 print(x)
```

```
      3 print( x is None)
----> 4 x + 5

TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

L'espressione `expr is None` è un'espressione booleana che è vera se e solo se il (non) valore di `expr` è `None`.

Attenzione: se via aspettate che la vostra funzione restituisca un valore e invece ottenete `None`, forse avete gestito male i `return`.

Quando accade che una funzione termina senza restituire un valore? - quando arriva all'ultima istruzione senza aver mai eseguito un `return` - quando esegue `return None` - quando esegue `return` senza nessuna espressione.

Domanda: a che serve usare `return` o `return None` se non si vuole restituire nessun valore. Non basta che la funzione arrivi alla fine delle sue istruzioni?

0.1.7 Soluzione maggiore di un'equazione di 2o grado.

[19]:
```
import math

def secondogrado(A,B,C):
    """Una soluzione di Ax**2 + Bx + C == 0

Data un'equazione di secondo grado Ax**2 + Bx + C == 0,
questa funzione ne restituisce una soluzione, se ne esistono.
    """
    Delta = B**2 - 4*A*C

    if Delta < - 0.0000001:  # tolleranza
        return

    if Delta < 0.0000001:    # tolleranza
        return -B / (2*A)
    else:
        return (-B + math.sqrt(Delta)) / 2*A
```

[20]:
```
def printsecgra(A,B,C):
    polinomio= str(A) + "x**2 + " + str(B) +"x + " + str(C)
    t = secondogrado(A,B,C)

    if t is None:
        print("Il polinomio", polinomio, "non ha soluzione")
    else:
        print("Il polinomio", polinomio, "ha soluzione",t)
```

[21]:
```
printsecgra(2,-7,3)
printsecgra(1,-2,1)
printsecgra(4,3,4)
printsecgra(7,-7,4)
```

```
Il polinomio 2x**2 + -7x + 3 ha soluzione 12.0
Il polinomio 1x**2 + -2x + 1 ha soluzione 1.0
Il polinomio 4x**2 + 3x + 4 non ha soluzione
Il polinomio 7x**2 + -7x + 4 non ha soluzione
```

Notate che nel programma: - abbiamo definito due funzioni, e addirittura una richiama l'altra; - abbiamo usato `math.sqrt` per calcolare la radice quadrata; - la funzione usa `return` in diversi punti del codice; - a volte la funzione restituisce un `float`, a volte non restituisce nulla; - si poteva usare `elif Delta==0:`. Avrebbe fatto differenza?

0.1.8 Riassumendo

Abbiamo visto - a che servono le funzioni - definizione di nuove funzioni - uso dei parametri - uso del **return** per restituire un valore - funzioni che **non restituiscono** valori - documentazione di una funzione