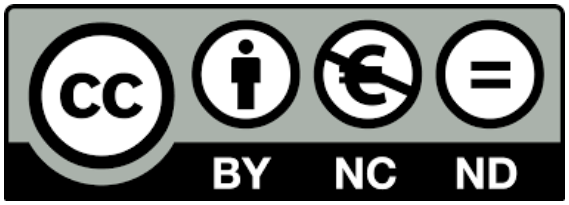
Introduzione alla programmazione

ITS - Umbria

A.S. 2022-23

MATERIA: fondamenti di informatica

Docente: prof. Paolo Bernardi



Funzioni: cosa sono

In Python le funzioni sono oggetti di base (first-class objects); l'istruzione def le crea ed assegna loro un nome, che è un riferimento. Al solito si usano i due punti ed il rientro (indentation) per delimitare il corpo della funzione. Gli argomenti, fra parentesi tonde, seguono il nome della funzione; alla chiamata la funzione viene eseguita, producendo un oggetto, che è il risultato della funzione e che viene restituito con un'istruzione return. In caso l'istruzione return non compaia nella funzione, viene restituito l'oggetto vuoto: None.

Funzioni: definizione

La sintassi per la definizione di una funzione è del tipo:

```
def nomefunzione(a,b,c):
    '''  docstring:
    descrizione funzione
    '''
    d=a
    e=b+c
    return d+e
```

Funzioni: chiamata

Per chiamare la funzione si utilizza una sintassi del tipo:

```
nomefunzione(r,s,t)
g=nomefunzione(r,s,t)
```

Nel secondo caso il risultato della funzione e' assegnato alla variabile 'g'

Funzioni: nome

Il nome della funzione è semplicemente un riferimento alla funzione; si distingue dalla chiamata alla funzione, ove devono apparire le parentesi tonde dopo il nome.

Il nome può stare in una lista, essere passato come argomento ad altre funzioni, e le funzioni possono anche essere chiavi di dizionari.

Solo le parentesi tonde dopo il nome indicano che la funzione va eseguita.

Funzioni: nome

Per vedere se un nome è un riferimento ad una funzione, si può usare la funzione callable(nome), che restituisce True se l'oggetto cui ci si riferisce è una funzione.

Funzioni: meccanismi di astrazione

A rigor di termini, le funzioni non sono elementi indispensabili. È possibile costruire qualsiasi algoritmo utilizzando solamente operatori interni e strutture di controllo.

Tuttavia, in ogni programma di un certo rilievo, il codice risulterebbe estremamente complesso, difficile da controllare e pressoché impossibile da mantenere.

Funzioni: meccanismi di astrazione

Il problema è che il cervello umano può concentrarsi solo su pochi elementi alla volta. Gli psicologi parlano di tre elementi senza difficoltà (fino a un massimo di sette).

La complessità viene affrontata sviluppando meccanismi per semplificarla o nasconderla. Questi meccanismi vengono chiamati astrazioni. In parole povere, l'astrazione nasconde dei dettagli consentendo così di trattare più elementi come se fossero uno solo.

Funzioni: meccanismi di astrazione

Funzioni per eliminare le ridondanze

Il primo modo nel quale le funzioni servono da meccanismo di astrazione è l'eliminazione del codice ridondante o ripetitivo.

Il codice ridondante è svantaggioso per diverse ragioni. Per prima cosa, costringe il programmatore a scrivere o copiare ripetutamente lo stesso codice e verificarne ogni volta la correttezza. In seguito, se il programmatore decidesse di migliorare l'algoritmo per aggiungere nuove caratteristiche o per renderlo più efficiente, dovrebbe modificare ogni istanza del codice ridondante nell'intero programma.

Funzioni: meccanismi di astrazione

Funzioni per eliminare le ridondanze

Facendo riferimento alla definizione di una singola funzione, al posto di multiple istanze di codice ridondante, il programmatore può scrivere un singolo algoritmo in un unico luogo, per esempio in un modulo di libreria. Qualsiasi altro modulo o programma può quindi importare la funzione e utilizzarla. Una volta importata, la funzione può essere chiamata ogniqualvolta si renda necessario. Quando il programmatore avrà bisogno di correggere o migliorare la funzione, sarà sufficiente modificare e verificare solamente la definizione della funzione. Non sarà necessario agire sulla parti del programma dove la funzione viene chiamata.

Funzioni: meccanismi di astrazione

Funzioni per nascondere la complessità

Un altro modo in cui le funzioni servono da meccanismo di astrazione è quello di nascondere i dettagli complessi.

Una chiamata di funzione indica al programmatore l'idea che sta dietro al processo, senza costringerlo a farsi strada attraverso il complesso codice che realizza quell'idea.

Una sorta di scatola nera, della quale si conosce a priori e deterministicamente il comportamento, ma non necessariamente l'implementazione, che a determinati stimoli in ingresso risponde sempre con gli stessi valori in uscita.

Funzioni: meccanismi di astrazione

Metodo generali e variazioni sistematiche

Un algoritmo è un metodo generale per risolvere una determinata classe di problemi. Gli specifici problemi che compongono una classe di problemi sono noti come istanze del problema.

Le istanze del problema per un dato algoritmo possono variare da un programma all'altro, e persino tra le differenti parti di uno stesso programma.

Funzioni: meccanismi di astrazione

Metodo generali e variazioni sistematiche

Quando si progetta un algoritmo, questo dovrebbe essere sufficientemente generale da essere utilizzato nel maggior numero possibile di istanze del problema, non solamente per una o per alcune di esse.

In altre parole, una funzione dovrebbe fornire un metodo generale per le variazioni sistematiche.

Funzioni: meccanismi di astrazione

Suddivisione del lavoro

In un programma le funzioni possono aiutare a suddividere il lavoro da svolgere. Idealmente, ciascuna funzione svolge un singolo compito, per esempio calcolare un integrale o formattare una tabella di dati da visualizzare Ogni funzione ha il compito di utilizzare certi dati, di calcolare determinati risultati e di restituirli alle parti del programma che li hanno richiesti.

Funzioni: argomenti

Gli argomenti sono passati per **assegnazione**: in Python le **variabili sono riferimenti ad oggetti**.

Nel passaggio degli argomenti, alla variabile nella funzione viene assegnato lo stesso oggetto della variabile corrispondente nella chiamata; cioè, viene fatta una copia dei riferimenti agli oggetti.

Funzioni: argomenti

I tipi delle variabili vengono definiti solo all'assegnazione dei riferimenti, per cui una funzione, a priori, non sa quali sono i tipi degli argomenti ed eventuali inconsistenze producono errori solo quando la funzione viene eseguita.

In questo modo Python implementa naturalmente il polimorfismo, cioè una stessa funzione può essere usata per dati di tipo diverso.

Funzioni: argomenti

Ad esempio la funzione:

```
def somma(a,b):
    return a+b
```

- se chiamata con: somma(3,2) produrrà 5
- se chiamata con stringhe come argomenti: somma('aa', 'bb') restituirà la stringa 'aabb'.
- se chiamata come: somma('a', 1) produrrà un errore.

Funzioni: argomenti

scratch.py | prova01.py

```
1   def cambia1(x):
2       for y in x:
3           if y % 2 == 0:
4               y = 0
5           else:
6               y = 1
7
8
9   def cambia2(x):
10      for i in range(len(x)):
11          if x[i] % 2 == 0:
12              x[i] = 0
13          else:
14              x[i] = 1
15
16
17  a = [x for x in range(1, 21)]
18  print("a =", a)
19  cambia1(a)
20  print("cambia1(a) =", a)
21  cambia2(a)
22  print("cambia2(a) =", a)
```

ATTENZIONE: riassegnare le variabili in argomento entro la funzione non ha effetti sul chiamante, ma gli oggetti mutabili possono essere cambiati nelle funzioni, operando sui loro riferimenti.

Funzioni: argomenti

ATTENZIONE: riassegnare le variabili in argomento entro la funzione non ha effetti sul chiamante, ma gli oggetti mutabili possono essere cambiati nelle funzioni, operando sui loro riferimenti.

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
cambia1(a) = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
cambia2(a) = [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]

Process finished with exit code 0
```

Funzioni: argomenti

Le funzioni possono avere valori di default per gli argomenti.

Ad esempio una funzione definita con:

```
def func(a = 'uno'):
```

può essere chiamata semplicemente con:

```
func()
```

ed il suo argomento a sarà il default: la stringa 'uno'

oppure con

```
func('due')
```

ed il suo argomento a sarà la stringa 'due'

Funzioni: argomenti

Una funzione può anche essere chiamata dando valori ai parametri per nome (keyword arguments), con una sintassi tipo:

```
func(a = 'sei')
```

in questo caso alla variabile a entro la funzione, viene assegnata la stringa "sei".

Una funzione può essere definita in modo che i suoi argomenti siano visti, entro la funzione, come una tupla o come un dizionario; le definizioni della funzione avranno in questi casi rispettivamente la sintassi:

```
def func(*nome):
def func(**nome):
```

Nel caso del dizionario gli argomenti sono passati per nome ed i nomi diventano le chiavi del dizionario.

Funzioni: argomenti

Questi modo di passare gli argomenti possono essere combinati, in questo caso, nelle chiamate e nella funzione, vanno passati in questo ordine:

- Gli argomenti posizionali
- Gli argomenti che finiscono in una tupla
- Gli argomenti che finiscono nel dizionario (con passaggio per nome)

Funzioni: argomenti

```
def func(a,b,c):  # esempio di funzione
    print(a)
    print(b)
    print(c)


func(1,2,3)            # chiamata con argomenti passati per posizione
func( b=2,a=1,c=3)     # argomenti passati per nome
func(1,c=3,b=2)        # argomenti passati per posizione, quello che resta per nome

def func(*a):
      print(a)
      '''
      Qui in a finisce una tupla di argomenti,
      la chiamata puo' avere numero variabile di argomenti
      func(1,2,3) stampa la tupla: (1,2,3)
      '''
```

Funzioni: argomenti

```
def func(**d):
      print(d)
      '''
      Qui gli argomenti finiscono in un dizionario
      gli argomenti sono passati per nome
      ed i nomi delle variabili sono le chiavi
      func(a=1,b=2,c=3) stampa:   {'a': 1, 'c': 3, 'b': 2}
      '''


def func(a,*b,**d):
      print(a)
      print(b)
      print(d)
      '''
      Vanno prima gli argomenti posizionali,
      poi quelli per la tupla, infine quelli per il dizionario.
      Chiamata come :  func(1, 2,3,4, s=10,q=20 )
      stampa:  a=1  ;  b=[2,3,4] ; d={s:10,q:20}
      '''
```

Funzioni: argomenti

In Python3 abbiamo anche funzioni con argomenti passati per nome dopo quelli che finiscono in una lista

```
def func(a,*b,c):

Qui l'ultimo argomento puo' essere dato solo per nome
con chiamata:   func(1,2, 3, c=40) ,
ed avremo a==1 ; b==[2,3] ; c==40
```

Funzioni: Valori restituiti

La funzione restituisce un valore specificato nell'istruzione **return**.

Se non viene eseguita l'istruzione **return** il valore restituito dalla funzione è l'oggetto speciale di nome **None** che è per definizione un **oggetto vuoto**.

Una funzione può anche restituire una **tupla**, con la sintassi del tipo: ***return a, b, c***

Funzioni: Valori restituiti

scratch.py | prova01.py | prova02.py

```
1   def somma(x):
2       y = sum(x)
3       return y
4
5
6   b = [a for a in range(1, 21)]
7   print("Valore restituito :", somma(b))
8
```

Esempio di funzione con return

```
Valore restituito : 210

Process finished with exit code 0
```

Funzioni: Valori restituiti

scratch.py | prova01.py | prova02.py

```
1  def stampa(x):
2      print(x)
3
4
5  b = [a for a in range(1, 21)]
6  stampa(b)
7
```

Esempio di funzione senza return

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

Process finished with exit code 0
```

Funzioni: Campo di validità della funzione (scope della funzione)

Una funzione è valida dal punto del programma in cui si incontra in poi; quando la funzione viene incontrata viene "eseguita", nel senso che il suo nome (che in realtà è un riferimento) diviene valido e ad esso sono associate le operazioni contenute nel corpo della funzione. In questo modo è possibile definire una funzione in modo diverso a seconda del flusso del programma. Una funzione può essere definita all'interno di un'altra funzione: in questo caso essa è visibile solo all'interno della funzione contenitrice e viene deferenziata quando si esce dalla funzione che l'ha generata.

Funzioni: Campo di validità della funzione (scope della funzione)

```
if a>b:
    def func(a,b):
        return a-b
else:
    def func(a,b):
        return b-a
```

Queste istruzioni definiscono la funzione func come la differenza fra il più grande dei due valori in a e b.

A seconda dei casi la funzione è definita in modo diverso.

Funzioni: Campo di validità delle variabili (scope delle variabili)

scratch.py | prova01.py | prova02.py

```
1  def somma(x):
2      y = sum(x)
3      return y
4
5
6  x = [a for a in range(1, 21)]
7  y = "abc"
8  print("Valore restituito :", somma(x))
9
```

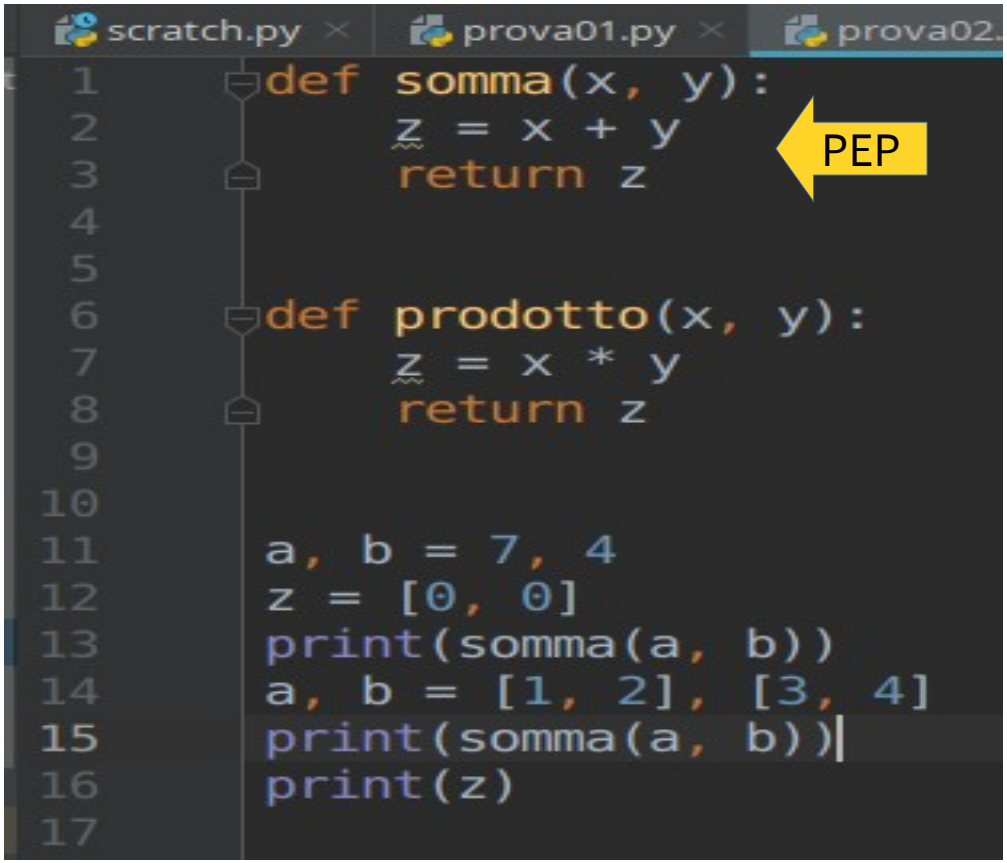
PEP

```
Valore restituito : 210

Process finished with exit code 0
```

Una variabile definita in una funzione non è visibile al di fuori della funzione. Può avere lo stesso nome di una variabile esterna senza confusione.

Funzioni: Campo di validità delle variabili (scope delle variabili)



Una variabile definita nel blocco in cui la funzione è chiamata è vista entro la funzione, ma non può essere modificata entro la funzione, a meno che non sia definita "global" entro la funzione.

Funzioni: Campo di validità delle variabili (scope delle variabili)

scratch.py | prova01.py | prova02.py

```
1   def somma(x, y):
2       z = x + y
3       return z
4
5
6   def prodotto(x, y):
7       z = x * y
8       return z
9
10
11  a, b, = 7, 4
12  print(somma(a, b))
13  a, b = [1, 2], [3, 4]
14  print(somma(a, b))
15
```

CORRETTO

```
11
[1, 2, 3, 4]

Process finished with exit code 0
```

Funzioni: Campo di validità delle variabili (scope delle variabili)

scratch.py | prova01.py | prova02.py

```
1   def somma(x, y):
2       z = x + y
3       return z
4
5
6   a, b, z = 7, 4, "ciao"
7   print(somma(a, b))
8   print(z)
9   a, b, z = [1, 2], [3, 4], "ok"
10  print(somma(a, b))
11  print(z)
12
```

```
11
ciao
[1, 2, 3, 4]
ok

Process finished with exit code 0
```

CORRETTO

- **Shadow variable**
- **Notare i valori di z**

Funzioni: Campo di validità delle variabili (scope delle variabili)

scratch.py | prova01.py | prova02.py

```
1   def somma(x, y):
2       global z
3       z = x + y
4       return z
5
6
7   a, b, z = 7, 4, "ciao"
8   print(somma(a, b))
9   print(z)
10  a, b, z = [1, 2], [3, 4], "ok"
11  print(somma(a, b))
12  print(z)
13
```

```
11
11
[1, 2, 3, 4]
[1, 2, 3, 4]

Process finished with exit code 0
```

CORRETTO

- **Z è global**
- **Notare i valori di z**

Funzioni: funzioni di ordine superiore

Le funzioni di ordine superiore sono delle funzioni che ricevono in ingresso funzioni e parametri, secondo la sintassi

$$f(f_1 [, f_2, f_3, \ldots], x_1 [, x_2, x_3, \ldots])$$

con f_1, f_2, f_3, ... un insieme di funzioni

con x_1, x_2, x_3, ... un insieme di parametri

e restituiscono in uscita un vettore di valori y_1, y_2, …

che dipendono dai valori x secondo le trasformazioni f

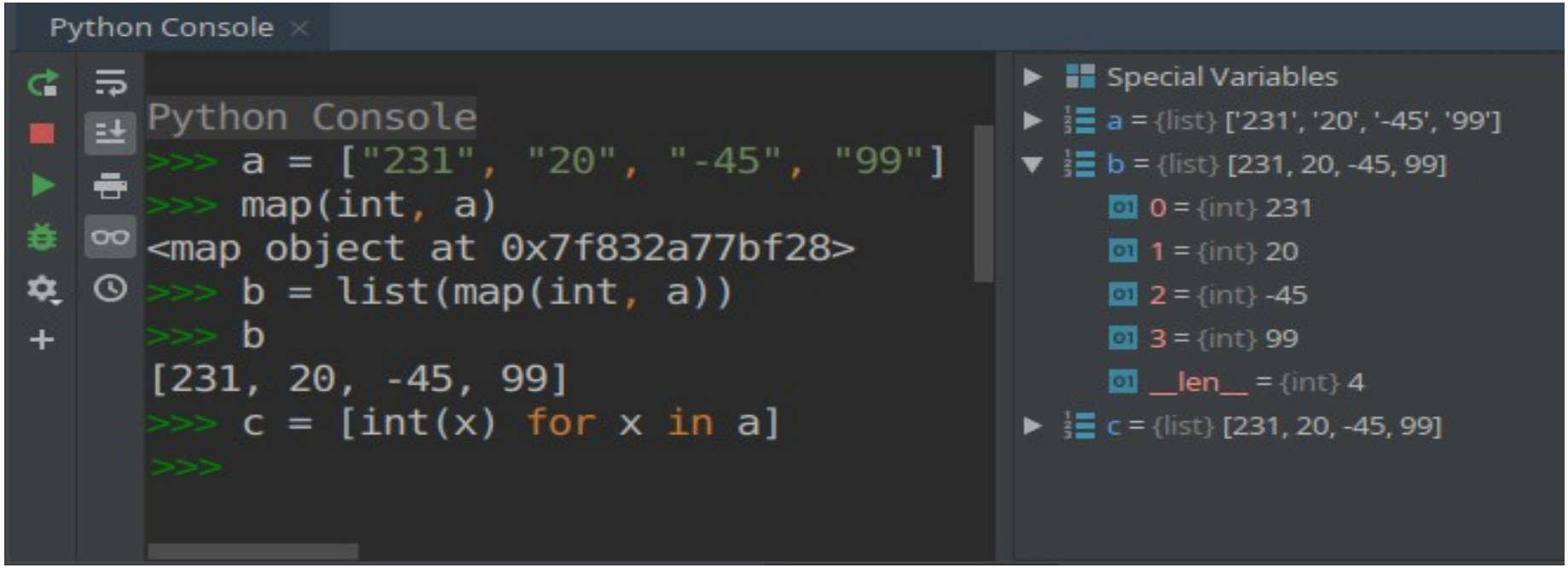
Funzioni: funzioni di mappatura (map)

Il primo tipo utile di funzione di ordine superiore da considerare si chiama **mappatura**.

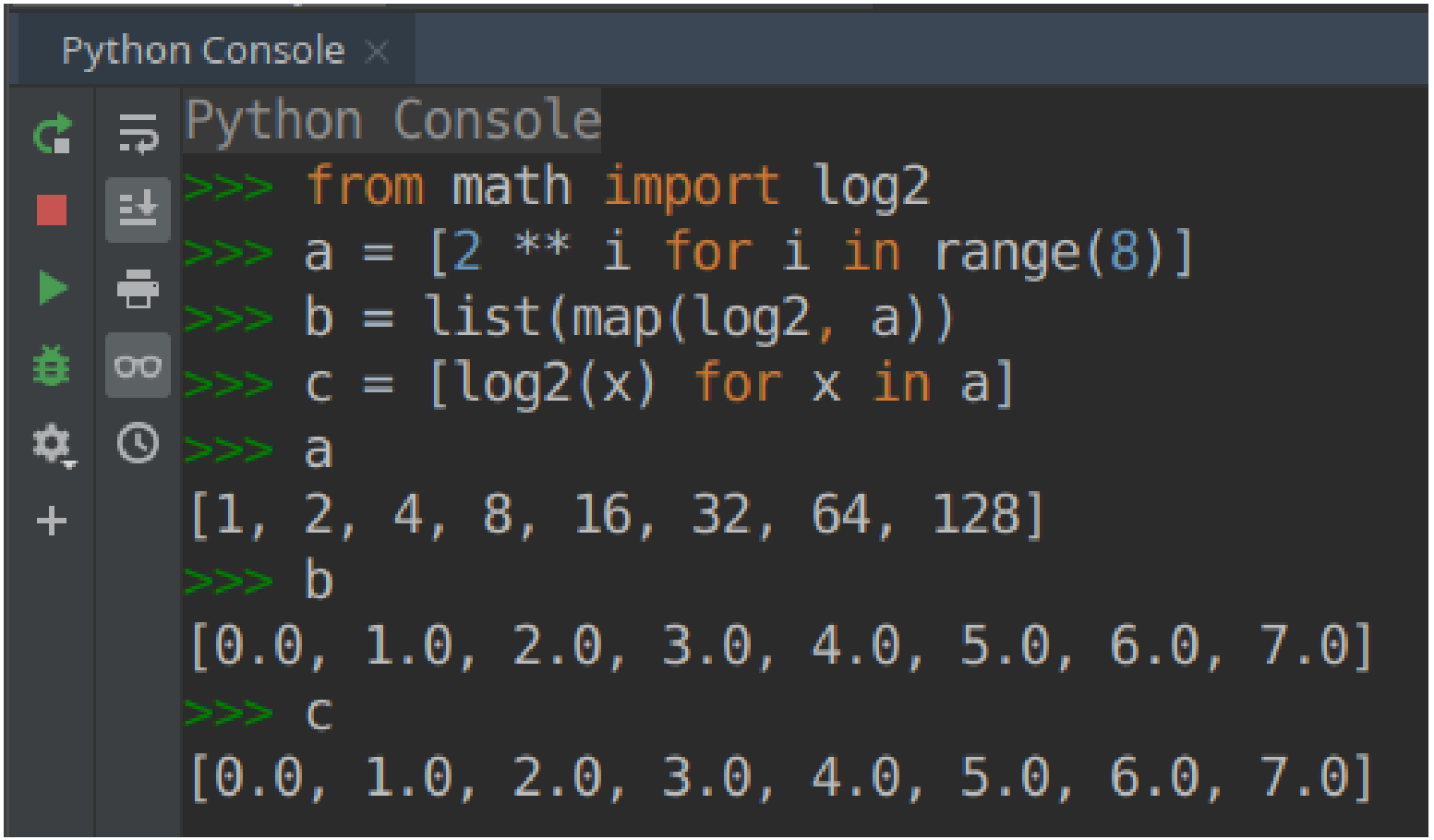
Questo processo applica una funzione a ciascun valore in una sequenza (come una lista, una tupla o una stringa) e restituisce una nuova sequenza di risultati. Python include la funzione **map** per questo compito.

Funzioni: funzioni di mappatura (map)

Supponiamo di avere una lista chiamata a che contiene stringhe che rappresentano numeri interi e vogliamo una nuova lista con i relativi interi.



Funzioni: funzioni di mappatura (map)



Funzioni: funzioni di filtraggio (filter)

Un secondo tipo di funzione di ordine superiore è detto **filtraggio**. In questo processo, una funzione chiamata predicato viene applicata a ciascun valore di una lista. Se il predicato restituisce **True**, il valore passa il test e viene aggiunto a un oggetto filtro (simile a un oggetto mappa), altrimenti il valore non viene preso in considerazione.

Funzioni: funzioni di filtraggio (filter)

```
Python Console
>>> from random import randint
>>> def odd(n):
...     return n % 2
...
>>> a = [randint(1, 1000) for i in range(10)]
>>> a
[630, 497, 552, 194, 999, 161, 950, 848, 259, 221]
>>> b = filter(odd, a)
>>> b
<filter object at 0x7f2837a0bb38>
>>> b = list(filter(odd, a))
>>> b
[497, 999, 161, 259, 221]
>>> c = [x for x in a if x % 2 == 1]
>>> c
[497, 999, 161, 259, 221]
```

Supponiamo di avere una lista chiamata a che contiene numeri interi e dalla quale volgiamo filtrare ed estrarre solo i numeri dispari.

Funzioni: funzioni di riduzione (reduce)

L'ultimo esempio di funzione di ordine superiore si chiama **riduzione**.

In questo caso, abbiamo una lista di valori ai quali applichiamo ripetutamente una funzione per ottenere un singolo valore.

Una sommatoria è un tipico esempio di questo processo. Il primo valore viene aggiunto al secondo, la somma risultante viene aggiunta al terzo valore è così via fino a ottenere la somma totale di tutti i valori.

Funzioni: funzioni di riduzione (reduce)

Il modulo **functools** di Python include la funzione **reduce** che richiede una funzione di due argomenti e una lista di valori. La funzione **reduce** restituisce il risultato dell'applicazione della funzione reiterata su ogni singolo elemento della sequenza.

Funzioni: funzioni di riduzione (reduce)

```
Python Console
>>> from functools import reduce
>>> def multiply(x, y):
...     return x * y
...
>>> a = [1, 2, 3, 4]
>>> reduce(multiply, a)
24
```

Funzioni: funzioni anonime (lambda)

Sebbene le funzioni di ordine superiore possano semplificare notevolmente il codice, spesso è oneroso definire nuove funzioni da passare come argomenti alle funzioni di ordine superiore. Python include un meccanismo che consente ai programmatori di creare funzioni nel modo appena descritto: **lambda** è una funzione anonima; non ha un nome, ma contiene i nomi dei suoi argomenti e un'espressione. Quando lambda viene applicata ai suoi argomenti, la sua espressione viene valutata e il valore risultante viene restituito.

Funzioni: funzioni anonime (lambda)

Sono funzioni di una sola istruzione, senza nome, con sintassi:

lambda argomento1, argomento2, argomento3: *espressione*

```
f= lambda x,y : x+y

f(2,3)      produce 5
```

```
L=[(lambda  x: x+x), (lambda x: x*x)]

for f in L :
  print(f(3))       # Produce : 6 , 9
```

Funzioni: funzioni anonime (lambda)

Altro esempio:

```
Python Console
>>> f = lambda x, y: x if x >= y else y
>>> f(3, 2)
3
>>> f(3, 5)
5

>>>
```

Funzioni: esempio

```
Python Console
>>> from functools import reduce
>>> f = lambda x, y: x if x >= y else y
>>> a = [1, 2, 3, 4, -1, 5, 0]
>>> reduce(f, a)
5

>>>
```

Alcuni esercizi

1) Utilizzare la list comprehension per creare la lista dei primi dieci quadrati:
   a = [0, 1, 4, 9…]; si creino, successivamente, due nuove funzioni che accettino in input la lista precedente e restituiscano in output due tuple, secondo le seguenti specifiche:
   - solo numeri pari, utilizzando internamente la funzione **filter**
   - la radice quadrata, utilizzando internamente la funzione **map**
2) Realizzazione di una funzione che ricevuto in ingresso una tupla di numeri interi ne calcoli lo **MCD** e lo restituisca
3) Realizzazione di una funzione che ricevuto in ingresso una tupla di numeri interi ne calcoli lo **mcm** e lo restituisca
4) Realizzazione di una funzione che ricevuto in ingresso un numero intero restituisca una **tupla contenente i suoi divisori**

Alcuni esercizi

5) Dal sito https://it.lipsum.com si scarichino i primi 5 paragrafi di Lorem Ipsum, salvandoli su un file di testo chiamato "lorem.txt". Data una stringa da tastiera (parola), si apra il file e si contino le occorrenze della parola nel file, stampando a video il risultato.
6) Si generi un dizionario che abbia come chiave i primi 50 numeri e per elemento una tupla dei suoi divisori. Si stampi a video il dizionario. Successivamente di memorizzi il dizionario in un file chiamato "divisori.txt"

Il linguaggio Python (ver. 3)

Bibliografia

- **Marco Beri, Programmare in Python, Apogeo**
- **Kenneth A. Lambert, Programmazione in Python, Apogeo**
- **Python Software Foundation, https://docs.python.org/3/**
- **Python - sito ufficiale della comunità italiana, www.python.it/doc/**
- **W3C programming language tutorials, www.w3schools.com/python/**
- **“The Hitchhiker’s Guide to Python!”, Kenneth Reitz, Tanya Schlusser, 2018,**
  - https://media.readthedocs.org/pdf/python-guide/latest/python-guide.pdf
- **www.tutorialspoint.com**
  - www.tutorialspoint.com/python3/index.htm

Bibliografia

Software (free license)

- Interprete Python3: www.python.org/downloads/
- IDE: PyCharm (community): www.jetbrains.com/pycharm/download/

Immagini (free license)

- Fonte: www.duckduckgo.com