Programmazione in Python

Funzioni

Dario Pescini[1]

Università degli Studi di Milano-Bicocca

Dipartimento di Statistica e Metodi Quantitativi

dario.pescini@unimib.it

Creazione nuovi operatori: funzioni

L'uso delle funzioni permette di:

- evitare di riscrivere lo stesso codice più volte
- incapsulare porzioni complesse di codice per aumentare la leggibilità dello stesso
- applicare un approccio divide et impera
- aumentare il riutilizzo del codice in programmi diversi
- ...

Funzioni: idea di fondo

Operatore somma

- $(\mathbb{R}, \mathbb{R}) \to \mathbb{R}$
- dominio $\to$ codominio
- $c = a + b$
- 7 = 5 + 2

Gli elementi necessari sono:

- una definizione non ambigua tramite una rappresentazione astratta
- la sua applicazione a valori concreti della definizione restituisce un valore concreto

Funzioni: definizione

Ogni funzione in Python deve essere definita come segue:

```
def nomeDellaFunzione( lista_parametri ):

    parte esecutiva
    return valore
```

Funzione

- `def` identificatore della definizione di una funzione
- `nomeDellaFunzione` identificatore della funzione
- `lista_parametri` lista dei parametri separati da virgola
  `par1, par2, …`
- `parte esecutiva locale` obbedisce alle stesse regole di quella del programma principale, ma le variabili sono locali se non altrimenti definito
- `return valore` restituisce al programma chiamante il valore (se presente) e termina l'esecuzione della funzione

Funzioni: definizione

```
def nomeDellaFunzione( lista_parametri ):

    parte esecutiva
    return valore
```

lista_parametri

- La lista dei parametri fornita nell'intestazione serve all'interprete per definire i parametri che verranno passati alla funzione nella chiamata. Lista parametri formali.
- É una lista di elementi separati da virgola.
- Tramite `nomeParametro` la funzione accederà al valore ad esso associato.
- L'associazione `nomeParametro` valore avviene solo in fase di chiamata.

Funzioni: definizione

```
def nomeDellaFunzione( lista_parametri ):

    parte esecutiva
    return valore
```

parte esecutiva

- segue le stesse regole di quella della parte principale del programma.
- le variabili qui definite sono accessibili solo localmente. Variabili locali.

Funzioni: definizione - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo
```

Funzioni: definizione - Esempio

```
def max(a, b):

    if a > b:
        return a
    else:
        return b
```

Funzioni: chiamata

Una funzione in Python viene utilizzata tramite una chiamata nel codice principale nel seguente modo:

```
nomeDellaFunzione( lista_valori )
```

Funzione

- `lista_valori` è la lista di valori da passare alla funzione separati da virgola. lista parametri attuali
- L'associazione tra parametri formali e attuali avviene in maniera posizionale.
- parametri formali e attuali devono essere in egual numero.

```
max(x, y)
```

Funzioni: chiamata - Esempio

```
def max( a, b):

    if a > b:
        return a
    else:
        return b

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print max(0,100)
```

Funzioni: esecuzione

Prima di eseguire le istruzioni di una funzione:

1. L'elaboratore riserva memoria per ogni parametro formale della funzione.
2. Le espressioni corrispondenti ai parametri attuali vengono valutate.
3. I valori ottenuti dalla valutazione dei parametri attuali vengono assegnati ai corrispondenti parametri formali.
4. Viene eseguita la parte esecutiva della funzione.
5. l'istruzione `return` copia il valore nella memoria ad esso riservata (visibile dal chiamante) e restituisce il controllo al programma chiamante.

Il passaggio di parametri avviene per riferimento all'oggetto!

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5 5	-2				

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5
5	-2	5			

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5
5	-2	5			

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5
5	-2	5			

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5
5	-2	5			

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5
5	-2	5			
5	-2	5	0	100	
5	-2	5	0	100	

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5
5	-2	5			
5	-2	5	0	100	
5	-2	5	0	100	

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5
5	-2	5			
5	-2	5	0	100	
5	-2	5	0	100	

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5
5	-2	5			
5	-2	5	0	100	
5	-2	5	0	100	100

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5
5	-2	5			
5	-2	5	0	100	
5	-2	5	0	100	100

Funzioni: chiamata - Esempio

```
def max(a, b):

    if a > b:
        massimo = a
    else:
        massimo = b

    return massimo

alpha = 5
beta = -2

gamma = max(alpha, beta)
print gamma
print alpha, beta
print max(0,100)
```

main			max		
alpha	beta	gamma	a	b	massimo
5					
5	-2				
5	-2		5	-2	
5	-2		5	-2	5
5	-2	5			
5	-2	5	0	100	
5	-2	5	0	100	100

Funzioni: visibilità variabili

Le variabili definite all'interno di una funzione sono locali!

```
def main():
    x = 3
    f()

def f():
    print(x)

main()
```

genera un errore

Log Errore

```
dario@vulcano: ipython variabileLocaleWrong.py
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/Users/dario/ownCloud/Works/Didattica/Lezioni/Informatica/Esercizi/variabileLocaleWrong.py in <module>()
      6     print(x)
      7
----> 8 main()
      9

/Users/dario/ownCloud/Works/Didattica/Lezioni/Informatica/Esercizi/variabileLocaleWrong.py in main()
      1 def main():
      2     x = 3
----> 3     f()
      4
      5 def f():

/Users/dario/ownCloud/Works/Didattica/Lezioni/Informatica/Esercizi/variabileLocaleWrong.py in f()
      4
      5 def f():
----> 6     print(x)
      7
      8 main()

NameError: global name 'x' is not defined
dario@vulcano: _
```

Funzioni: visibilità variabili

corretto

```
def main():
    x = 3
    f(x)

def f(x):
    print(x)

main()
```

```
dario@vulcano: ipython variabileLocale.py
3
dario@vulcano: _
```

Funzioni: visibilità variabili

variante

```
def main():
    x = 3
    f(x)
    print x

def f(x):
    x = 7
    print(x)

main()
```

Funzioni: visibilità variabili

variante

```
def main():
    x = 3
    f(x)
    print x

def f(x):
    x = 7
    print(x)

main()
```

```
dario@vulcano: ipython funzioneMax3.py
7
3
dario@vulcano:
```

Built-in Functions.

Esiste un insieme di funzioni già definite nel linguaggio:

abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

Utilità Funzioni: esempio

Costruire un programma che calcoli il coefficiente binomiale

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Utilità Funzioni: esempio

Costruire un programma che calcoli il coefficiente binomiale

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Esempio

Prima costruisco una funzione che mi calcoli il valore di $n!$:

$$n! = n\,(n-1)\,(n-2)\,\ldots\,2\,1$$

Esercizio - fattoriale

```
n = input("inserire il valore di n ")

i = n
fattoriale = 1
while i > 1:
    fattoriale = fattoriale * i
    i = i - 1

print "\n %s! = " % n, fattoriale
```

Esercizio - binomiale

```
def fattoriale(numero):
    i = numero
    prodotto = 1
    while i > 1:
        prodotto = prodotto * i
        i = i - 1
    return prodotto


print "inserire i valori n e k"
n = input("n ")
k = input("k ")

binomiale = fattoriale(n) / (fattoriale(k) * fattoriale(n-k))

print "il coefficiente binomiale è ", binomiale
```

Esercizio - binomiale

```
import math as m

print "inserire i valori n e k"
n = input("n ")
k = input("k ")

binomiale = m.factorial(n) / (m.factorial(k) *
    m.factorial(n-k))

print "il coefficiente binomiale è ", binomiale
```

Esercizio - binomiale

```
import math as m

print "inserire i valori n e k"
n = input("n ")
k = input("k ")

binomiale = m.factorial(n) / (m.factorial(k) *
    m.factorial(n-k))

print "il coefficiente binomiale è ", binomiale
```

Modulo math

Moduli

Sono file che contengono definizioni di funzioni.

fibonacci.py

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a + b

def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a + b
    return result
```

Moduli

Servono per

- l'archiviazione permanente di funzioni
- importare funzioni (funzionalità) nel main attuale o in altri moduli
- facilitare il riutilizzo del codice
- semplificare la scrittura di programmi complessi

Moduli: uso

```
import fibonacci

...

fibonacci.fib2(100)
```

import

- `import` identificatore per il caricamento del modulo
- `fibonacci` nome del modulo. Corrisponde al nome del file senza estensione `.py`
- `fibonacci.` nome del modulo per la chiamata delle funzioni
- `fib2(100)` identificatore della funzione e parametro per chiamata della funzione

Moduli: uso

```
import fibonacci as fibo

...

fibo.fib2(100)
```

`import` … `as`

- `as` comando per definizione alias
- `fibo` identificatore alias
- `fibo.` chiamata tramite nome alternativo modulo

Moduli: uso

```
from fibonacci import fib2

...

fib2(100)
```

`from … import`

- `from` comando per indicare il modulo da cui caricare le funzioni
- `import` comando per caricare la funzione
- `fib2` identificatore funzione da caricare
- `fib2(100)` chiamata della funzione (manca nome del modulo).

É da preferire il metodo precedente perché comunque viene processato l'intero modulo e l'uso del solo nome della funzione riduce la chiarezza del codice.

modulo math

- `isinf(x)`
- `isnan(x)`
- `ceil(x)`
- `floor(x)`
- `fabs(x)`
- `factorial(n)`
- `fsum(iterabile)`
- `exp(x)`
- `log(x[, base])`
- `log10(x)`
- `pow(x, y)`
- `sqrt(x)`
- `https://docs.python.org/2.7/library/math.html`

Terminologia: moduli, package e funzioni

- Un `modulo` è un file che contiene una raccolta di funzioni pronte per l'uso
- `Package`: per svolgere compiti complessi può essere necessario sviluppare più moduli che lavorano in sinergia
- Il programma di installazione di python, oltre ad installare l'interprete, installa anche la `libreria standard` di python, un insieme di package e moduli (pronti per essere utilizzati), ognuno dedicato ad un compito specifico
- La libreria standard di Python è molto estesa
  - operazioni matematiche
  - gestione di oggetti grafici
  - operazioni sul file system
  - …
- Se avete bisogno di un modulo non presente nella libreria standard, dovrete recuperarlo e importarlo nel vostro computer (vedremo più avanti come si fa)

Libreria standard di Python

- Per poter utilizzare le funzioni di un modulo dobbiamo dire all'interprete di caricare il modulo in memoria. Questa operazione viene chiamata importazione:

```
import math
```

- Per utilizzare una funzione di un modulo dobbiamo specificare il nome del modulo che la contiene e il nome della funzione separati da un punto. Questo formato è chiamato notazione punto.

```
decibel = math.log10(17.0) # calcola il log naturale
angolo = 1.5
altezza = math.sin(angolo) # calcola il seno di un angolo
```

Funzioni dichiarate esternamente

- File itamat.py

```
def somma(op1,op2):
    ris = op1 + op2
    return ris

def sottrai(x,y):
    return x-y
```

- File esempiouso.py

```
import itamat
a=2
b=1
sm=itamat.somma(a,b)
print("risultato della somma: "+str(sm))
st=itamat.sottrai(a,b)
print ("risultato della sottrazione: "+str(st))
```

Percorso file della libreria

Come vengono localizzati i moduli (i file contenenti la definizione delle funzioni) o i pacchetti?

- Nell'esempio precedente, l'interprete python
  - cerca il file itamat.py (nel file system viene cercato il nome indicato nell'import con l'aggiunta del suffisso .py) …
  - secondo l'ordine descritto qui di seguito
    - directory in cui si trova il file esempiouso.py
    - directory in cui si trovano i moduli e i package della *libreria standard* di python (directory creata e riempita al momento dell'installazione di python)

  …non appena il file viene trovato, la ricerca termina

Libreria standard di python

- Nella documentazione ufficiale, oltre alla sintassi del linguaggio vengono descritti in dettaglio i contenuti della liberia standard
- Una modifica alla libreria standard determina una variazione della versione di python.
- Volete vedere dove python va a cercare i moduli della libreria, in aggiunta alla directory corrente?

```
import sys
print(sys.path)
```

```
['',
'/System/Library/.../Versions/2.7/lib/python27.zip',
'/System/Library/.../Versions/2.7/lib/python2.7',
...
'/Library/Python/2.7/site-packages',
'/System/Library/.../Versions/2.7/Extras/lib/python']
```

Package Manager

- In aggiunta alla libreria standard, è possibile installare dei pacchetti aggiuntivi
  - Posso usare un modulo o un package copiandolo nella mia directory di lavoro corrente. In questo modulo, però altri utenti non riuscirebbero ad utilizzarlo
  - L'installazione manuale di un pacchetto nel computer (per tutti gli utenti) può essere un'operazione complicata
  - Per questo motivo sono stati introdotti i *Package Manager*
- Il package manager è un software che si occupa di installare pacchetti aggiuntivi, rendendoli disponibili per tutti gli utenti di uno specifico computer

Repository di pacchetti

- Molti dei package manager esistenti, scaricano pacchetti da fonti accessibili via internet
- Esistono diverse fonti di pacchetti aggiuntivi (sia liberamente scaricabili, sia a pagamento).
- Una delle più grandi fonti di pacchetti python scaricabili è il Python Package Index. Maggiori dettagli qua: https://pypi.python.org/pypi
- Il contenuto della libreria e il tipo di package manager costituisce la maggior differenza tra le diverse distribuzioni del linguaggio python.
- Nella distribuzione python che utilizziamo in laboratorio, utilizziamo il package manger Conda. Se siete curiosi, potete trovare maggiori informazioni qua