Funzioni - Parte 1

Mirko Cesarini - Dario Pescini
nome.cognome@unimib.it

Università di Milano Bicocca

Funzioni

- Avete già visto ed utilizzato delle funzioni

```
1 a=input("Digita ...")
```

- La funzione può essere vista come un *nome* attribuito ad un insieme di istruzioni
  - codice (insieme di istruzioni) indipendente dalla parte rimanente del programma
  - filosofia: scrivi il codice di una funzione una volta, riutilizzalo più volte
- Caratteristiche di una funzione:
  - esegue un compito specifico
  - la funzione può essere *chiamata* (*invocata*, *attivata*) in diversi punti del programma di cui fa parte come se fosse una singola istruzione

Vantaggi delle funzioni

- Attraverso il riutilizzo, facilitano la scrittura del codice
  - All'interno di uno script aiutano a ridurre la duplicazione del codice
  - E' possibile riutilizzare una funzione in uno script diverso da quello in cui (la funzione) è stata scritta
  - E' possibile utilizzare funzioni già scritte da altri, senza conoscerne l'implementazione
- Migliorano ...
  - la comprensione del codice
  - l'organizzazione
  - le operazioni di manutenzione ed evoluzione

Funzioni e dati

- Una funzione in Python
  - può essere chiamata ad operare più volte su dati diversi
  - può restituire o non restituire un risultato

```
1 l1=input("Valore? ")
2 l2=input(msg)
3 l3=abs(-2)
4 print('Fine')
```

- Parametri passati alle funzioni: **msg**, **-2**
- Variabili su cui vengono memorizzati i risultati: **l1**, **l2**, **l3**

- In generale

Output $\leftarrow$ **Funzione** $\leftarrow$ **Parametri (in input)**

Funzioni composte

In Python le funzioni possono essere composte, facendo in modo che il risultato di una possa essere usato come argomento di un'altra:

```
1 print(float(abs(-3)))
```

```
3.0
```

3.0 ← **float()** ← **abs()** ← **-3**

Terminologia

- Subroutine è un altro nome che può essere usato al posto di funzione
- Precisazione:nei linguaggi di programmazione, le subroutine si distinguono in procedure e funzioni
  - Procedura (in altri linguaggi): può accettare parametri, ma non restituisce alcun risultato
  - Funzione (in altri linguaggi): può accettare parametri, restituisce un risultato
  - Esempi:

```
1 a=input()                #classificabile come funzione
2 print("Messaggio")       #classificabile come procedura
```

- Nell'ultimo esempio, print non restituisce un valore, si occupa solo di stampare un'informazione a video
- Ulteriore precisazione: in python non si distingue tra procedure e funzioni, si parla genericamente di funzioni

Prossimi passi

- Come creare ed utilizzare una funzione all'interno di uno script python
- Come importare ed utilizzare una funzione già esistente (creata da altre persone)

Creazione di una nuova funzione

Mirko Cesarini - Dario Pescini
nome.cognome@unimib.it

Università di Milano Bicocca

Aggiungere nuove funzioni

- La creazione di nuove funzioni è una fra le peculiarità più utili di un linguaggio di programmazione
- La sintassi per la definizione di una funzione in python è:

```
1 def nomeFunzione( parametro1, parametro2, ..., parametroN ):
2       istruzione1
3       ...
4       istruzioneN
```

- Può essere usato qualsiasi nome per una funzione, fatta eccezione per le parole riservate di Python
- Una funzione deve essere definita prima di essere usata

Flusso di esecuzione e funzioni

- L'esecuzione inizia dalla "prima istruzione" del corpo principale del programma
- #cp Corpo principale del programma
- #cf Corpo della funzione. Istruzioni eseguite solo quando la funzione viene chiamata dal corpo principale del programma
- Le funzioni è come se non esistessero fino a che non vengono chiamate

```
1  nome='Mario'                          #cp
2  lingua = 'italiano'                   #cp
3  saluto=cs('italiano')                 #cp
4  print(saluto+' '+nome)                #cp
5  #stampa: ciao Mario                   #cp
6  print(cs('inglese')+' Susan')         #cp
7  #stampa: hello Susan                  #cp
8
9  def cs(lingua):
10     if lingua=='italiano':            #cf
11         return 'ciao'                 #cf
12     elif lingua=='inglese':           #cf
13         return 'hello'                #cf
14     else:                             #cf
15         return ''                     #cf
```

Notate qualche problema nel codice?

Flusso di esecuzione 2

- Le funzioni possono essere definite in qualsiasi punto del programma
- Unica condizione: una funzione deve essere definita prima di poter essere usata
- **La slide precedente contiene un errore**. Qua a fianco trovate la soluzione corretta
- Per evitare problemi di questo tipo conviene dichiarare tutte le funzioni all'inizio del file

```
1  def cs(lingua):
2      if lingua=='italiano':      #cf
3          return 'ciao'           #cf
4      elif lingua=='inglese':     #cf
5          return 'hello'          #cf
6      else:                       #cf
7          return ''               #cf
8
9  nome='Mario'                    #cp
10 lingua = 'italiano'             #cp
11 saluto=cs('italiano')           #cp
12 print(saluto+' '+nome)          #cp
13 #stampa: ciao Mario             #cp
14 print(cs('inglese')+' Susan')   #cp
15 #stampa: hello Susan            #cp
```

Parametri, come mai?

Output $\leftarrow$ **Funzione** $\leftarrow$ **parametri (in input)**

- Esempi di uso di funzioni

```
1 print( moltiplica(3, 5) )
```

```
15
```

```
1 print( moltiplica(4, 2) )
```

```
8
```

- Quando definisco una funzione non conosco a priori i valori sui quali dovrà operare . . .
- come faccio a manipolare tali valori dentro la funzione?

Parametri formali, parametri attuali

- Soluzione: nella definizione uso i *parametri formali* per dare un nome . . . ai dati che ancora non conosco

```
1 def moltiplica(x, y):  #x ed y sono parametri formali
2     return x * y  # return comunica il risultato al
3                   # programma chiamante e termina
4                   # l'esecuzione della funzione
5 a=10
6 b=moltiplica(a,20)  # a e 20 sono parametri attuali
7 print(b)
```

```
200
```

- L'interprete Python, di volta in volta, collegherà i valori reali (chiamati parametri attuali) ai parametri formali

Variabili nelle funzioni

- E' possibile dichiarare delle variabili all'interno delle funzioni

```
1 def moltiplica(x, y):
2     risultato=0
3     risultato = x * y
4     return risultato
```

- Al termine della funzione, le variabili dichiarate internamente (per es. *risultato*) cessano di esistere
- E' opportuno dare alle variabili, dichiarate nelle funzioni, nomi diversi dai nomi delle variabili dichiarate nel corpo principale del programma
  - ...questo argomento sarà affrontato in maniera più approfondita in seguito
  - per ora seguite questa raccomandazione

Calcolo dei numeri primi (richiamo)

Riprendiamo l'algoritmo per l'individuazione dei numeri primi compresi tra 2 ed n.

Versione originale

```
1  n = int(input("Quale N? "))
2  num=2
3  while num<=n:
4      div = 2
5      primo = True
6      while div<num and primo==
7                                  True:
8          if num%div==0:
9              primo = False
10         div=div+1
11     if primo == True:
12         print(num)
13     num = num + 1
14 print("\n")
```

Utilizzando le funzioni

```
1  def verificaPrimo(val):
2      div = 2
3      while div < val:
4          if val%div==0:
5              return False
6          div=div+1
7      return True
8
9  n = int(input("Quale N? "))
10 num=2
11 while num<=n:
12     ris=verificaPrimo(num)
13     if ris == True:
14         print(num)
15     num = num + 1
16 print("\n")
```

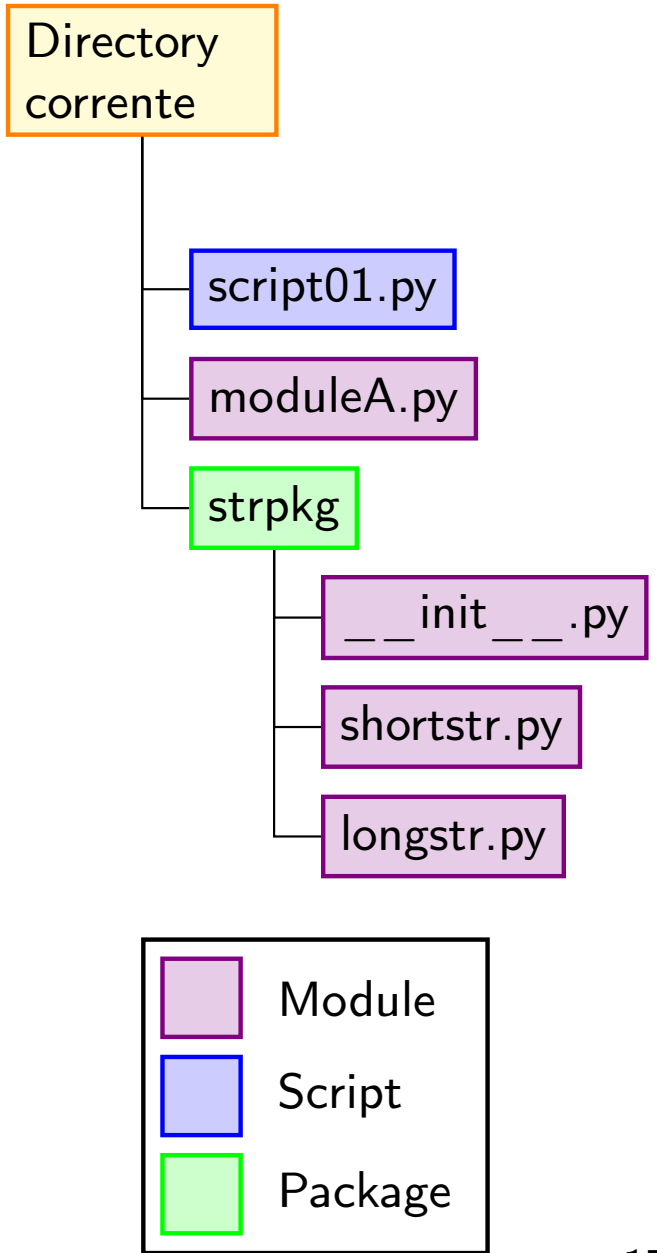
Importare ed utilizzare una funzione dall'esterno

Mirko Cesarini - Dario Pescini
nome.cognome@unimib.it

Università di Milano Bicocca

Terminologia: moduli, package e funzioni

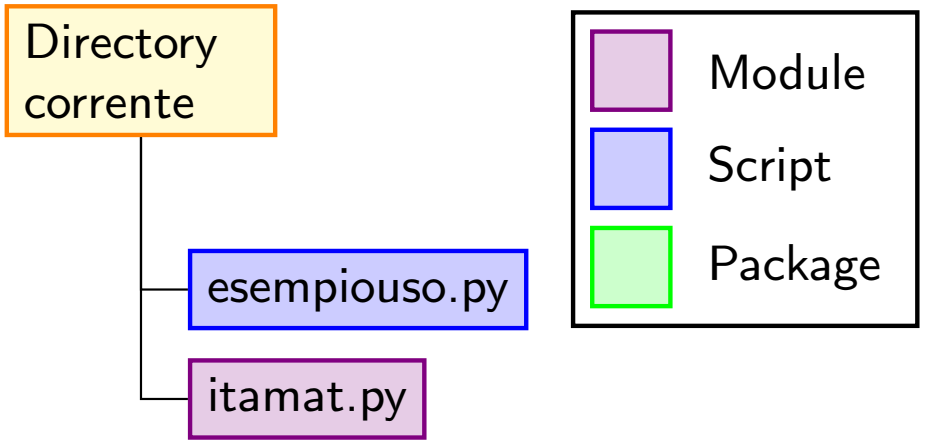
- *Modulo*: un file .py con all'interno una collezione di funzioni pronte per l'uso
- *Package*: un insieme di moduli che *collaborano* tra loro per svolgere compiti complessi
  - Un package è una directory contenente uno o più moduli
  - La directory di un package deve avere un file particolare: __init__.py per evitare che una directory contenente alcuni script .py venga scambiata per un package
- Per poter utilizzare le funzioni di un modulo dobbiamo dire all'interpreta di caricare il modulo in memoria. Questa operazione viene chiamata importazione



Importare un modulo (presente nella dir. corrente)

- File itamat.py (modulo)

```
1 def somma(op1,op2):
2     ris = op1 + op2
3     return ris
4
5 def sottrai(x,y):
6     return x-y
```



- File esempiouso.py

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

Localizzazione di moduli nel file system

- Come vengono localizzati i moduli (i file contenenti la definizione delle funzioni) o i pacchetti?
- Cosa viene cercato?
  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)
- Dove viene cercato itamat.py?
  Inizia una ricerca in diverse directory, secondo l'ordine descritto qui di seguito. Non appena viene trovato un file itamat.py, la ricerca termina.
  - directory di lavoro corrente (nel ns caso, la directory in cui si trova il file esempiouso.py)
  - directory in cui si trovano i moduli e i package della *libreria standard* di python
- Che cosa è la *libreria standard di python*?

Libreria standard di Python

- 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 installarlo nel vostro computer (vedremo più avanti come)

Importare un modulo della libreria standard

- I moduli della libreria standard possono essere importati anche se non risiedono nella directory di lavoro corrente

```
1 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.

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

Focus sulla libreria standard di python

- Nella documentazione ufficiale del linguaggio python, 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?

```
1 import sys
2 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.

Conda Package Manager

- Nella distribuzione python che utilizziamo in laboratorio, utilizziamo il package manger *Conda*
- Se siete curiosi, potete trovare maggiori informazioni qua