Laurea in Ingegneria Gestionale

Corso di Fondamenti di Informatica
A.A. 2017/2018

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Funzioni - 1

Le seguenti slide sono tratte principalmente dal lavoro di Giorgio Fumera (Univ. degli Studi di Cagliari) e riadattate da Raffele Nicolussi (Sapienza) e Domenico Lembo (Sapienza). Alcune sono invece una rielaborazione di slide analoghe di Diego Calvanese (Univ. di Bolzano).

Funzioni

Le *funzioni* nei linguaggi di alto livello

. Tutti i linguaggi di alto livello mettono a disposizione degli utenti, come parte dell'ambiente di programmazione, un insieme di programmi che realizzano operazioni di utilità generale. Per esempio:

- calcolo di funzioni trigonometriche e logaritmiche
- accesso ai *file* della memoria secondaria
- realizzazione di interfacce grafiche per i propri programmi

Tali programmi sono detti funzioni per analogia con il concetto di funzione matematica, poiché sono in grado di elaborare un determinato insieme di valori di ingresso e di produrre un risultato.

Funzioni predefinite e definite dall'utente

Le funzioni disponibili in un linguaggio di programmazione sono dette predefinite, o *built-in*.

L'insieme di tali funzioni viene detto libreria.

Esempi di funzioni predefinite incontrate fino ad ora: `print(..)`, `input(..)`, `int(..)`, `float(..)`, `str(..)`, `range(..)`, `len(..)`, `abs(..)`,…

Tutti i linguaggi di programmazione consentono inoltre ai programmatori di definire **nuove** funzioni: tali funzioni sono dette definite dall'utente (*user-defined*).

Utilità delle funzioni

Le funzioni **predefinite** evitano agli utenti di dover scrivere programmi per realizzare molte operazioni di utilità generale.

La definizione di **nuove** funzioni è uno strumento che presenta diversi vantaggi:

- consente di semplificare la scrittura di programmi complessi suddividendoli in più parti (funzioni), ciascuna delle quali svolge un compito **distinto** dalle altre, e può essere sviluppata in modo **indipendente** da esse
- l'esecuzione di una funzione può essere richiesta in diversi punti di uno stesso programma, senza dover scrivere più volte le sue istruzioni
- una stessa funzione può essere usata in programmi diversi

La libreria del linguaggio Python

Il linguaggio Python comprende un vasto insieme di funzioni di libreria, **disponibili in qualsiasi ambiente di programmazione**.

Alcune funzioni predefinite sono direttamente accessibili dalla *shell* e dai propri programmi. Le altre sono suddivise in diverse categorie, dette a loro volta *librerie* (per es., la libreria matematica); per poter usare queste ultime è necessaria un'istruzione particolare.

In questo corso si useranno (*almeno*) due librerie:

- funzioni matematiche
- funzioni per la generazione di numeri casuali

Caratteristiche delle funzioni

A ogni funzione è associato un **nome simbolico** (analogo ai nomi delle variabili), detto nome della funzione.

Ogni funzione può elaborare un determinato numero di valori di ingresso, detti (per analogia con gli argomenti delle funzioni matematiche) argomenti.

Ogni funzione **può** infine restituire un valore di un determinato tipo, come **risultato** dell'elaborazione svolta sugli argomenti.

Come caso particolare, esistono anche funzioni che non ricevono nessun argomento

Esecuzione di una funzione: *chiamata*

L'esecuzione di una funzione si ottiene attraverso una una specifica **espressione**, detta chiamata.

Sintassi: nome-funzione (arg$_1$, arg$_2$, . . . , arg$_n$)

- arg$_1$, . . . ,arg $_n$ sono **espressioni** Python, i cui **valori** costituiranno gli argomenti della funzione
- il numero degli argomenti e il tipo di ciascuno di essi (per es., numeri interi, numeri frazionari, stringhe, valori logici) dipende dalla specifica funzione; se il tipo di un argomento non è tra quelli previsti, si otterrà un messaggio d'errore
- come tutte le espressioni, anche la chiamata di una funzione produce un valore: questo coincide con il valore restituito dalla funzione

Principali funzioni Python predefinite: `input`

L'espressione `input` già vista in precedenza **è in realtà una funzione predefinita del linguaggio Python** (si noti che la sua sintassi coincide con quella della chiamata di una funzione).

Rivisitando la sintassi di `input` dal punto di vista di una funzione, si deduce che `input` può non ricevere argomenti oppure può riceverne uno (di norma una stringa, anche se può essere un'espressione qualsiasi):

- `input()`
- `input(`arg`)`

Una chiamata di `input` produce la stampa nella *shell* del valore di arg (se presente), l'acquisizione di una sequenza di caratteri che dovranno essere inseriti attraverso la tastiera (fino alla pressione del tasto `INVIO`) e la restituzione del valore di tale espressione.

Principali funzioni Python predefinite: raw_input

Nelle versioni di Python precedenti al 3.x se si desidera acquisire attraverso la tastiera una sequenza di caratteri da memorizzare sotto forma di stringa, l'uso della funzione `input` richiede che l'utente inserisca anche gli apici (singoli o doppi) in apertura e in chiusura della sequenza.

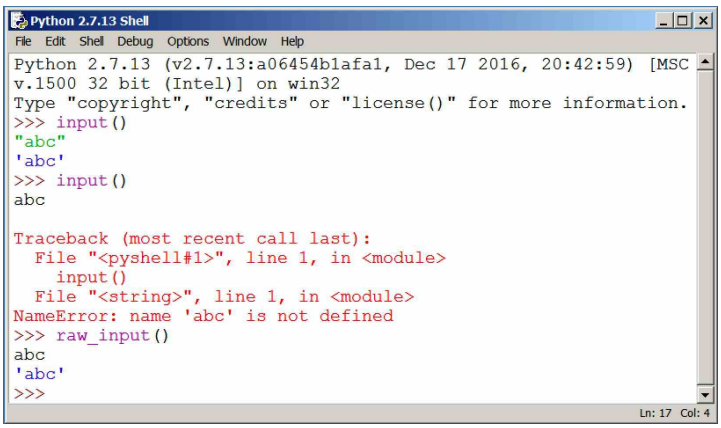
Per evitare ciò è possibile usare la funzione `raw_input`. Tale funzione è identica a `input`, tranne per il fatto che essa non interpreta la sequenza di caratteri inserita attraverso la tastiera come se fosse un'espressione, ma la restituisce sotto forma di stringa.

La sintassi della chiamata è la seguente:

- `raw_input()`
- `raw_input(arg)`

Differenze tra input raw_input

Un esempio della differenza tra `input` e `raw_input`: nella seconda chiamata a `input` la sequenza di caratteri `abc` viene intepretata come il nome di una variabile (causando un errore, dato che tale variabile non è stata definita in precedenza). Nella chiamata di `raw_input`, la stessa sequenza non viene interpretata ma viene restituita sotto forma di stringa.



(solo per Python 2.x)

Principali funzioni Python predefinite

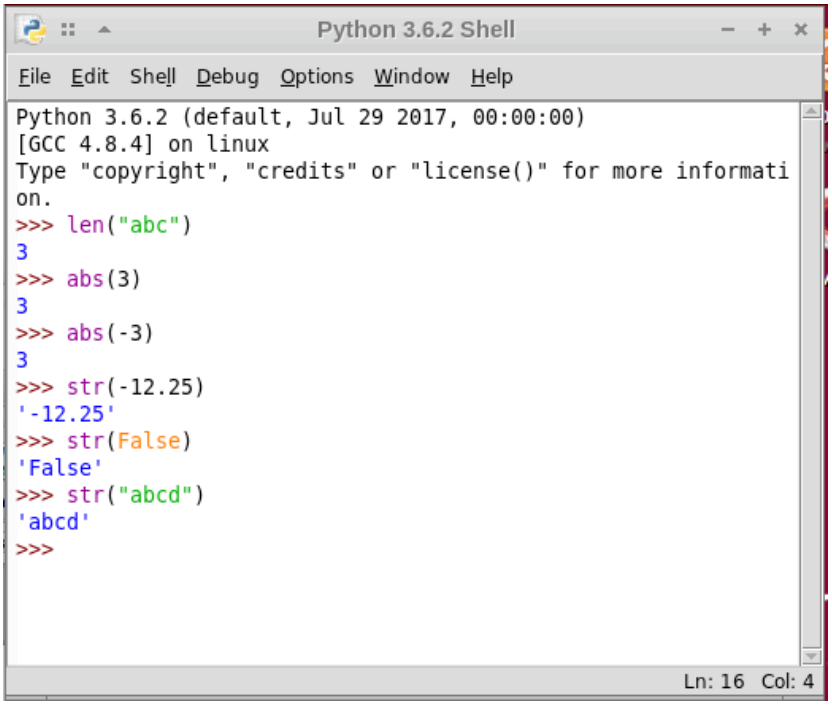
Altre funzioni predefinite di utilità generale sono le seguenti:

- `len(`stringa`)`
  restituisce il numero di caratteri di una stringa
- `abs(`numero`)`
  restituisce il valore assoluto di un numero
- `str(`espressione`)`
  restituisce una stringa composta dalla sequenza di caratteri corrispondenti alla rappresentazione del valore di espressione (che può essere di un qualsiasi tipo: numero, stringa, valore logico, ecc.)

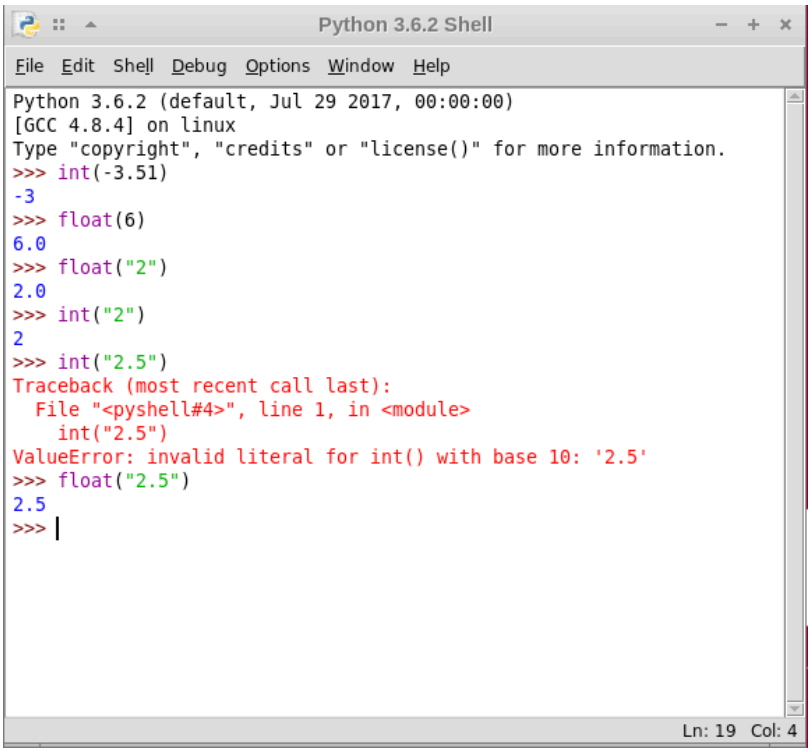
Principali funzioni Python predefinite

- `int(`numero`)`
  restituisce la parte intera di un numero
- `float(`numero`)`
  restituisce il valore di numero come numero frazionario (*floating point*);
- `int(`stringa`)`
  Se stringa contiene la rappresentazione di un numero **intero,** restituisce il numero corrispondente a tale valore; in caso contrario produce un errore
- `float(`stringa`)`
  Se stringa contiene la rappresentazione di un numero qualsiasi (sia intero che frazionario), restituisce il suo valore espresso come numero frazionario; in caso contrario produce un errore

Principali funzioni Python predefinite: esempi



Principali funzioni Python predefinite: esempi



Principali funzioni Python predefinite

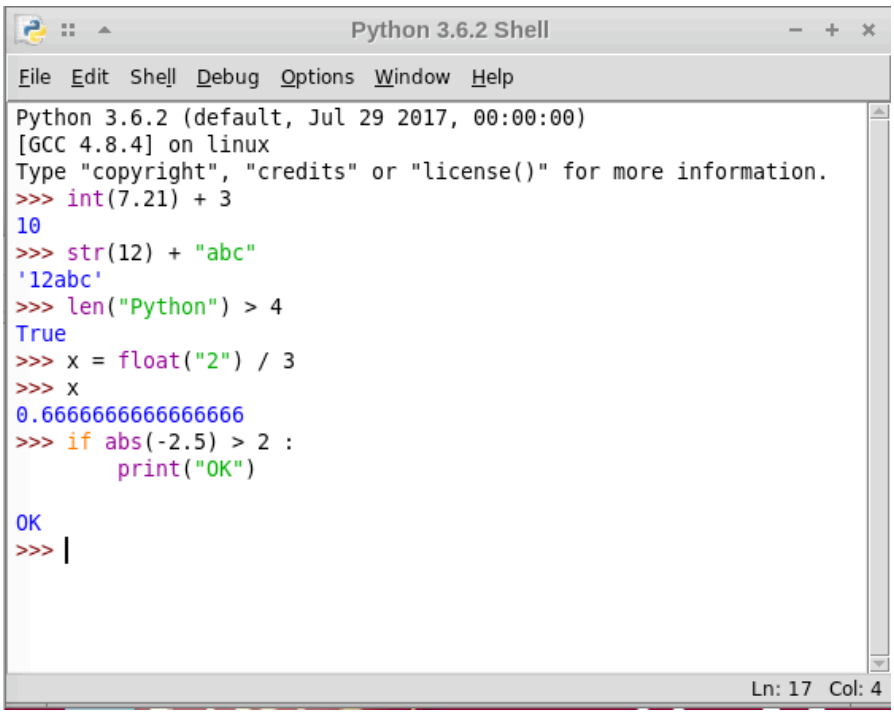
Ulteriori funzioni predefinite, necessarie per l'elaborazione di altri tipi di dato Python (liste e dizionari) e per l'accesso ai *file* della memoria secondaria, verranno presentate più avanti.

Ancora sulla sintassi della chiamata di funzioni

Come si è già detto, dal punto di vista sintattico la chiamata di una funzione è un'**espressione**, e quindi deve rispettare le stesse regole viste in precedenza per le espressioni:

- può essere scritta direttamente nella *shell* (come negli esempi precedenti): in questo caso anche il valore restituito dalla funzione verrà mostrato nella *shell*
- può comparire come operando di una qualsiasi espressione più complessa (aritmetica, logica, o composta da stringhe)
- può comparire nell'espressione di un'istruzione di assegnamento
- può comparire nelle espressioni condizionali delle istruzioni condizionali e iterative

Chiamata di funzioni: esempi



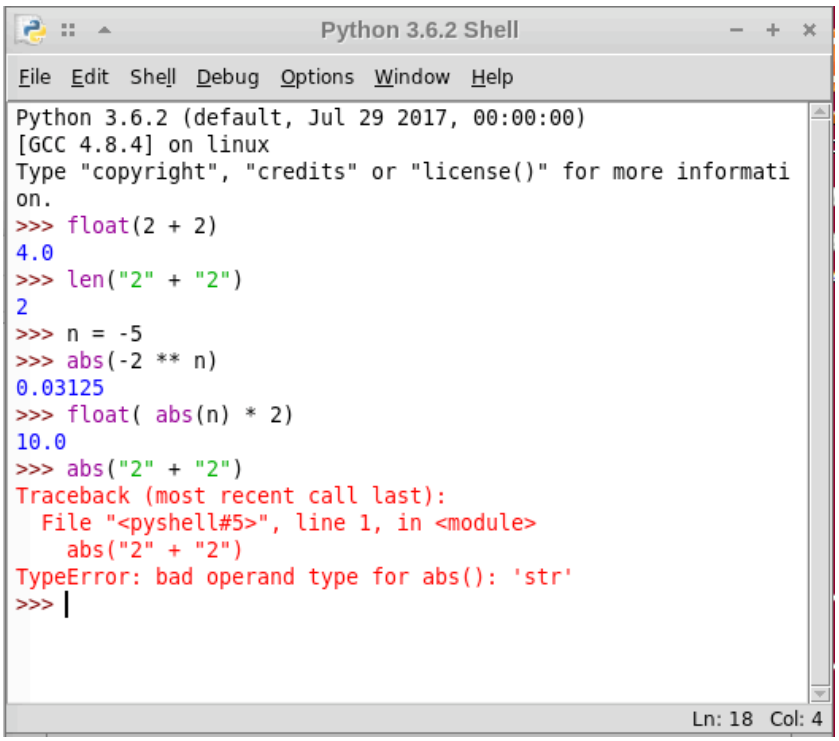
Chiamata di funzioni: sintassi degli argomenti

Si è anche detto che gli argomenti di una chiamata di funzione sono a loro volta costituiti da **espressioni**.

In ciascuno degli argomenti può quindi comparire un'espressione Python qualsiasi, purché produca un valore di un tipo previsto dalla funzione (in caso contrario si otterrà un messaggio d'errore).

Ne consegue come caso particolare che una chiamata di funzione può contenere tra le espressioni dei suoi argomenti altre chiamate di funzioni (chiamate *nidificate*).

Argomenti delle chiamate di funzione: esempi



Librerie di funzioni del linguaggio Python

Molte funzioni predefinite fanno parte di specifiche librerie Python, che a loro volta sono identificate univocamente da un nome simbolico.

Per esempio, le funzioni matematiche fanno parte di una libreria di nome `math`, mentre la libreria `random` comprende varie funzioni per la generazione di numeri casuali.

Le principali funzioni delle librerie `math` e `random` sono descritte di seguito.

Principali funzioni della libreria math

funzione	descrizione
`cos(x)`	coseno (x deve essere espresso in radianti)
`sin(x)`	seno (come sopra)
`tan(x)`	tangente (come sopra)
`acos(x)`	arco-coseno (x deve essere nell'intervallo [−1, 1])
`asin(x)`	arco-seno (come sopra)
`atan(x)`	arco-tangente
`radians(x)`	converte in radianti un angolo espresso in gradi
`degrees(x)`	converte in gradi un angolo espresso in radianti
`exp(x)`	e^x
`log(x)`	$\ln x$
`log(x,b)`	$\log_b x$
`log10(x)`	$\log_{10} x$
`pow(x,y)`	x^y
`sqrt(x)`	radice quadrata di x

Tutte le funzioni di questa libreria restituiscono un numero **frazionario**.

La libreria random

Alcune funzioni della libreria `random` sono le seguenti:

funzione	descrizione
`random()`	genera un numero reale nell'intervallo [0, 1), da una distribuzione di probabilità **uniforme** (cioè, ogni valore di tale intervallo ha la stessa probabilità di essere "estratto")
`uniform(a,b)`	come sopra, nell'intervallo [*a*, *b*) (gli argomenti sono numeri qualsiasi)
`randint(a,b)`	genera un numero intero nell'insieme {*a*, . . . , *b*}, da una distribuzione di probabilità **uniforme** (gli argomenti devono essere numeri interi)

Ogni chiamata di tali funzioni produce un numero **pseudo-casuale**, indipendente (in teoria) dai valori prodotti dalle chiamate precedenti.

L'istruzione from-import

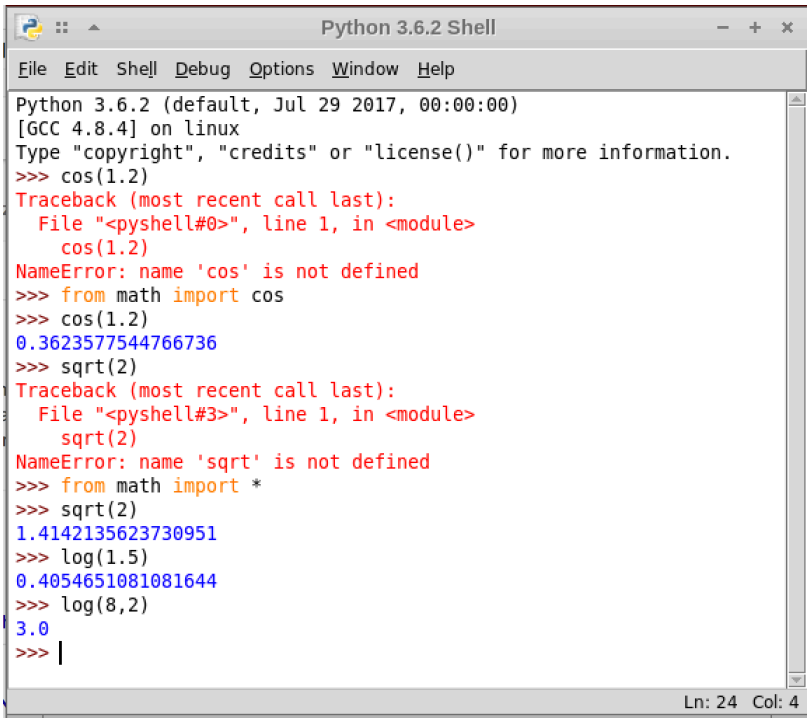
Prima di chiamare una funzione di librerie come `math` e `random` (sia nella *shell* che in un programma) è necessaria l'istruzione `from-import`, che prevede la seguente **sintassi**:

`from` nome-libreria `import` nome-funzione

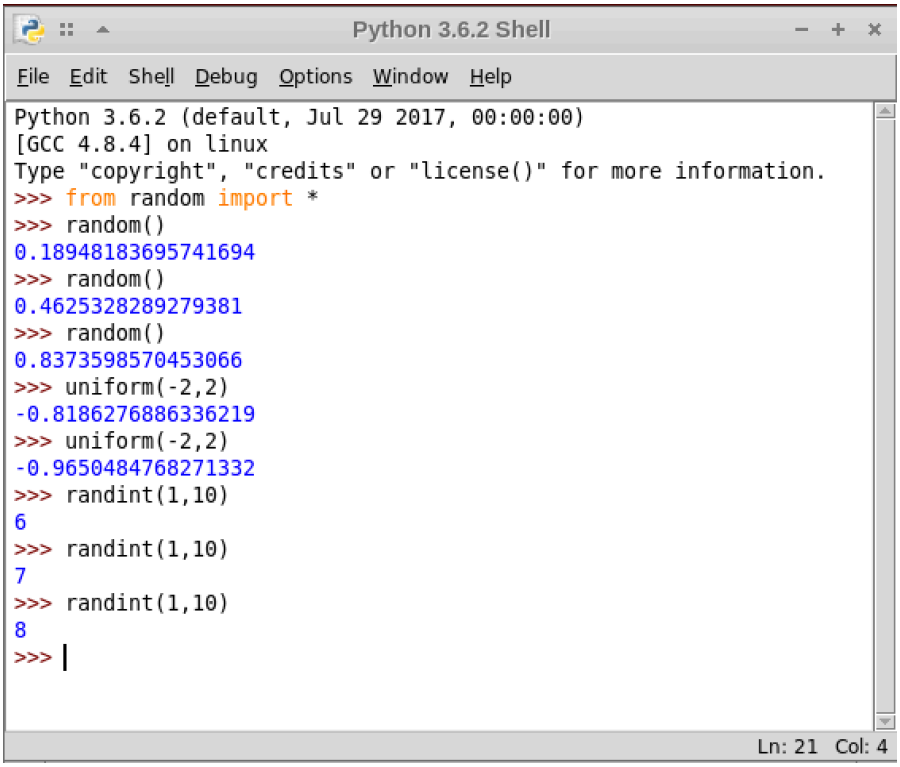
- nome-libreria è il nome simbolico di una libreria
- nome-funzione può essere:
  - il nome di una specifica funzione di tale libreria (questo consentirà di usare solo tale funzione)
  - il simbolo `*` indicante **tutte** le funzioni di tale libreria

Se tale istruzione non viene usata ogni chiamata produrrà un errore, come mostrato negli esempi seguenti.

L'istruzione from-import: esempi



L'istruzione from-import: esempi



La libreria math: costanti matematiche notevoli

Oltre a varie funzioni, nella libreria `math` sono definite due variabili che contengono il valore (approssimato) delle costanti matematiche π (3,14. . .) ed *e* (la base dei logaritmi naturali: (2,71. . .):

- `pi`
- `e`

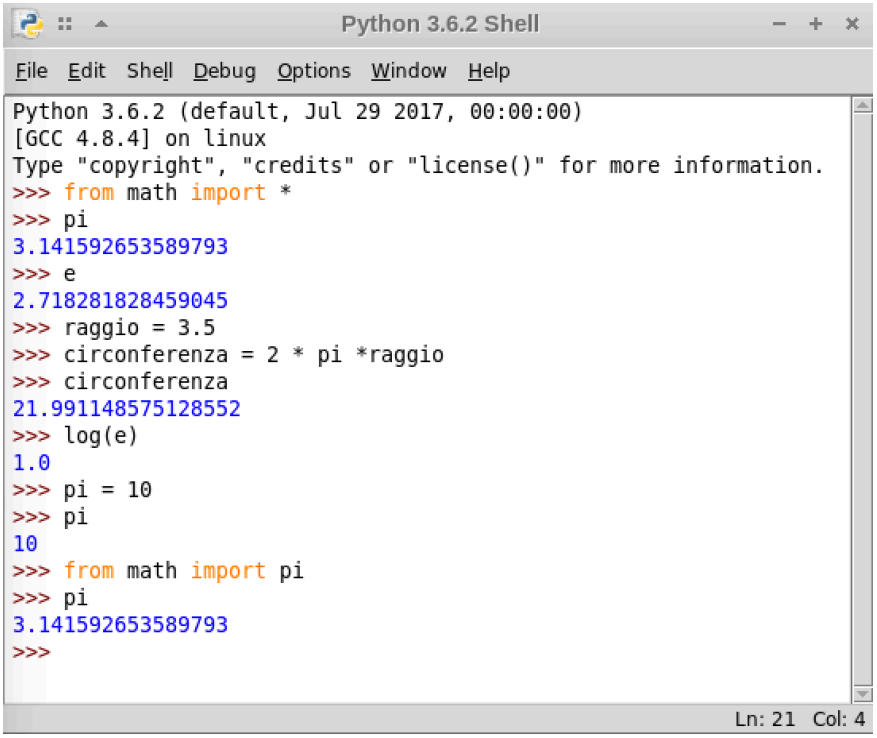
Anche per usare queste variabili è necessaria l'istruzione `from-import`, in una delle due versioni:

- `from math import *`
- `from math import` nome-variabile

  dove nome-variabile dovrà essere `pi` oppure `e`.

Nota: Il valore delle due variabili può essere modificato con istruzioni di assegnamento, anche se ciò non è consigliabile. Il loro valore originale può essere ripristinato eseguendo di nuovo l'istruzione `from-import`

Le variabili `pi` ed `e`: esempi



Definizione di nuove funzioni

Come si è già detto, i linguaggi di programmazione di alto livello consentono agli utenti anche la definizione di **nuove** funzioni. La sintassi della chiamata di tali funzioni è identica a quella delle funzioni predefinite.

La definizione di una nuova funzione è composta dai seguenti elementi:

- il **nome** della funzione
- il **numero** dei suoi argomenti
- la sequenza di istruzioni, detta corpo della funzione, che dovranno essere eseguite quando la funzione sarà chiamata

La definizione di una nuova funzione avviene attraverso l'istruzione `def`, che può essere scritta sia nella *shell* che in un file .py, ma si consiglia fortemente di scrivere sempre la funzione in un file.

Definizione di nuove funzioni: l'istruzione `def`

Sintassi:
`def` nome-funzione `(`par_1`, ..., `par_n`) :`
    corpo della funzione

- nome-funzione è un nome simbolico scelto dal programmatore, con gli stessi vincoli a cui sono soggetti i nomi delle variabili
- par_1, . . . ,par_n sono nomi (scelti dal programmatore) di variabili, dette parametri della funzione, alle quali l'interprete assegnerà i valori degli **argomenti** che verranno indicati nella **chiamata** della funzione
- corpo della funzione è una sequenza di una o più istruzioni **qualsiasi**, ciascuna scritta in una riga **distinta**, con un **rientro** di almeno un carattere, identico per **tutte** le istruzioni

La prima riga della definizione (contenente i nomi della funzione e dei parametri) è detta **intestazione** della funzione.

Definizione di nuove funzioni: l'istruzione `return`

Per concludere l'esecuzione di una funzione e indicare il valore che la funzione dovrà **restituire** come risultato della sua chiamata si usa l'istruzione `return`.

Sintassi: `return` espressione
Dove espressione è un'espressione Python **qualsiasi**.

Questa istruzione può essere usata solo **solo** all'interno di una funzione.

Se una funzione **non** deve restituire nessun valore:

- l'istruzione `return` può essere usata, senza l'indicazione di nessuna espressione, per concludere l'esecuzione della funzione
- se non si usa l'istruzione `return`, l'esecuzione della funzione terminerà dopo l'esecuzione dell'ultima istruzione del suo corpo

Definizione e chiamata di una funzione

L'esecuzione dell'istruzione `def` **non** comporta l'esecuzione delle istruzioni della funzione: tali istruzioni verranno eseguite solo attraverso una **chiamata** della funzione.

L'istruzione `def` dovrà essere eseguita una sola volta, **prima** di qualsiasi chiamata della funzione. In caso contrario il nome della funzione non sarà riconosciuto dall'inteprete, e la chiamata produrrà un messaggio d'errore.

Si noti che il riavvio della *shell* a seguito dell'esecuzione di un programma comporta la cancellazione delle eventuali funzioni, oltre alle variabili, definite in precedenza.

Chiamata di una funzione: meccanismo di esecuzione

Come per le funzioni predefinite, anche per quelle definite dall'utente il numero di argomenti indicati nella chiamata dovrà corrispondere al numero di argomenti previsti dalla funzione, cioè al numero dei parametri indicati nell'intestazione.

L'inteprete esegue la chiamata di una funzione nel modo seguente:

- **copia** il valore di ciascun argomento nel parametro corrispondente (quindi tali variabili possiedono già un valore nel momento in cui inizia l'esecuzione della funzione)
- esegue le istruzioni del corpo della funzione, fino a incontrare l'istruzione `return` oppure l'ultima istruzione del corpo
- se l'eventuale istruzione `return` è seguita da un'espressione, restituisce il valore di tale espressione come risultato della chiamata

Definizione di funzioni: esempio

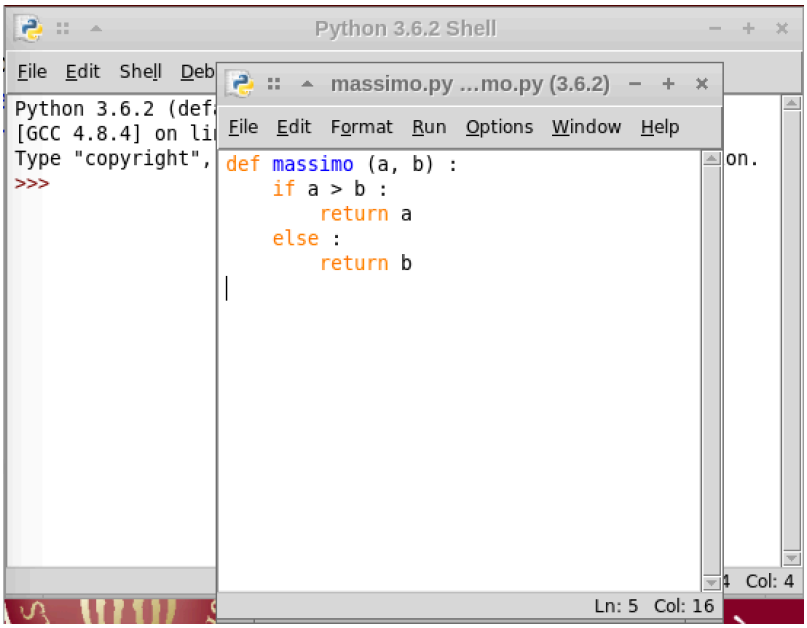
Si supponga di voler definire una funzione che restituisca il più grande tra due numeri ricevuti come argomenti.

Scegliendo `massimo` come nome della funzione, `a` e `b` come nomi dei suoi parametri, la funzione può essere definita come segue:

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

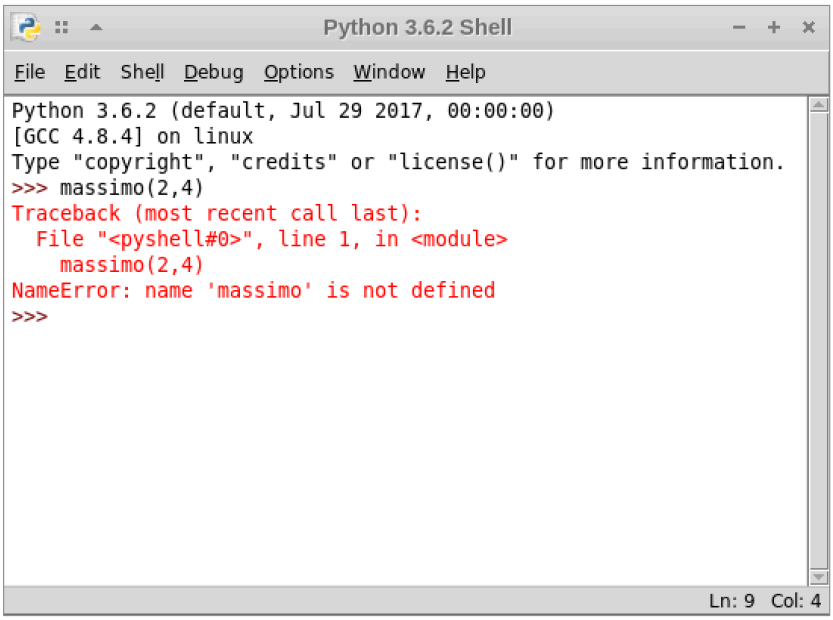
Definizione di funzioni: esempio

Scriviamo la definizione di una funzione in una finestra dell'`editor`, e salviamola in un `file`:



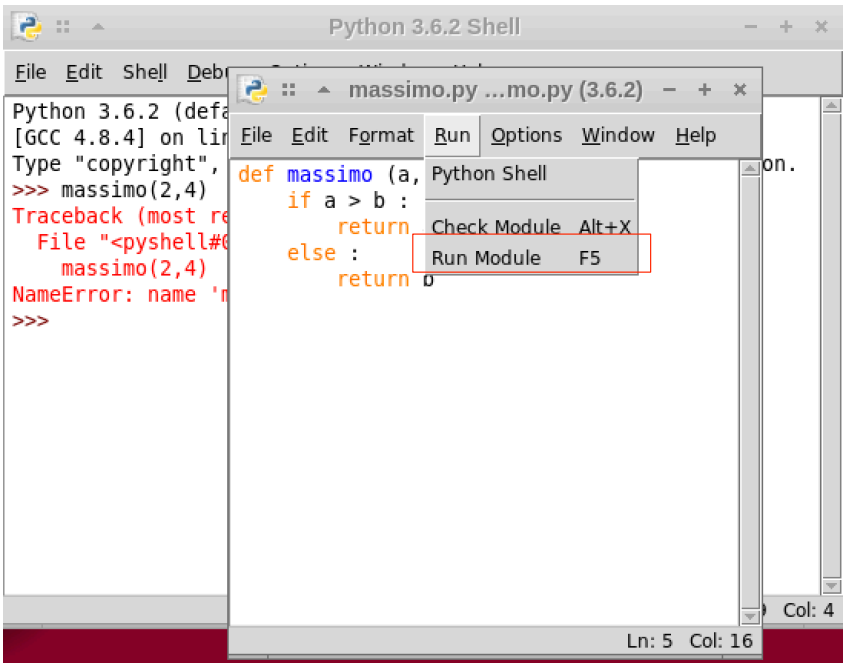
Definizione di funzioni: esempio

Dopo il salvataggio del *file* la funzione che esso contiene non è ancora "nota" all'interprete: una sua chiamata scritta nella *shell* produrrà un messaggio d'errore:



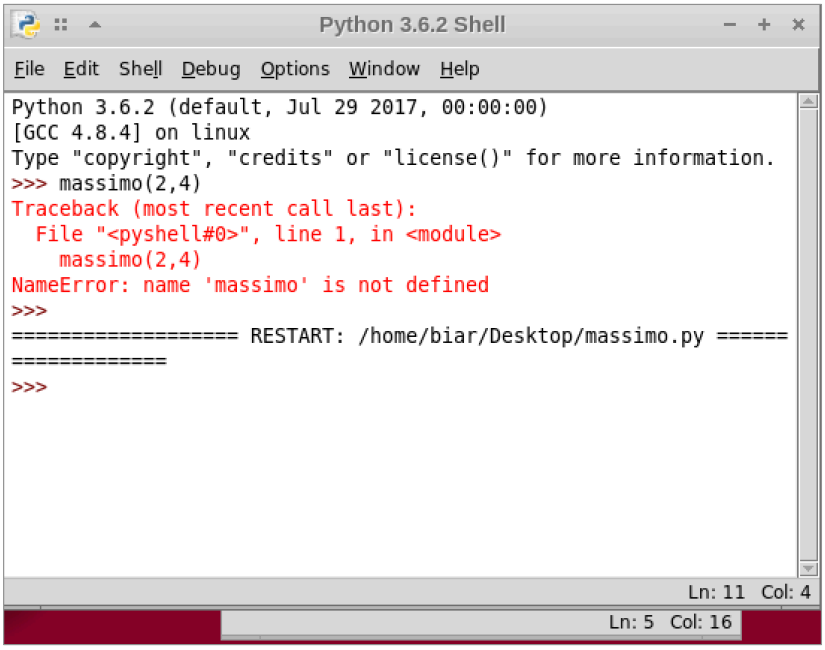
Definizione di funzioni: esempio

Prima di chiamare una funzione è necessario eseguire l'istruzione `def` nel *file* che contiene la definizione:



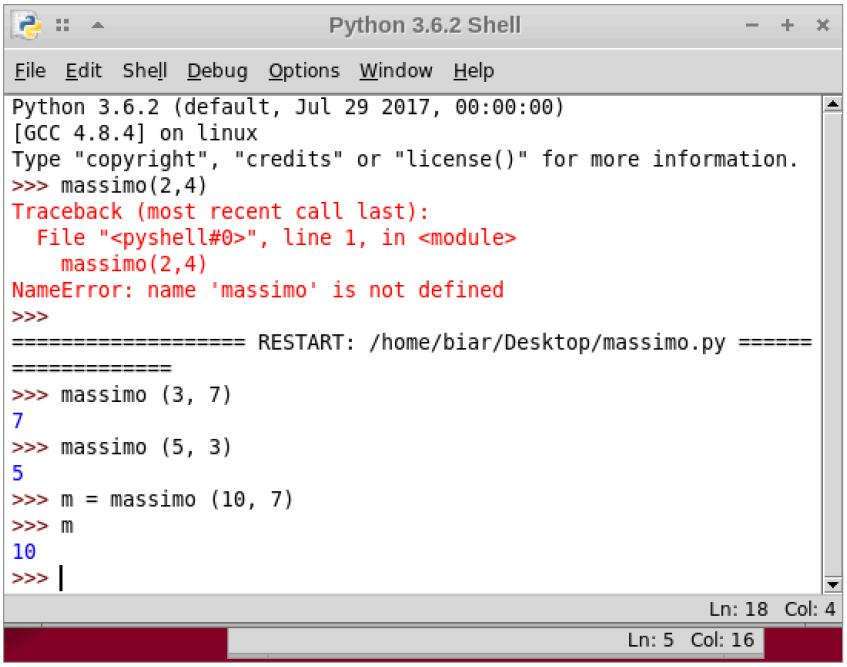
Definizione di funzioni: esempio

Si ricordi che l'esecuzione dell'istruzione `def` non comporta l'esecuzione delle istruzioni della funzione (ma solo la sua definizione):



Definizione di funzioni: esempio

Da questo momento è possibile chiamare la funzione dalla *shell*.



Ancora sui parametri di una funzione

Come si è detto in precedenza, quando una funzione viene chiamata, prima di iniziare l'esecuzione delle sue istruzioni l'interprete assegna ai parametri i valori degli argomenti indicati nella chiamata.

Nell'esempio precedente le variabili (parametri) `a` e `b` della funzione `massimo` avranno già un valore nel momento in cui inizierà l'esecuzione delle istruzioni del corpo della funzione.

Questo significa che i valori delle variabili che costituiscono i parametri della funzione **non devono** essere definiti per mezzo di istruzioni di assegnamento nel **corpo** della funzione, ma vengono definiti nelle **chiamate** della stessa funzione.

Definizione di funzioni: esempi

In precedenza si sono mostrati esempi di programmi per il calcolo del massimo comun divisore tra due numeri naturali con l'algoritmo di Euclide, e della somma dei primi *m* termini della serie armonica (per un dato valore di *m*).

Di seguito si mostra come gli stessi programmi possano essere scritti sotto forma di funzioni. In entrambi i casi quelli che nei programmi erano i valori d'ingresso (e venivano acquisiti usando la funzione `input`) sono ora definiti come argomenti della funzione, mentre il risultato, che nei programmi veniva stampato nella *shell*, viene restituito dalla funzione mediante l'istruzione `return`.

Le funzioni possono essere definite e chiamate come mostrato nell'esempio precedente.

Programma per il calcolo del massimo comun divisore

```
# File MCD_3.py

m = int(input("Primo numero: "))
n = int(input("Secondo numero: "))
while m != n :
    if m > n :
        m = m - n
    else :
        n = n - m
print ("Il MCD e'", m)
```

Funzione per il calcolo del massimo comun divisore

Calcolo del massimo comun divisore con l'algoritmo di Euclide (disponibile nel *file* `MCD_funzione.py`):

```
def mcd (a, b) :
    while a != b :
        if a < b :
            b = b - a
        else :
            a = a - b
    return a
```

Programma per la somma dei primi m termini della serie armonica

```
# File serieArmonica.py

m = int(input("Numero di termini della serie armonica: "))
serie = 0
k = 1
while k <= m :
    serie = serie + 1 / k
    k = k + 1
print ("Somma dei primi", m, " termini: ", serie)
```

Funzione per la somma dei primi m termini della serie armonica

Calcolo della somma dei primi *m* termini della serie armonica (disponibile nel *file* `serie_armonica_funzione.py`):

```
def serie_armonica (m) :
    serie = 0
    k = 1
    while k <= m :
        serie = serie + 1 / k
        k = k + 1
    return serie
```

Importante osservazione

- Il programma per il calcolo del MCD (file `MCD_3.py`) si occupa anche di gestire l'acquisizione dei valori in input
- La funzione `mcd` (salvata nel file `MCD_funzione.py`) non ha al suo interno i comandi per leggere i dati immessi dall'utente. Piuttosto "assume" che questi dati siano già noti, e che le vengano passati come parametri
- Quindi, per ottenere lo stesso comportamento del programma, ma usando la funzione `mcd`, dobbiamo scrivere un altro programma che la invochi

```
m = int(input("Primo numero: "))
n = int(input("Secondo numero: "))
m = mcd(m,n)
print ("Il MCD e'", m)
```

- Per il calolo dei primi m termini della serie armonica vale un discorso analogo

Funzioni senza argomenti o senza risultato

Come caso particolare è possibile definire **funzioni che non ricevono argomenti** (come nel caso della funzione predefinita `random` della libreria omonima, descritta in precedenza), **oppure funzioni che non restituiscono un risultato.**

Lo scopo di una funzione che non restituisce un risultato potrebbe essere, per esempio, quello di **mostrare** un messaggio nella *shell*.

Se si vuole definire una funzione che **non riceve argomenti**, nell'intestazione si dovranno solo scrivere le parentesi tonde, senza il nome di alcun parametro al loro interno. In modo analogo, nella chiamata si dovranno scrivere dopo il nome della funzione solo le parentesi tonde, senza nessun argomento al loro interno.

Se si vuole definire una funzione che **non restituisce** un risultato, non si dovrà usare nel suo corpo l'istruzione `return` espressione

Definizione di funzioni: esempio

Un semplice esempio di funzione che non riceve argomenti **e** non restituisce un risultato: il suo scopo è stampare il messaggio `Buongiorno` nella *shell*.

```
def saluto() :
    print ("Buongiorno")
```

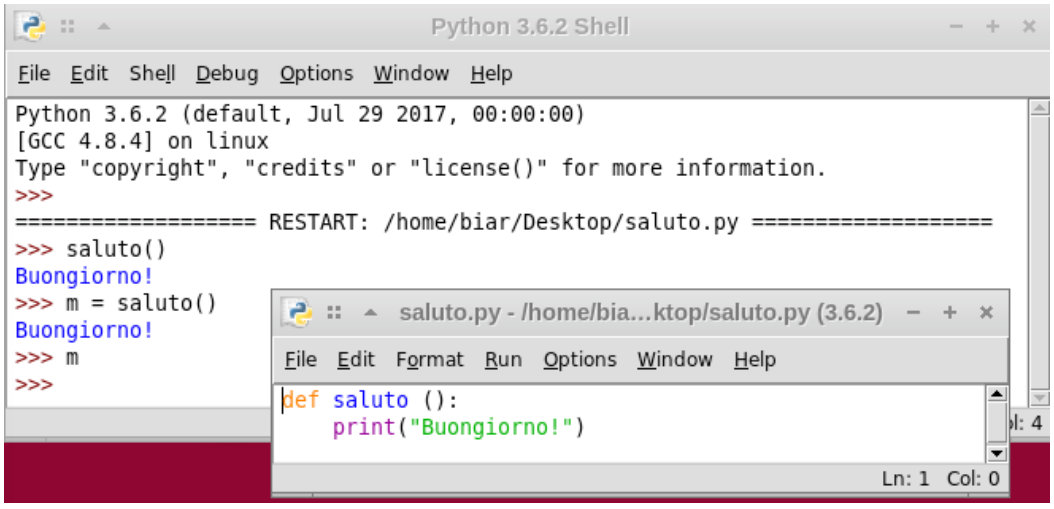
Il fatto che questa funzione non restituisca un risultato può essere evidenziato scrivendo nella *shell* una chiamata come la seguente (**dopo** aver eseguito l'istruzione `def` per definire la funzione):

```
m = saluto()
```

Si scriva poi nella *shell* l'espressione `m` per mostrare il valore della stessa variabile, come mostrato di seguito.

Definizione di funzioni: esempio

Si può vedere che l'espressione `m` non produce nessun risultato. Il motivo di ciò è che la chiamata della funzione non **restituisce** nessun valore, e quindi alla variabile `m` non viene assegnato nessun valore. Si noti che il messaggio `Buongiorno` viene **stampato** nella *shell*, ma non viene **restituito** come risultato della chiamata.

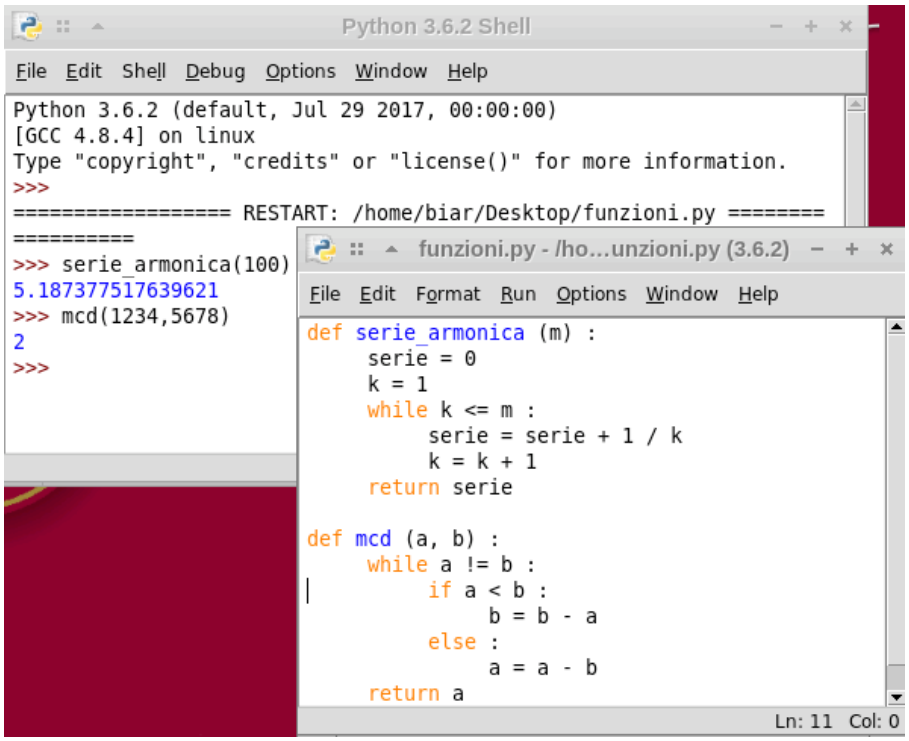


Definizione di più funzioni in uno stesso *file*

È anche possibile scrivere in uno stesso *file* la definizione di più funzioni, cioè una sequenza di istruzioni `def`.

Dopo aver eseguito il *file* (cioè, le istruzioni `def`), tutte le funzioni al suo interno saranno disponibili nella *shell*, come mostrato nell'esempio seguente.

Definizione di più funzioni in un *file*: esempio



Chiamate di funzioni all'interno di altre funzioni

Nelle istruzioni del corpo di una funzione possono comparire chiamate di altre funzioni, sia predefinite che definite dall'utente.

Se si vuole chiamare una funzione predefinita appartenente a una delle **librerie** Python (come `math` e `random`) sarà necessario inserire **prima** della chiamata la corrispondente istruzione `from-import`.

Di norma l'istruzione `from-import` viene inserita all'inizio del *file* che contiene la definizione delle proprie funzioni, **non** nel corpo di una di tali funzioni.

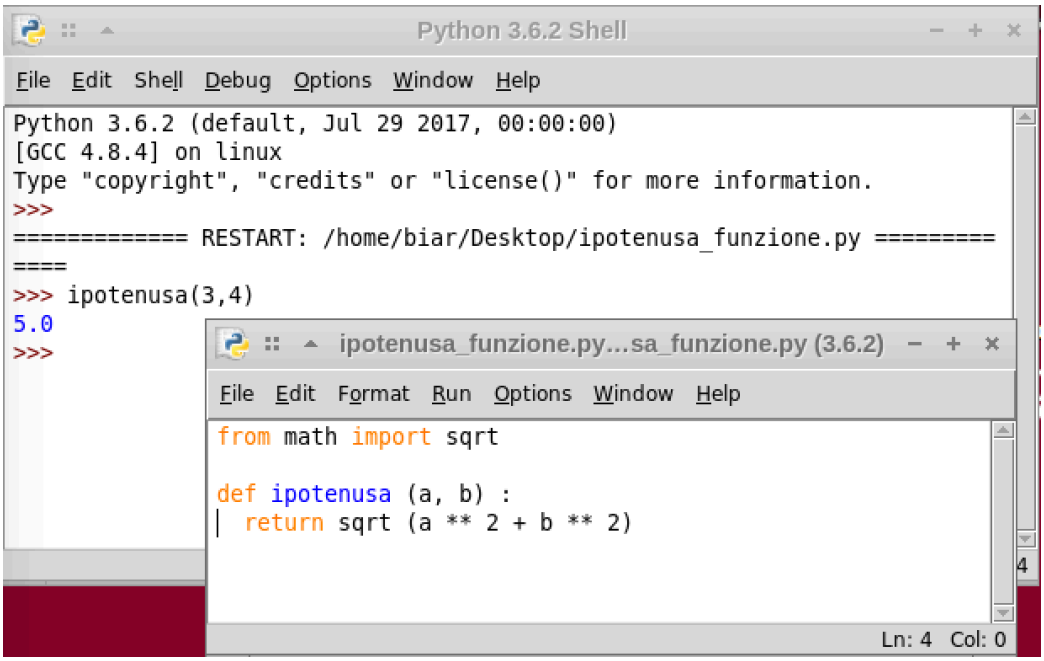
Chiamate di funzioni all'interno di altre funzioni: esempio

In questo esempio (disponibile nel *file* `ipotenusa_funzione.py`) si definisce una funzione che calcola la lunghezza dell'ipotenusa di un triangolo rettangolo, date le lunghezze dei cateti, usando la funzione `sqrt` definita nella libreria `math` (le istruzioni che seguono devono essere scritte con l'*editor* di `IDLE` in uno **stesso** *file*).

```
from math import sqrt

def ipotenusa (a, b) :
    return sqrt (a ** 2 + b ** 2)
```

Chiamate di funzioni all'interno di altre funzioni: esempio



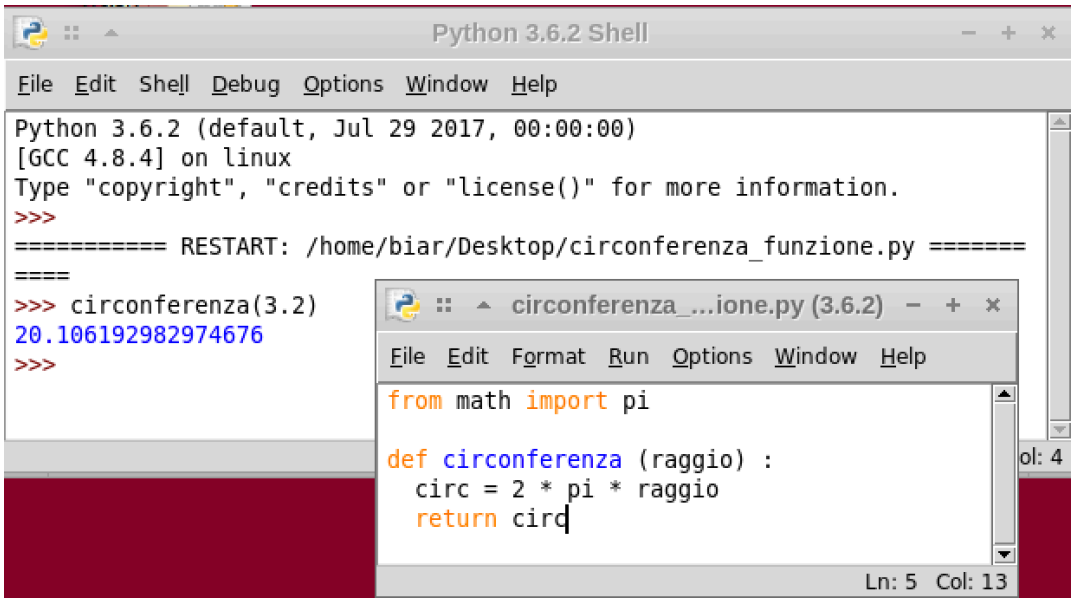
Chiamate di funzioni all'interno di altre funzioni: esempio

La funzione seguente (disponibile nel *file* `circonferenza_funzione.py`) calcola la lunghezza della circonferenza di un cerchio, dato il suo raggio, usando la variabile `pi` definita nella libreria `math`.

```
from math import pi

def circonferenza(raggio) :
    circ = 2 * pi * raggio
    return circ
```

Chiamate di funzioni all'interno di altre funzioni: esempio



Chiamate di funzioni all'interno di altre funzioni

Per poter chiamare dall'interno di una funzione un'altra funzione definita dall'utente sono disponibili due alternative:

- le **definizioni** delle due funzioni devono trovarsi nello **stesso** *file*
  oppure
- le due funzioni possono essere definite in *file* **diversi**, ma tali *file* dovranno trovarsi in una **stessa** cartella, e nel *file* che contiene la chiamata dell'altra funzione si dovrà inserire l'istruzione

`from` nome-file `import` nome-funzione

dove:

- nome-file è il nome del *file* che contiene la definizione dell'altra funzione (senza l'estensione `.py`)
- nome-funzione è il nome di tale funzione

Un esempio è mostrato nelle slide seguenti.

Chiamate di funzioni all'interno di altre funzioni: esempio

In precedenza si è visto l'esempio di un **programma che stampa nella *shell* tutti i numeri primi fino a un dato intero *n*, il cui valore viene acquisito attraverso la tastiera** (si veda il *file* `sequenzaNumeriPrimi.py`).

In questo esempio **si mostra come realizzare la stessa operazione attraverso due funzioni**.

La prima (di nome `numero_primo`) riceve come argomento un numero naturale e determina se questo sia primo, restituendo il valore `True` oppure `False`.

La seconda (`stampa_numeri_primi`) riceve come argomento il valore di *n*, scandisce tutti gli interi da *1* a *n* con un'istruzione iterativa, per ciascuno di essi chiama la funzione `numero_primo` per determinare se sia primo o meno, e stampa solo i numeri primi.

stampa numeri primi -- programma

```
# File sequenzaNumeriPrimi.py

n = int(input("Inserire un numero naturale maggiore di 1: "))
print ("I numeri primi tra 1 e", n, "sono:")
cont = 2
while cont <= n :
    primo = True
    divisore = 2
    while divisore <= cont / 2 and primo == True :
        if cont % divisore == 0 :
            primo = False
        else :
            divisore = divisore + 1
    if primo == True :
        print (cont)
    cont = cont + 1
```

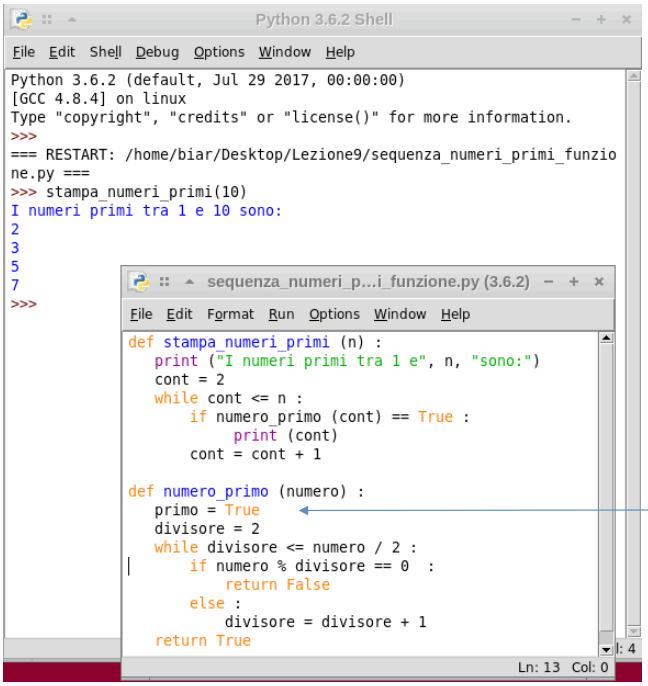
Chiamate di funzioni all'interno di altre funzioni: esempio

```
def stampa_numeri_primi (n) :
    print ("I numeri primi tra 1 e", n, "sono:")
    cont = 2
    while cont <= n :
        if numero_primo (cont) == True :
            print (cont)
        cont = cont + 1

def numero_primo (numero) :
    primo = True
    divisore = 2
    while divisore <= numero / 2 :
        if numero % divisore == 0  :
            return False
        else :
            divisore = divisore + 1
    return True
```

Chiamate di funzioni all'interno di altre funzioni: esempio

Nella prima versione, disponibile nel *file* `sequenza_numeri_primi_funzione.py`, le due funzioni vengono definite nello stesso *file*:



Non più utilizzato perché il risultato è immediatamente restituito con il return

Chiamate di funzioni all'interno di altre funzioni: esempio

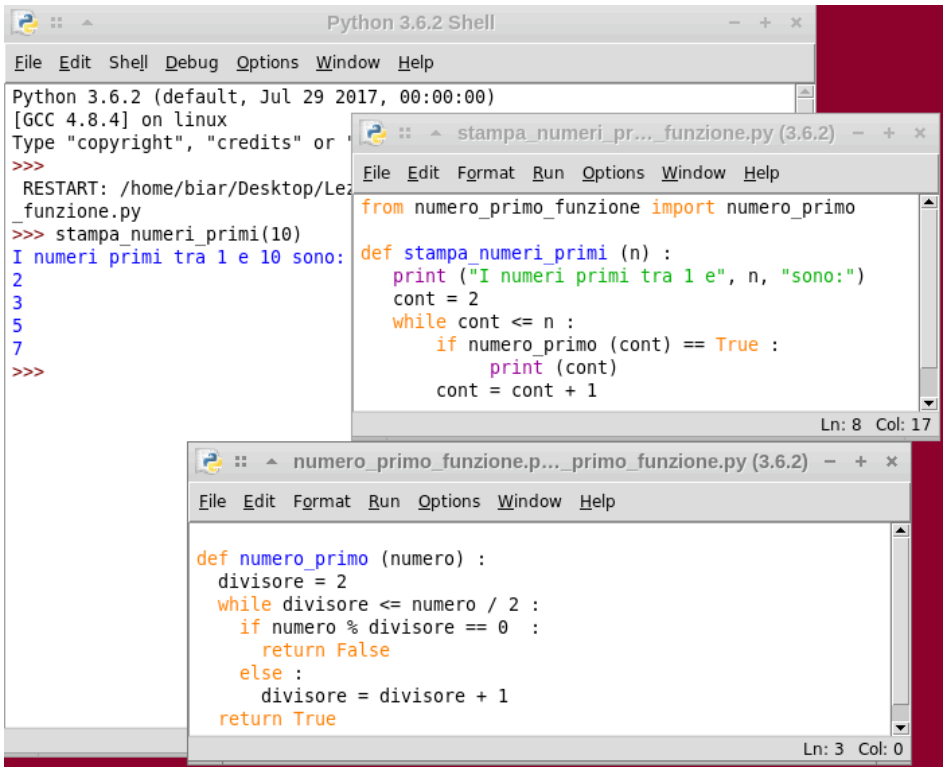
Nella seconda versione, disponibile nei *file* `stampa_numeri_primi_funzione.py` e `numero_primo_funzione.py`, **le funzioni vengono definite in due *file* diversi che devono essere memorizzati in una stessa cartella.**

Nel *file* `stampa_numeri_primi_funzione.py`, che contiene la definizione della funzione `stampa_numeri_primi`, si deve aggiungere l'istruzione

```
from numero_primo_funzione import numero_primo
```

Si noti che in questo caso prima di poter chiamare le funzioni bisogna eseguire solo le istruzioni del *file* `stampa_numeri_primi_funzione.py`, poiché la definizione della funzione contenuta nell'altro *file* viene eseguita come conseguenza dell'istruzione `from-import`.

Chiamate di funzioni all'interno di altre funzioni: esempio



Programmi che contengono definizioni di funzioni

In un *file* è anche possibile scrivere un programma (cioè una sequenza di istruzioni) **che contenga sia istruzioni `def` per la definizione di funzioni, che altre istruzioni che chiamano tali funzioni.**

L'unico vincolo è che ogni chiamata compaia **dopo** la definizione della funzione corrispondente.

Di norma, le definizioni di funzioni vengono scritte all'inizio del *file*.

Esempio

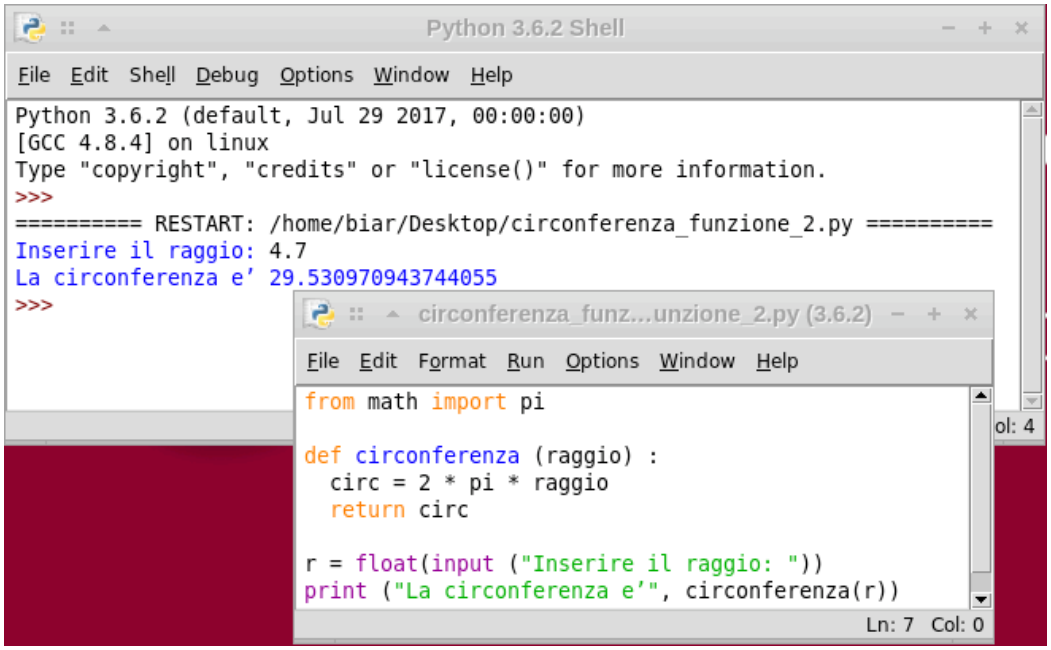
In questo programma (disponibile nel *file* `circonferenza_funzione_2.py`) si definisce la funzione `circonferenza` già vista in precedenza; le istruzioni successive acquisiscono il valore del raggio e stampano la lunghezza della circonferenza corrispondente, calcolata con una chiamata alla funzione `circonferenza`.

```
from math import pi

def circonferenza (raggio) :
    circ = 2 * pi * raggio
    return circ

r = float(input ("Inserire il raggio: "))
print ("La circonferenza e'", circonferenza(r))
```

Esempio


Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Python 3.6.2 (default, Jul 29 2017, 00:00:00)
[GCC 4.8.4] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: /home/biar/Desktop/circonferenza_funzione_2.py ==========
Inserire il raggio: 4.7
La circonferenza e' 29.530970943744055
>>>
circonferenza_funz...unzione_2.py (3.6.2)
File Edit Format Run Options Window Help
from math import pi
def circonferenza (raggio) :
circ = 2 * pi * raggio
return circ
r = float(input ("Inserire il raggio: "))
print ("La circonferenza e'", circonferenza(r))
Ln: 7 Col: 0