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 - 2

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).

Programmi che contengono definizioni di funzioni: altro esempio

```
# file cubo.py

def main() :
 r1 = volumeCubo(2)
 r2 = volumeCubo(10)
 print("Un cubo con lato di lunghezza 2 ha volume",r1)
 print("Un cubo con lato di lunghezza 10 ha volume",r2)

## funzione che calcola il volume di un cubo.
#
def volumeCubo(sideLength) :
    volume = sideLength ** 3
    return volume

# Inizio del programma.
main()
```

Funzioni: variabili *locali*

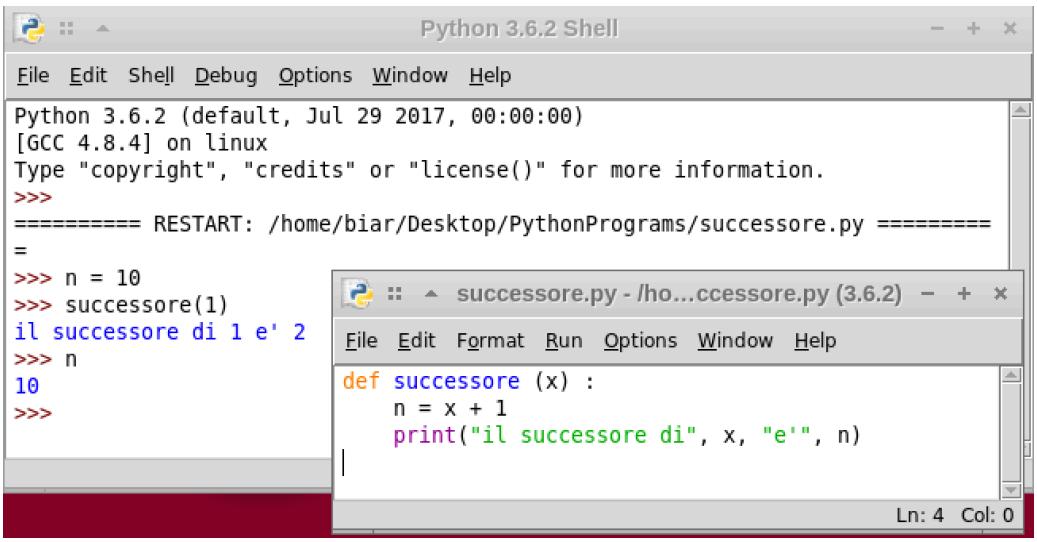
I parametri di una funzione e le eventuali altre variabili alle quali viene **assegnato** un valore all'interno di essa sono dette locali, **cioè vengono "create" dall'interprete nel momento in cui la funzione viene eseguita** (con una chiamata), **e vengono "distrutte" quando l'esecuzione della funzione termina.**

In particolare, se la chiamata della funzione viene scritta nella *shell*, e prima della chiamata è stata definita nella *shell* una variabile con lo stesso nome di una delle variabili della funzione:

- le due variabili vengono associate a celle di memoria **distinte**
- durante l'esecuzione della funzione l'interprete potrà accedere solo alla variabile associata alla funzione, e non potrà quindi usare o modificare il valore della variabile con lo stesso nome definita nella *shell*, **che manterrà il valore originale**

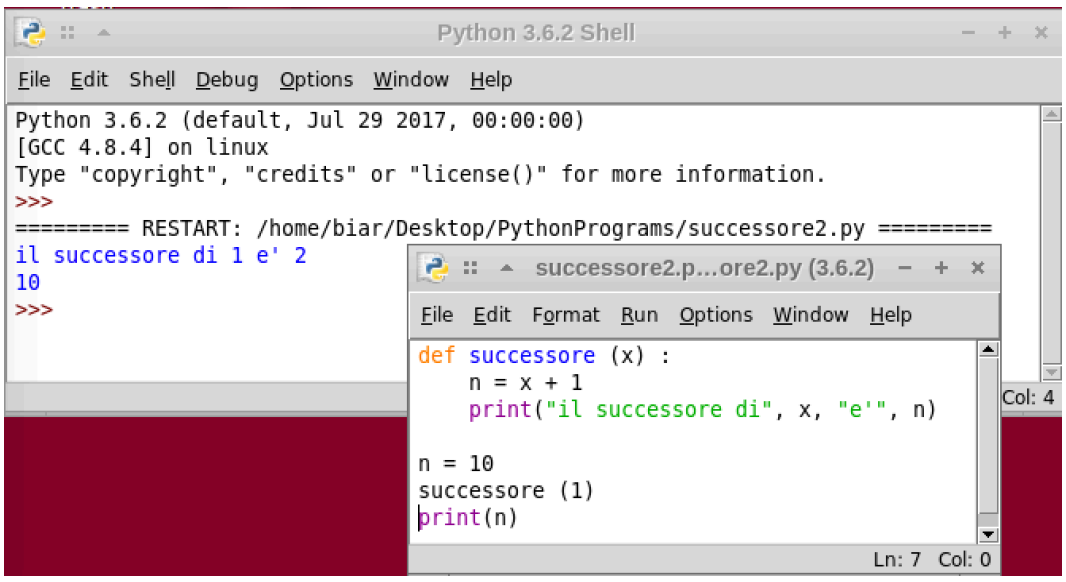
Variabili locali: esempio

In questo esempio si può osservare che il valore della variabile *n* definita nella *shell* **prima** della chiamata della funzione `successore` non viene modificato durante l'esecuzione della funzione dall'istruzione `n = x + 1` (notare che la funzione non **restituisce** nessun valore):



Variabili locali

Lo stesso accade se la chiamata della funzione si trova nelle istruzioni di un programma scritto in un *file*, rispetto alle variabili definite in tale programma, come mostra l'esempio seguente:



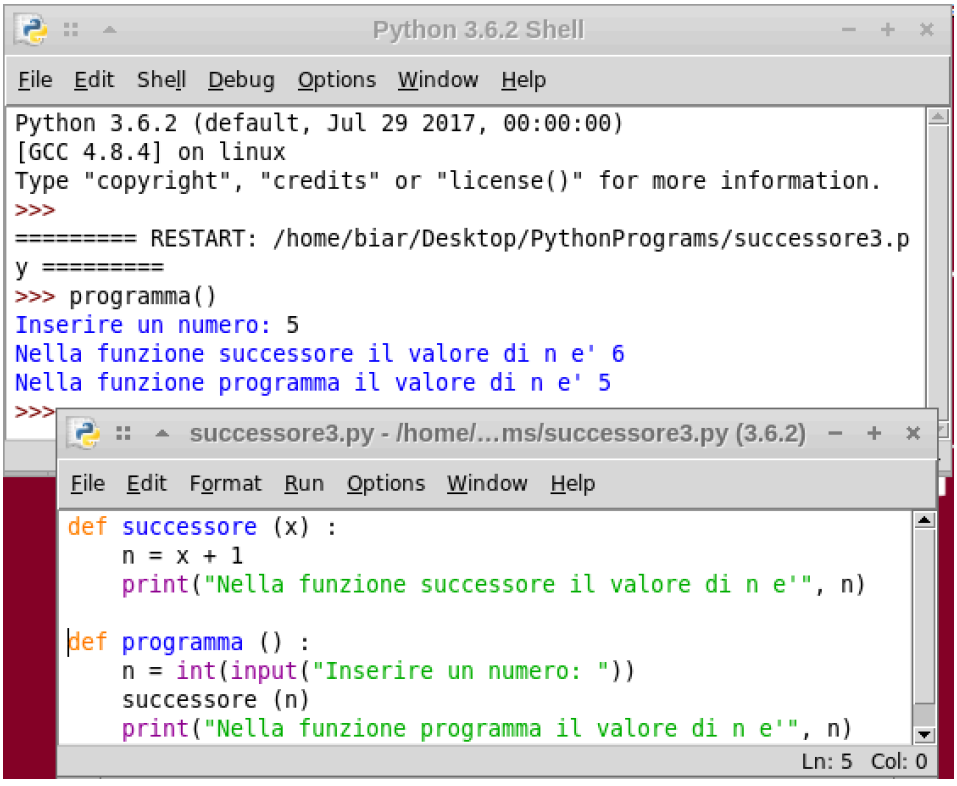
Variabili locali

Anche variabili definite in funzioni diverse e aventi lo stesso nome sono variabili locali.

Questo significa che se una funzione è chiamata da istruzioni che si trovano nel corpo di un'altra funzione, ognuna di esse potrà accedere solo alle proprie variabili, e non potrà accedere o modificare quelle dell'altra funzione.

Un esempio è mostrato di seguito.

Variabili locali: esempio



Variabili locali

Il fatto che le variabili definite in una funzione siano locali consente di definire nuove funzioni senza preoccuparsi dell'eventuale presenza di variabili con lo stesso nome nei programmi che le useranno.

Analogamente, quando si scrive un programma che chiama funzioni predefinite o definite dall'utente non ci si deve preoccupare dei nomi dei parametri e delle eventuali variabili definite in tali funzioni.

Funzioni: variabili *globali*

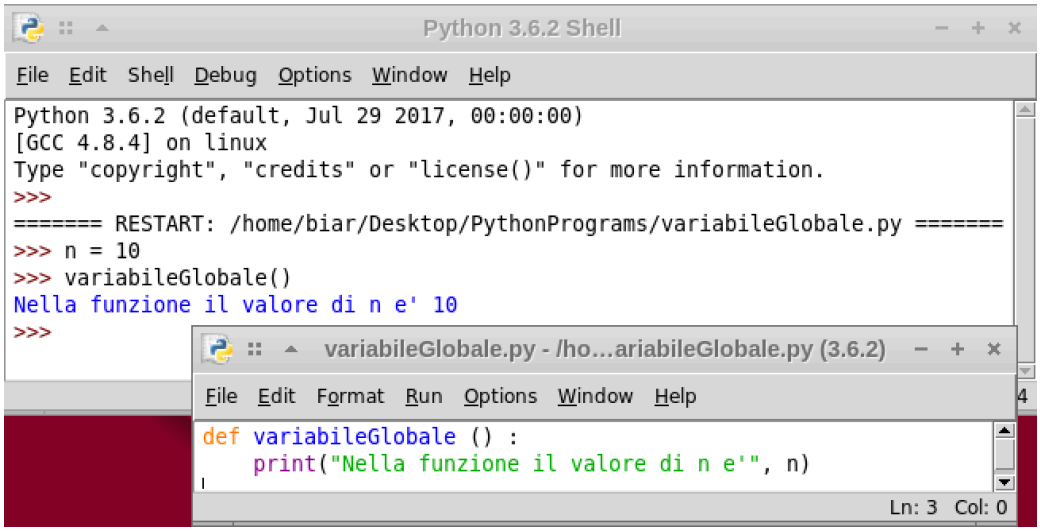
Se invece all'interno di una funzione il nome di una variabile (che non sia uno dei parametri) compare in un'espressione senza che in precedenza nella funzione sia stato **assegnato** a essa alcun valore, tale variabile è considerata globale, cioè l'interprete assume che il suo valore sia stato definito nelle istruzioni precedenti la **chiamata** della funzione (scritte nella *shell* o in un programma).

In questo modo le istruzioni di una funzione possono accede al valore di variabile definita nella *shell* o nel programma chiamante (se tale variabile non esiste si ottiene un messaggio d'errore).

In generale è preferibile **evitare** l'uso di variabili globali nelle funzioni, poiché la loro presenza rende più difficile assicurare la correttezza di un programma.

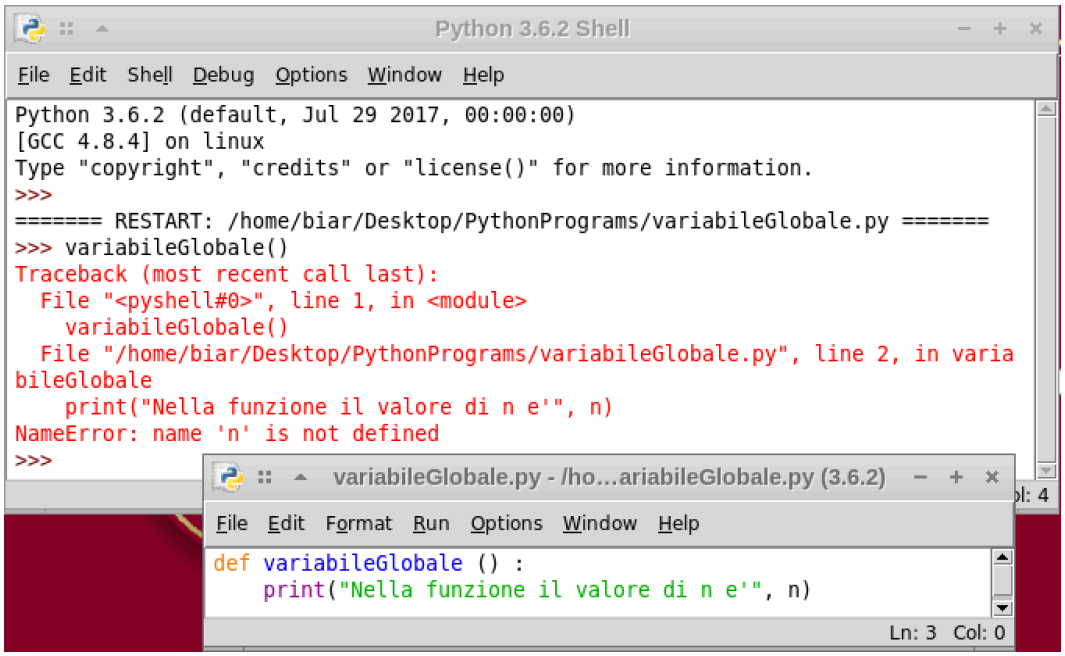
Variabili globali: esempio

Nella funzione `variabileGlobale` si fa riferimento al valore della variabile `n` senza che a essa sia stato assegnato in precedenza alcun valore all'interno della stessa funzione: in questo caso l'interprete accede alla variabile `n` già definita nella `shell`.



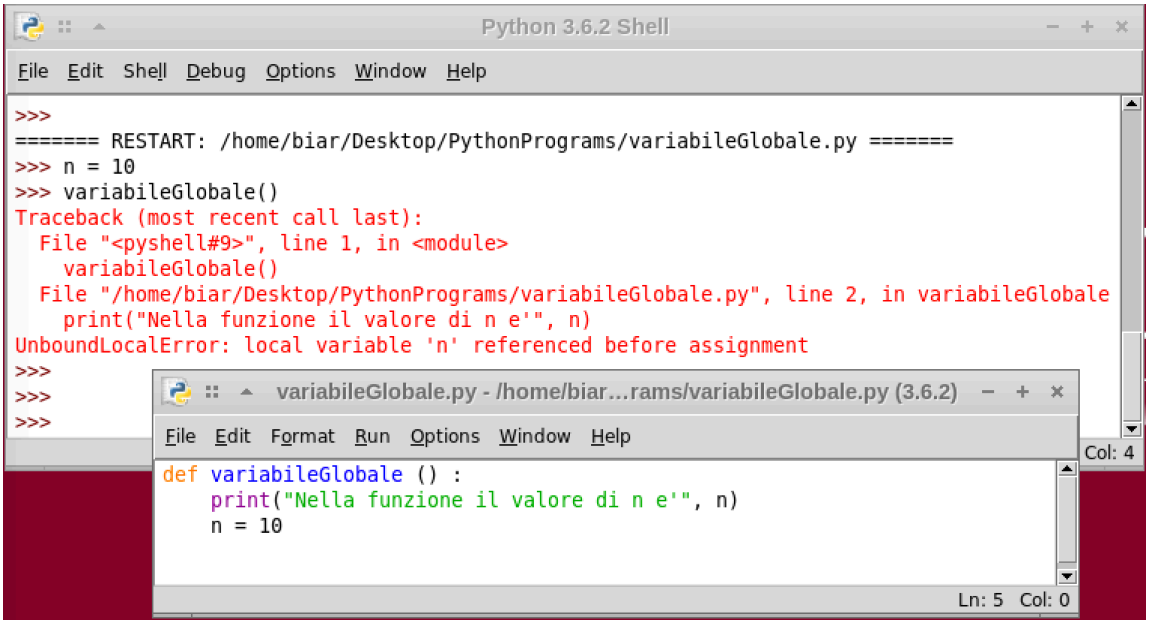
Variabili globali: esempio

Ovviamente, se nella `shell` non fosse già stata definita una variabile di nome `n` si otterrebbe un messaggio d'errore durante l'esecuzione della funzione:



Variabili locali e globali: attenzione

Se una variabile è globale, non può mai essere usata all'interno della funzione nel lato sinistro di una assegnazione. In altri termini, per la funzione una variabile globale è in sola lettura



Strutturazione dei programmi

Come si è detto in precedenza, nei linguaggi di alto livello la definizione di nuove funzioni consente di **strutturare** un programma suddividendolo in diversi "sottoprogrammi" (funzioni), ciascuno dei quali esegue un'operazione distinta e indipendente dagli altri.
Questo stile di programmazione è detto modulare.

In particolare, è buona norma cercare di suddividere un programma in funzioni che siano il più possibile **brevi**.

La programmazione modulare presenta diversi vantaggi:

- semplifica la scrittura, la comprensione e la modifica dei programmi
- rende più facile l'individuazione di eventuali errori
- consente di usare una **stessa** funzione in programmi **diversi**

Strutturazione dei programmi

Un programma Python può essere strutturato suddividendolo in una o più funzioni e in una (breve) sequenza di istruzioni da eseguire all'avvio del programma.

Le funzioni e le istruzioni del programma "principale" potranno trovarsi in uno o più *file*. Nel caso di più *file* si dovranno prevedere opportune istruzioni `from-import`.

Per eseguire il programma si dovrà eseguire il *file* che contiene le istruzioni del programma "principale".

In alternativa, anche le istruzioni del programma "principale" potranno essere scritte sotto forma di funzione. In questo caso il programma potrà essere avviato chiamando dalla *shell* tale funzione.

Strutturazione dei programmi: esempio

Si vuole scrivere un programma che acquisisca un numero naturale *n* e **stampi tutti i numeri primi compresi tra 1 e** *n*.

In precedenza si sono definite due funzioni per verificare se un numero sia primo, e per stampare tutti i numeri primi in un certo intervallo (si veda il *file* `sequenza_numeri_primi_funzione.py`).

Di seguito si mostra come l'intero programma (compresa l'acquisizione del valore di *n*) possa essere strutturato nei due modi sopra descritti.

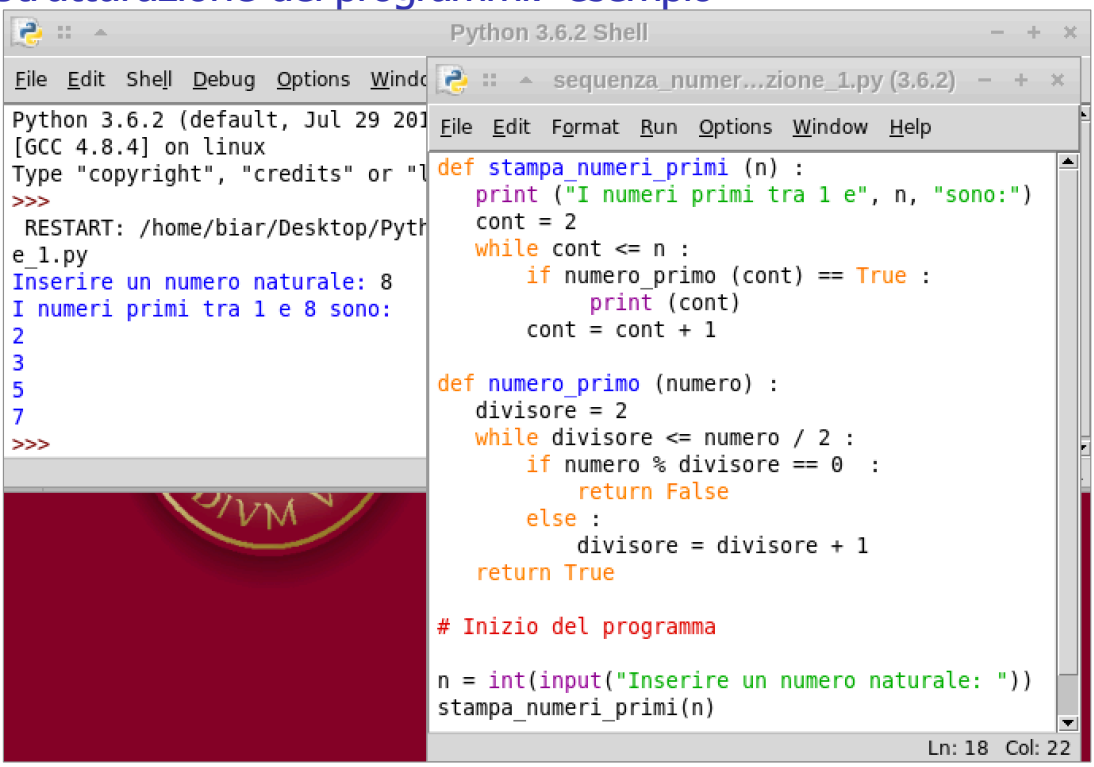
Strutturazione dei programmi: esempio

Nel *file* `sequenza_numeri_primi_programma_1.py` alle due funzioni si aggiungono le istruzioni che acquisiscono il valore di *n* e chiamano la funzione `stampa_numeri_primi`:

```
n = int(input("Inserire un numero naturale: "))
stampa_numeri_primi(n)
```

Il programma può essere avviato eseguendo il *file* che lo contiene.

Strutturazione dei programmi: esempio



Strutturazione dei programmi: esempio

Nel *file* `sequenza_numeri_primi_programma_2.py` anche le istruzioni che acquisiscono il valore di *n* e chiamano la funzione `stampa_numeri_primi` sono inserite in una funzione (`programma_principale`):

```
def programma_principale () :
    n = int(input("Inserire un numero naturale: ") )
    stampa_numeri_primi(n)
```

In questo caso per eseguire il programma si dovrà chiamare dalla *shell* la funzione `programma_principale` (dopo aver eseguito le istruzioni `def`).

Strutturazione dei programmi: esempio