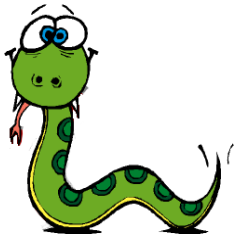
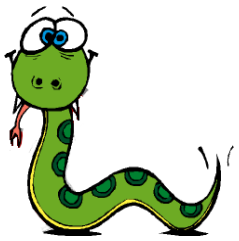


Tecnologie dell'Informazione e della Comunicazione

Capitolo 3
Funzioni

Prof. Mauro Gaspari: gaspari@cs.unibo.it

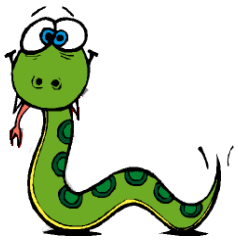
Chiamate di funzione



- Esempio di ***chiamata di funzione*** (= ***function call***).

```
>>>type("32")
<type 'string'>
```

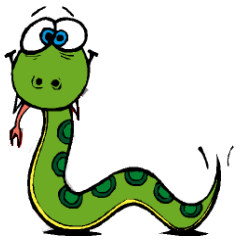
- type e' il ***nome della funzione*** e il suo risultato e' il tipo di un valore o di una variabile.
- I valori dati in ingresso a una funzione sono detti ***argomenti*** della funzione e devono essere inclusi tra parentesi (separati da virgole se sono piu' di uno).
- Di solito si dice che una funzione *prende* (= *takes*) uno o piu' argomenti e ***restituisce*** (= ***returns***) un risultato.
- Il risultato viene detto ***valore restituito*** (= ***return value***).



Esempi funzioni 1

- Invece di stampare il valore restituito è possibile assegnarlo a una variabile.

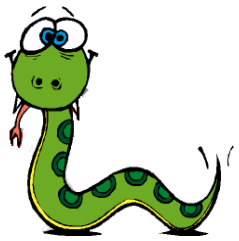
```
>>> betty = type("32")
>>> print betty
<type 'string'>
```



Esempi funzioni 2

- Funzione id: prende come argomento un valore o una variabile e restituisce un intero che rappresenta un identificatore univoco a quel valore.

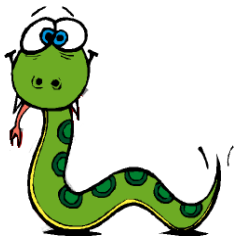
```
>>> id(3)
134882108
>>> betty = 3
>>> id(betty)
134882108
```



Conversione di Tipi

- Python fornisce una collezione di funzioni predefinite che convertono valori da un tipo ad un altro

```
>>> int("32")
32
>>> int("Hello")
ValueError: invalid literal for int(): Hello
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

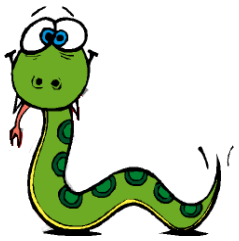


Esempi conversione di tipo

```
>>> float(32)
32.0
>>> float("3.14159")
3.14159
>>> str(32)
'32'
>>> str(3.14149)
'3.14149'
```

- NB. Python distingue tra i valori numerici 1 e 1.0 perche' questi sono rappresentati in modo diverso sul computer.

Coercion



- Problema divisione intera:

  minutes/60 da risultato 0 anche se si minutes=59

- Una possibile soluzione e' quella di convertire minutes in float:

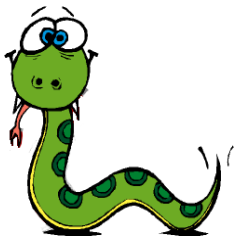
```
>>> minute = 59
>>> float(minute) / 60.0
0.983333333333
```

- In alternativa si possono utilizzare delle regole di conversione di tipo automatica che si indicano con il nome di ***coercion***

- Ad esempio per gli operatori matematici se uno degli operatori e' un float l'atro viene convertito automaticamente in float:

```
>>> minute = 59
>>> minute / 60.0
0.983333333333
```

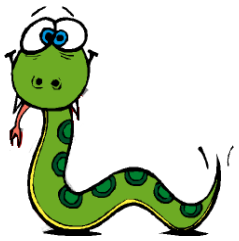
Funzioni e espressioni



- Le funzioni possono apparire anche nelle espressioni e vengono valutate non appena si incontrano, tenendo presente le regole della precedenza tra operatori.
- Le funzioni possono avere come argomento anche espressioni che contengono altre chiamate di funzione. In tal caso vengono prima valutate le funzioni piu' interne.
- La regola di valutazione che usa python si chiama ***interna sinistra***: per valutare una funzione si valutano i suoi argomenti partendo da quello interno piu' a sinistra (se ce ne e' piu' di uno).

```
>>>str(56*76)
'4256'
```

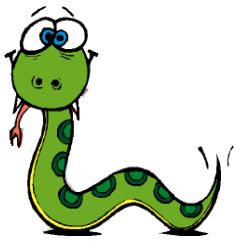
Funzioni Matematiche



- Oltre alle funzioni matematiche di base, python ha una buona libreria di funzioni matematiche.
- Ad esempio: log, sin.
- NB. pero' queste funzioni non sono accessibili direttamente in python e' necessario caricare il modulo che le contiene, con il comando che segue:

```
>>> import math
```

Funzioni e moduli



- Per chiamare una delle funzioni di un modulo e' necessario inserire il nome del modulo prima del nome della funzione separato da un punto (= ***dot notation***).
- NB. le funzioni (con, tan,...etc...) lavorano con radianti.

```
>>> decibel = math.log10 (17.0)
>>> angle = 1.5
>>> height = math.sin(angle)

>>> degrees = 45
>>> angle = degrees * 2 * math.pi / 360.0
>>> math.sin(angle)
0.707106781187
```

Da gradi a radianti

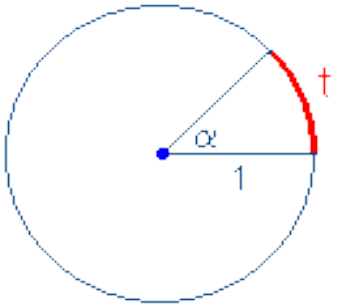
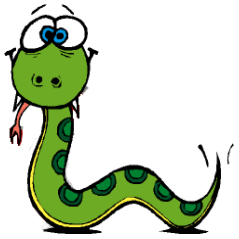
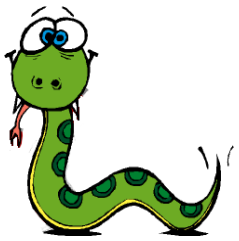


Fig. 1

Gradi e radianti



- Ad ogni angolo al centro a corrisponde un arco di circonferenza t.
- Tra angoli e archi c'è una relazione di proporzionalità diretta: se si raddoppia l'angolo si raddoppia anche l'arco, se si triplica l'angolo si triplica anche l'arco, ecc. (e viceversa).
- Possiamo quindi assumere come misura dell'angolo la misura dell'arco corrispondente.
- Parleremo in questo caso di misura dell'angolo in **radianti**.
- Ad esempio ad un angolo di 90° corrisponde un arco di lunghezza p/2; quindi la misura in radianti di un angolo di 90° è p/2.
- In generale possiamo scrivere la proporzione a : t = 360 : 2p



Composizione di funzioni

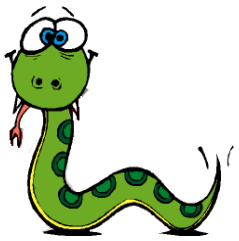
- Abbiamo gia' detto che una qualsiasi espressione puo' essere data come argomento ad una funzione.

```
>>> x = math.cos(angle + math.pi/2)
```

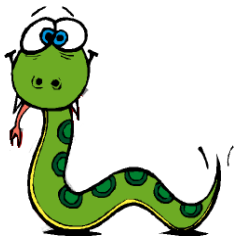
- NB. un espressione puo' essere anche il risultato di un altra funzione:

```
>>> x = math.exp(math.log(10.0))
```

Nuove funzioni



- La possibilita' di definire nuove funzioni per risolvere problemi particolari e' una delle caratteristiche piu' utili dei linguaggi di programmazione.
- Una funzione e' in genere una sequenza di comandi, a cui viene dato un nome, che esegue una certa operazione.
- Tutte le funzioni che abbiamo visto fino ad ora sono predefinite in python. In genere queste funzioni si chiamano ***primitive***.
- Le funzioni primitive si possono utilizzare senza conoscere nel dettaglio come sono realizzate. E' sufficiente conoscere la loro semantica.

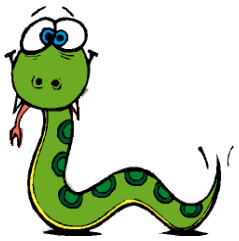


Definizione di funzione

- Sintassi:

```
def NAME( LIST OF PARAMETERS ):
    STATEMENTS
```

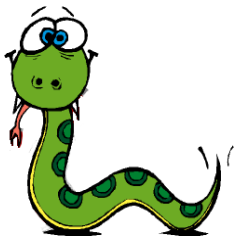
- I nomi delle funzioni hanno la stessa sintassi dei nomi delle variabili, si puo' utilizzare un qualsiasi nome ad eccezione delle parole chiave.
- La lista di parametri specifica le informazioni che vanno passate per poter utilizzare una funzione.



Definizioni di funzioni

- Le definizioni di funzione iniziano sempre con il def.

```
>>> def hello():
...     print "Hello, how are you?"
...
>>> hello()
Hello, how are you?
>>>
```

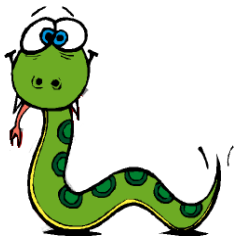


Il nome della funzione

Il nome della funzione e' ‘hello’

```
>>> def hello():
...     print "Hello, how are you?"
...
>>> hello()
Hello, how are you?
>>>
```

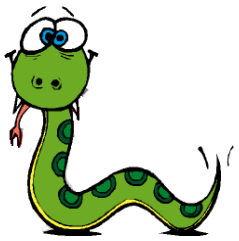
Le funzioni permettono di associare un nome ad una sequenza di comandi.



I parametri

sono sempre tra parentesi.
si tratta di una funzione senza parametri

```
>>> def hello():
...     print "Hello, how are you?"
...
>>> hello()
Hello, how are you?
>>>
```

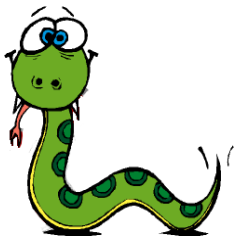


Attenzione ai “:”

La definizione di una funzione deve terminare con i duepunti (“:”)

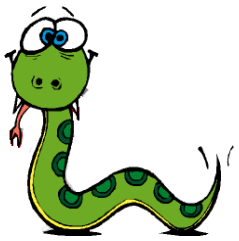
```
>>> def hello():
...     print "Hello, how are you?"
...
>>> hello()
Hello, how are you?
>>>
```

Il codice o ***corpo*** (= ***body***)



E' la lista dei comandi che vengono eseguiti quando una funzione viene chiamata, va bene un qualsiasi comando python.

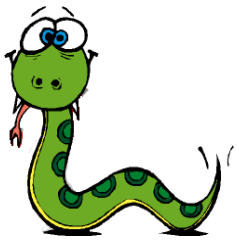
```
>>> def hello():
...     print "Hello, how are you?"
...
>>> hello()
Hello, how are you?
>>>
```



Chiamate di funzione

Quando si chiama una funzione si chiede a Python di eseguire i comandi del suo corpo.

```
>>> def hello():
...     print "Hello, how are you?"
...
>>> hello()
Hello, how are you?
>>>
```

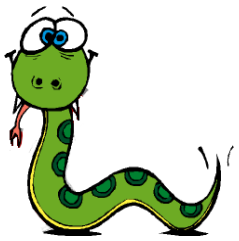


Come chiamare una funzione

La ***chiamata*** (= ***function call***) inizia con il nome della funzione in questo caso e' "hello"

```
>>> def hello():
...     print "Hello, how are you?"
...
>>> hello()
Hello, how are you?
>>>
```

NB. La definizione di una funzione non genera output in python (in altri linguaggi interpretati puo' generarlo: in LISP restituisce il nome).

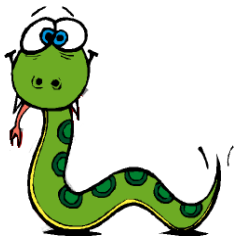


Passare gli argomenti

Gli argomenti vengono sempre passati tra parentesi, si tratta di una funzione senza argomenti (la lista e' vuota).

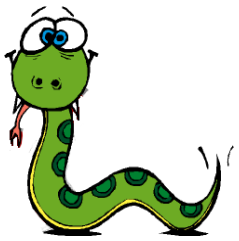
```
>>> def hello():
...     print "Hello, how are you?"
...
>>> hello()
Hello, how are you?
>>>
```

Esecuzione della funzione



```
>>> def hello():
...     print "Hello, how are you?"
...
>>> hello()
Hello, how are you?
>>>
```

NB. Una funzione deve essere definita prima di essere chiamata e quindi eseguita. Bisogna prima eseguire (valutare) la sua definizione.



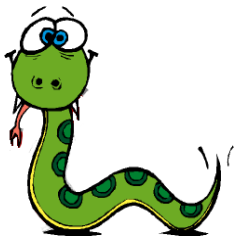
Esempi funzioni

```
def newLine():
  print

print "First Line."
newLine()
print "Second Line."
```

Cosa succede se l'eseguo?

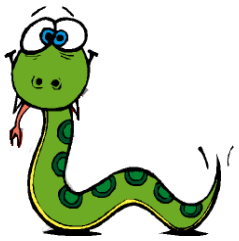
```
def threeLines():
  newLine()
  newLine()
  newLine()

print "First Line."
threeLines()
print "Second Line."
```

Come mai python non include l'istruzione print nella funzione?



Osservazioni

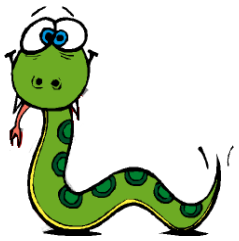
- La definizione di funzione termina appena si trova un comando non indentato.
- E' possibile chiamare più volte la stessa funzione.
- Una funzione se necessario può chiamare un altra funzione.
- Le funzioni permettono di semplificare i programmi perche' computazioni complesse possono essere definite in una funzione ed invocate solo con il nome della funzione.
- In questo modo si possono anche eliminare ripetizioni.



Flusso di esecuzione

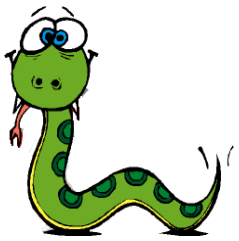
- Per capire se una funzione e' effettivamente definita prima di essere utilizzata e' necessario avere le idee chiare su l'ordine di esecuzione dei comandi in un programma.
- L'ordine dei comandi è determinato dal **flusso di esecuzione**.
  - L'esecuzione parte sempre dal primo comando di un programma (all'inizio del file).
  - I comandi sono eseguiti uno alla volta in ordine dall'alto verso il basso.

Funzioni e flusso di esecuzione

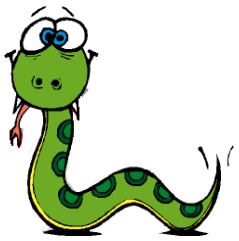


- Le definizioni di funzione non alterano il flusso di esecuzione di un programma.
- Quando si incontra una definizione di funzione questa viene definita, ovvero si associa al nome della funzione la sequenza di comandi che la compone.
- Una volta definita la funzione può essere chiamata.
- I comandi che la compongono sono effettivamente seguiti solo al momento della chiamata.
- NB. è possibile definire una funzione all'interno di un altra funzione.

Esecuzione di una funzione

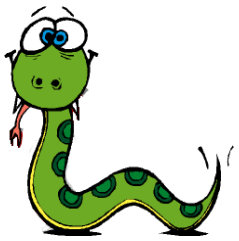


- Cosa accade al flusso di controllo quando raggiunge una chiamata di funzione?
  - La funzione viene chiamata e il flusso di controllo comincia ad eseguire i comandi della funzione.
  - Quando i comandi della funzione sono terminati il flusso di controllo torna al comando che segue la chiamata.
- Per capire come funziona un programma e' importante seguire il flusso di esecuzione:
  - di solito si usa un ***program counter*** per indicare l'istruzione corrente.



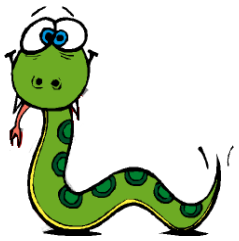
Esempio flusso di esecuzione

```
def newLine():
  print

def threeLines():
  newLine()
  newLine()
  newLine()

print "First Line."
threeLines()
print "Second Line."
```



Parametri e argomenti

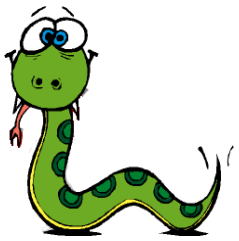
- Abbiamo visto che alcune delle funzioni richiedono argomenti. questo e' naturale, ad esempio per calcolare il seno di un certo numero e' necessario dare alla funzione quel numero.
- Ci sono funzioni che necessitano di piu' argomenti:
  `pow(base,esponente)`
- Gli argomenti che vengono passati ad una funzione vengono assegnati a delle variabili presenti nel corpo della funzione che si chiamano ***parametri*** (= ***parameters***) .



Esempio: definizione

Questa funzione ha un unico parametro. Al momento della chiamata il parametro sara' accessibile tramite la variabile ***nome***

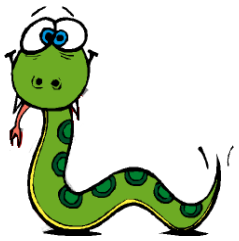
```
>>> def ciao(nome):
...     print "Ciao", nome
...
>>> ciao("Mauro")
Ciao Mauro
>>>
```



Esempio: chiamata

La chiamata di funzione ha bisogno di un argomento. Si usa la stringa "Mauro".

```
>>> def ciao(nome):
...     print "Ciao", nome
...
>>> ciao("Mauro")
Ciao Mauro
>>>
```

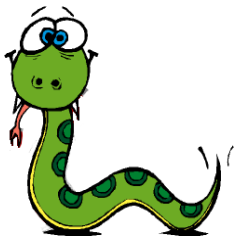


Esempio: esecuzione

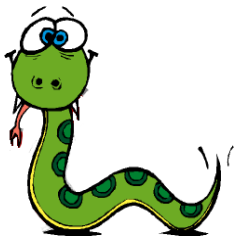
Al momento della chiamata si assegna la stringa “Mauro” alla variabile (nome), dopo di che, si esegue il comando nel body.

```
>>> def ciao(nome):
...     print "Ciao", nome
...
>>> ciao("Mauro")
Ciao Mauro
>>>
```

Esempio di funzione con un parametro



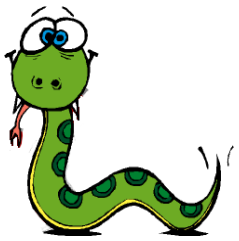
```
def printTwice(bruce):
  print bruce, bruce


>>> printTwice('Spam')
Spam Spam
>>> printTwice(5)
5 5
>>> printTwice(3.14159)
3.14159 3.14159
```



Osservazioni

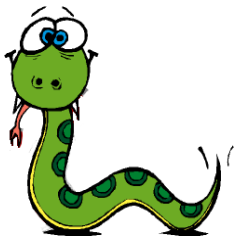
- La funzione printTwice
  - ha un solo argomento assegnato ad un parametro di nome bruce.
  - funziona con qualsiasi tipo di argomento.
- La composizione tra funzioni si puo' applicare anche alle funzioni definite dall'utente.

```
>>> printTwice('Spam'*4)
SpamSpamSpamSpam SpamSpamSpamSpam
>>> printTwice(math.cos(math.pi))
-1.0 -1.0
```



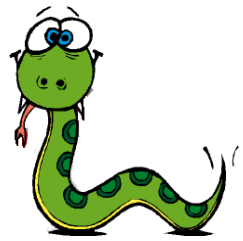
Regola valutazione funzioni

- NB. In python le espressioni che vengono passate alle funzioni vengono valutate prima di eseguire la funzione.
- Questa regola si chiama: ***valutazione per valore:*** gli argomenti vanno valutati prima di applicare la funzione.
- Quindi se uno dei parametri e' una variabile questa viene valutata e quindi non e' possibile accedere a quella variabile (ad esempio per assegnargli qualcosa) quando si esegue il codice della funzione.



Esempio

```
>>> michael = 'Pippo mangia pluto.'
>>> printTwice(michael)
Pippo mangia pluto. Pippo mangia pluto.
```

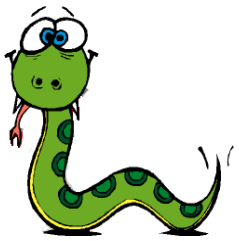


Esempio con due parametri

Due parametri nella definizione

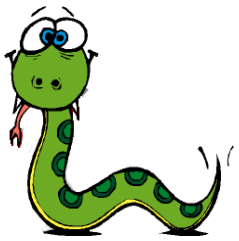
```
>>> def subtract(x, y):
...     print x-y
...
>>> subtract(8, 5)
3
>>>
```

Due parametri nella chiamata



Domande

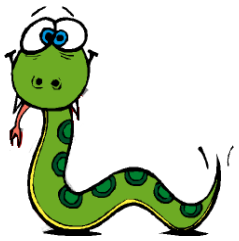
- Cosa succede se si chiama una funzione e non si fa niente con il risultato?
- Cosa succede se si utilizza una funzione senza risultato in un espressione?
  `newline() + 7`



Valore di default

Se non si indica altrimenti una funzione per
default restituisce il valore: `None`

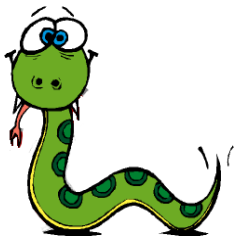
```
>>> def subtract(x, y):
...     print x-y
...
>>> x = subtract(8, 5)
3
>>> print x
None
>>>
```



NB. non dipende dalla print

```
>>> def subtract(x, y):
...     x-y
...
>>> x = subtract(8, 5)
>>> print x
None
>>>
```

Anche se l'ultimo comando di una funzione restituisce qualcosa la funzione restituisce None.



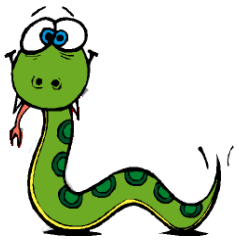
Come restituire valori?

Il comando return dice a Python di uscire da una funzione e di restituire un certo valore.

```
>>> def subtract(x, y):
...     return x-y
...
>>> x = subtract(8, 5)
>>> print x
3
>>>
```

In genere il valore restituito puo' essere di un tipo qualsiasi.

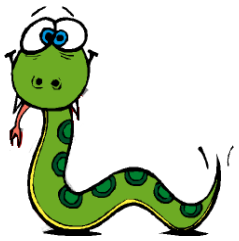
Variabili locali e parametri



- Quando si crea una variabile locale in una funzione questa esiste solo nella funzione non si puo' utilizzare al di fuori.

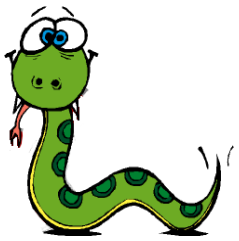
```
        def catTwice(part1, part2):
          cat = part1 + part2
          printTwice(cat)
>>> chant1 = "Pie Jesu domine, "
>>> chant2 = "Dona eis requiem."
>>> catTwice(chant1, chant2)
Pie Jesu domine, Dona eis requiem. Pie Jesu domine,
                                    Dona eis requiem.
>>>print cat
Name error: cat
```

- NB. Anche i parametri sono locali. Non e' quindi possibile utilizzarli all'interno di funzioni.

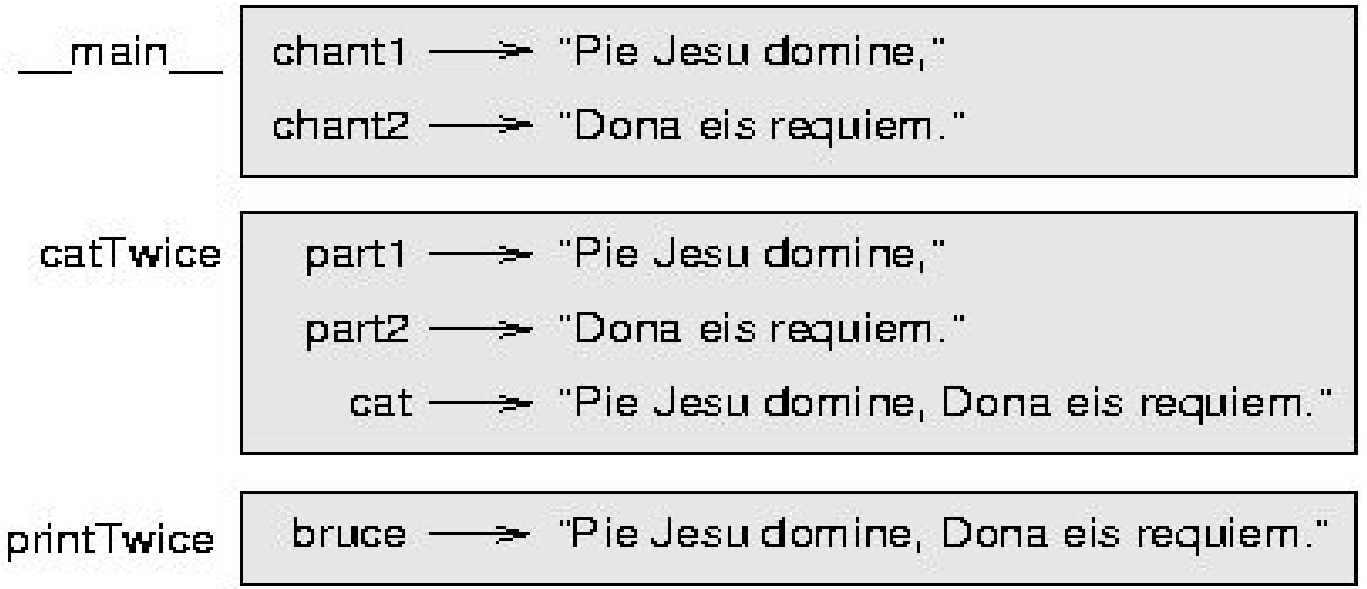


Diagrammi a stack

- I diagrammi a stack sono simili ai diagrammi di stato e tengono conto anche delle funzioni a cui le variabili appartengono.
- Ad ogni funzione puo' essere associato un frame.
- Un ***frame*** e' un contenitore a cui e' associato il nome di una funzione e che contiene i parametri e le variabili che appaiono in quella funzione.
- Un ***diagramma a stack*** e' costituito dai frame delle funzioni chiamate in un certo momento che non hanno ancora terminato l'esecuzione.

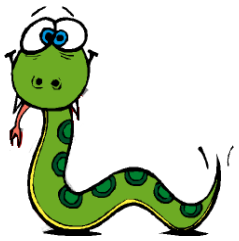


Esempio di diagramma a stack



- Il diagramma a stack mostra il flusso di esecuzione.
- Le variabili al di fuori di ogni funzione sono dette ***globali***, assumiamo che appartengono al diagramma __main__.

Diagrammi a stack e debugging



- Se si verifica un errore in una delle funzioni. ad esempio se si prova ad accedere alla variabile `cat` dalla funzione printTwice, si ottiene un errore di nome.
- NB. Python presenta anche la ***traccia*** delle chiamate di funzione.

```
Traceback (innermost last):
  File "test.py", line 13, in __main__
    catTwice(chant1, chant2)
  File "test.py", line 5, in catTwice
    printTwice(cat)
  File "test.py", line 9, in printTwice
    print cat
NameError: cat
```