Fondamenti di Informatica e Basi di Dati a.a. 2019/2020

DOCENTE: DOTT.SSA VALERIA FIONDA

basate sul materiale del Prof. **Mauro Gaspari**

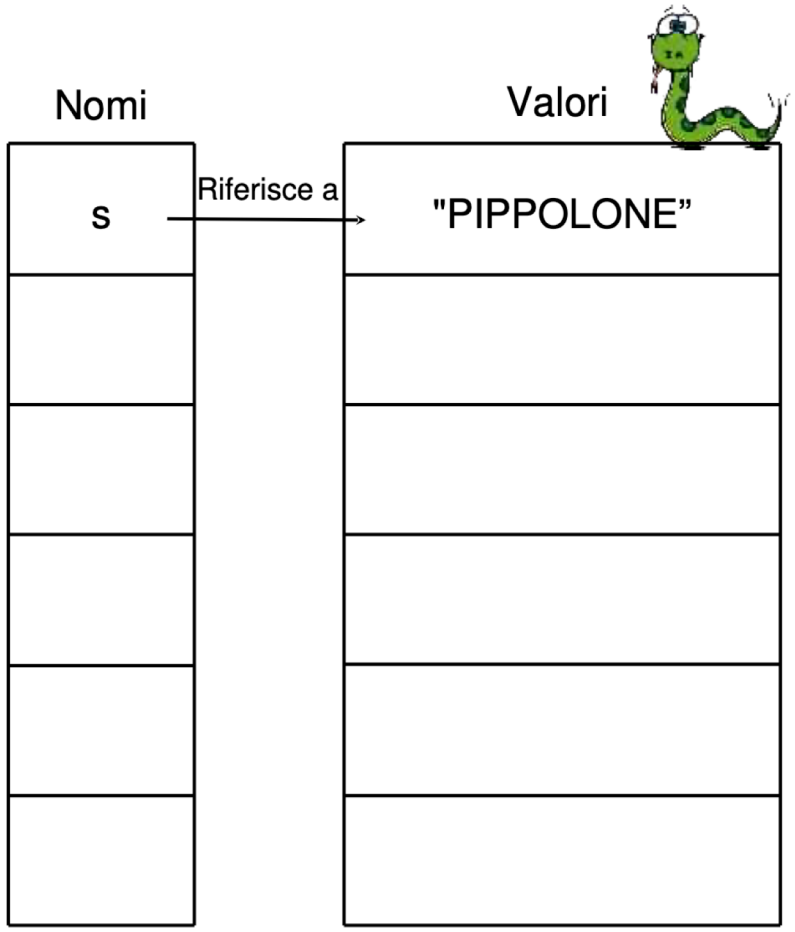
Introduzione alla programmazione con python

PYTHON

Variabili, Espressioni e Comandi

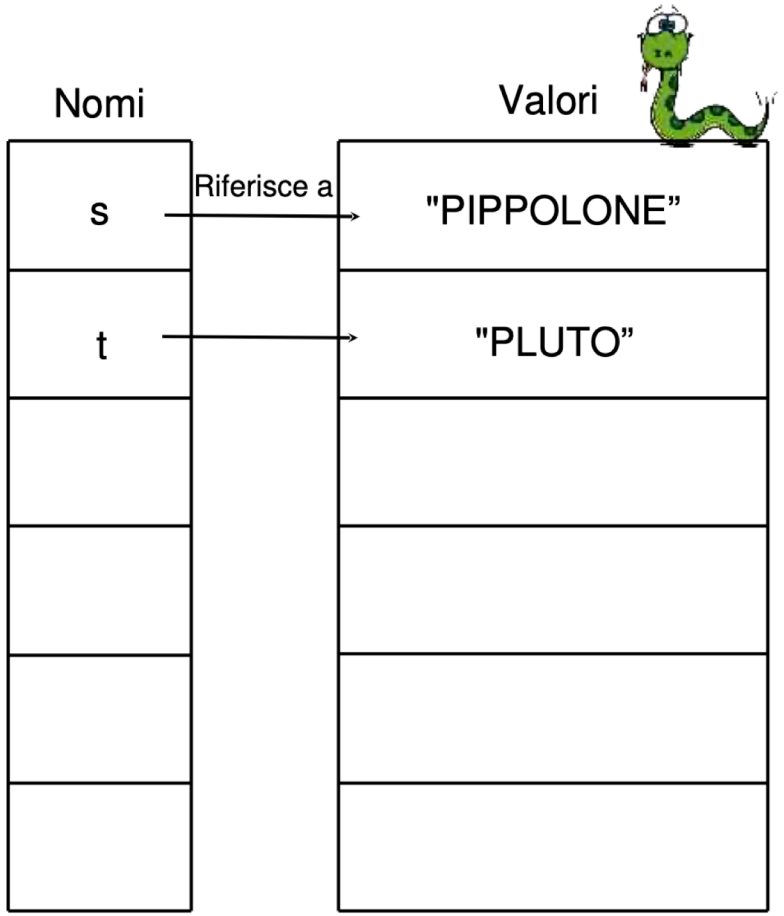
Un esempio

```
>>> s = “PIPPOLONE”
>>>
```



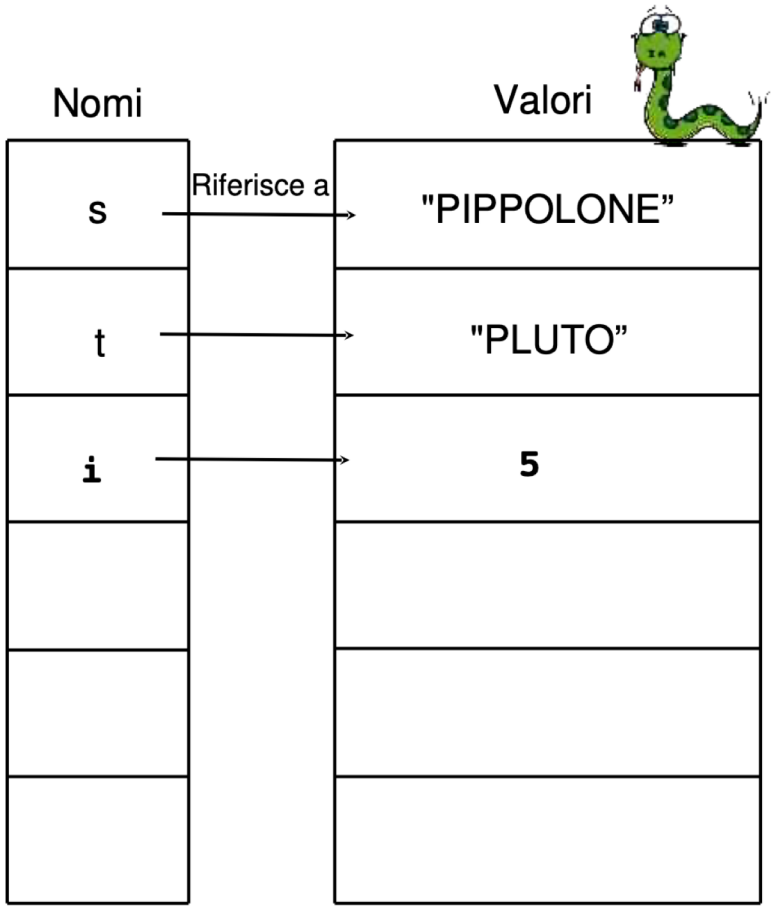
Un esempio

```
>>> s = “PIPPOLONE”
>>> t = “pluto”
>>>
```



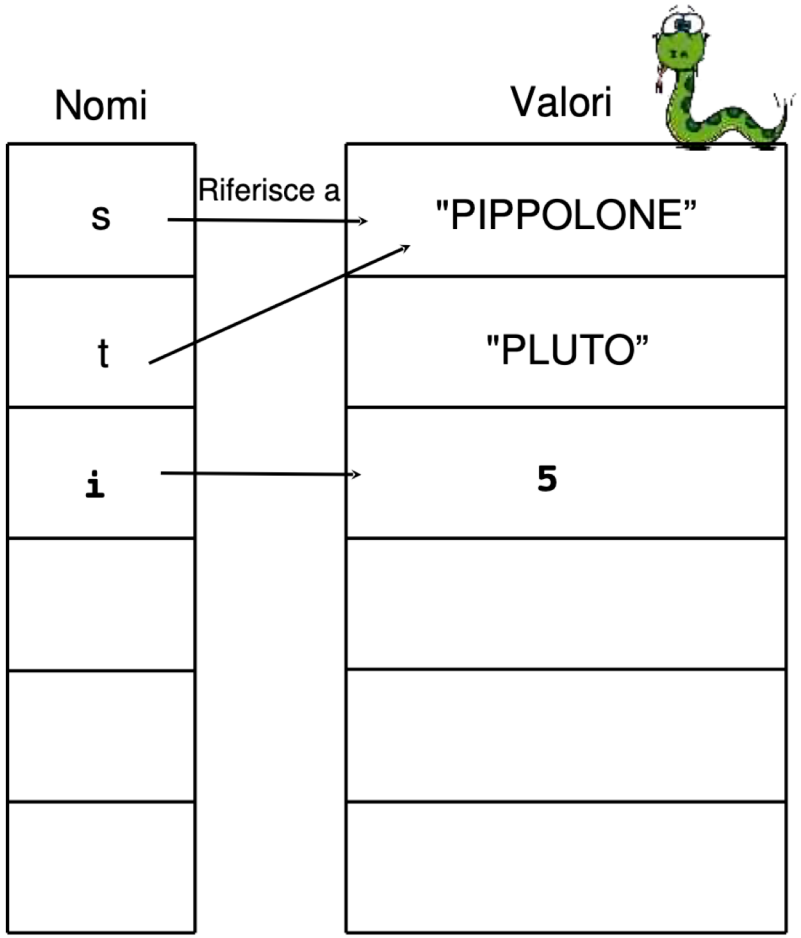
Un esempio

```
>>> s = “PIPPOLONE”
>>> t = “PLUTO”
>>> i = 5
>>> print i
5
>>>
```



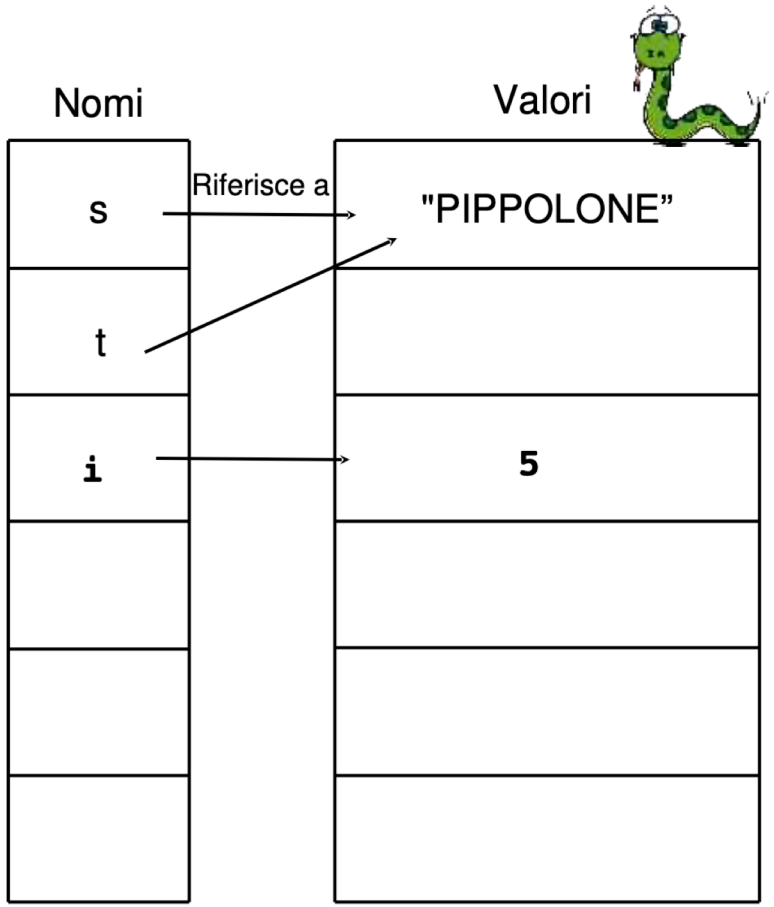
Un esempio

```
>>> s = “PIPPOLONE”
>>> t = “PLUTO”
>>> i = 5
>>> print i
5
>>> t = s
```



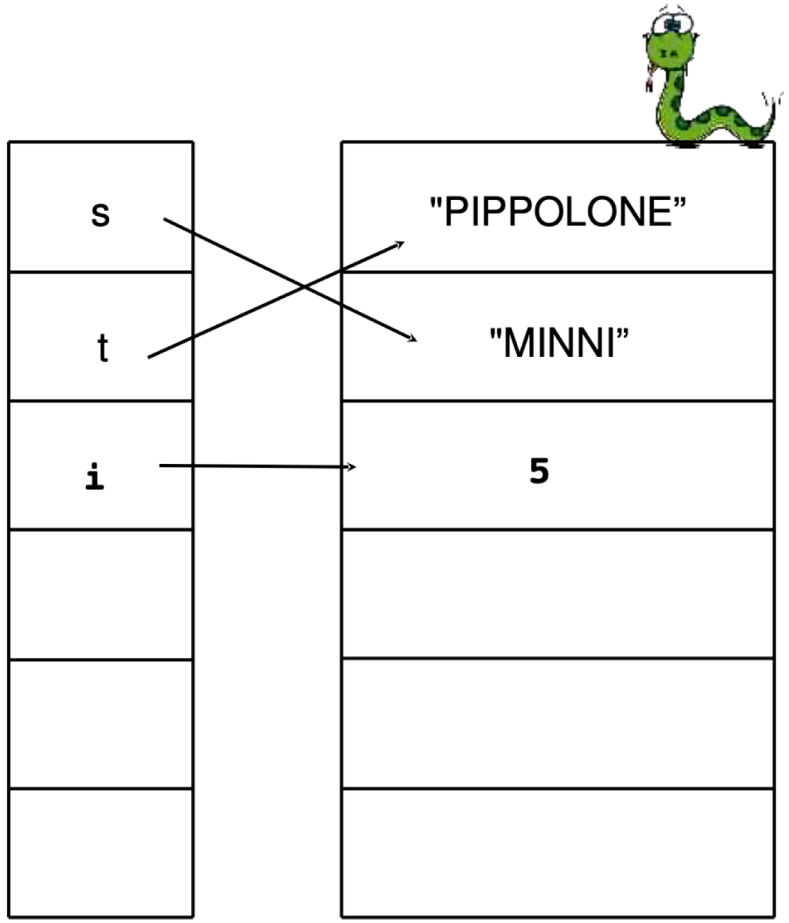
Un esempio

```
>>> s = “PIPPOLONE”
>>> t = “PLUTO”
>>> i = 5
>>> print i
5
>>> t = s
>>> print a
Traceback (most......
  File "<stdin>", line.
NameError: name 'a' is
not defined
>>>
```



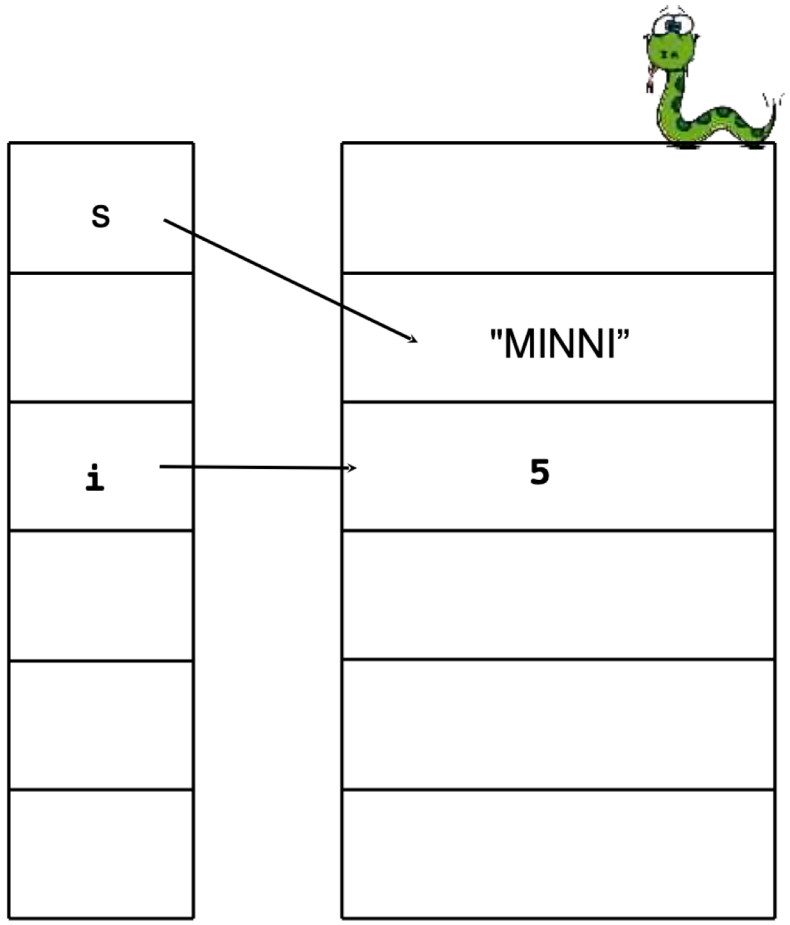
Un esempio

```
>>> s = “PIPPOLONE”
>>> t = “PLUTO”
>>> i = 5
>>> print i
5
>>> t = s
>>> print a
Traceback (most......
  File "<stdin>", ....
NameError: name 'a' is
not defined
>>> S = “MINNI”
>>> print t
PIPPOLONE
```



Un esempio

```
>>> s = “PIPPOLONE”
>>> t = “PLUTO”
>>> i = 5
>>> print i
5
>>> t = s
>>> print a
Traceback (most......
  File "<stdin>", ....
NameError: name 'a' is
not defined
>>> S = “MINNI”
>>> print t
PIPPOLONE
>>> del t
>>> print t
Traceback (most......
  File "<stdin>", ....
NameError: name 't' is
not defined
```



Commenti

- Quando i programmi diventano piu' lunghi e strutturati a volte diventa difficile, guardando un pezzo di codice, capire cosa questo fa’.
- Utile in fase di debugging o di aggiornamento del codice.
- E' una prassi comune aggiungere al codice commenti (in genere tutti i linguaggi di programmazione ad alto livello lo permettono).
- In python i commenti iniziano con il simbolo #.
- Tutto quello che segue il simbolo # viene ignorato fino alla fine della riga dove si trova.

Commenti

Tipologie di commenti:

- documentare il significato delle operazioni;
- segnalare possibili fonti di errori;
- segnalare modifiche al codice.

Esempio commenti

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60

percentage = (minute * 100) / 60 # att. divisione int
```

Alcuni operatori utili

- L'operatore modulo opera su numeri interi e in generale su espressioni intere. Restituisce il resto della divisione intera.

```
>>> quotient = 7 / 3
>>> print quotient
2
>>> remainder = 7 % 3
>>> print remainder
1
```

- Si tratta di un operatore molto utile ad esempio per verificare se un numero e' divisibile per un altro:
se x%y da risultato 0 allora significa che x e' divisibile per y.

Espressioni Booleane

- Un *espressione booleana* e' un espressione che puo' essere vera o falsa.
- In python c'e' la seguente convenzione (comune a molti linguaggi di programmazione):
  - FALSO ha valore 0
  - VERO ha valore 1 oppure qualsiasi oggetto diverso da 0, anche un float o una stringa
- Ad esempio l'operatore == confronta due valori e produce un’espressione booleana.

```
>>> 5 == 5
1
>>> 5 == 6
0
```

Altri operatori di confronto

```
x != y          # x e' diverso da y
x > y           # x e' maggiore di y
x < y           # x e' minore di y
x >= y          # x e' maggiore o uguale di y
x <= y          # x e' minore o uguale di y
```

Espressioni Booleane

- Ci sono tre operatori logici: and, or, not.
- La loro semantica e' simile al significato che hanno in italiano.

Ad esempio:

- n%2 or n%3 e' vero se vale una delle due condizioni
- x==7 and z==6 e' vero se valgono entrambe.
- not(x==7 and z==6) nega la condizione precedente.

PYTHON

Funzioni

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* uno o piu' argomenti e ***restituisce*** (= ***returns***) un risultato.
- Il risultato viene detto ***valore restituito*** (= ***return value***)

Esempi funzioni

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

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

Esempi funzioni

- 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'
```

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.
- 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*).
- le funzioni (con, tan,...etc...) lavorano con radianti.

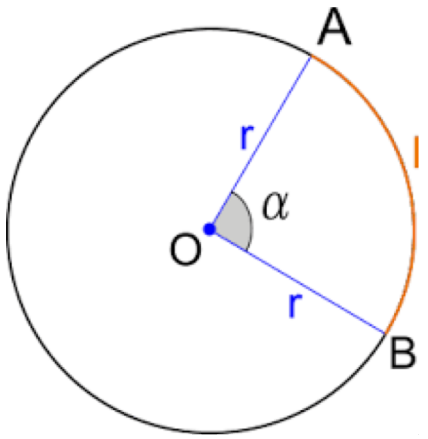
```
>>> decibel = math.log10 (17.0)
>>> angle = 1.5
>>> height = math.sin(angle)

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

Da gradi
a radianti

Gradi e radianti

- Ad ogni angolo al centro a corrisponde un arco di circonferenza l.
- 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 pi/2; quindi la misura in radianti di un angolo di 90° è pi/2.
- In generale possiamo scrivere la proporzione a : l = 360 : 2pi



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)
```

- 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.

Definizione di funzione

- Le definizioni di funzione iniziano sempre con il def.

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

Il nome della funzione è hello

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

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

Il codice o *corpo* (= *body*) e' la lista dei comandi che vengono eseguiti quando una funzione viene chiamata, va bene un qualsiasi comando python.

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?
>>>
```

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?
>>>
```

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

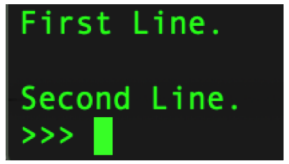
Gli argomenti vengono sempre passati tra parentesi, si tratta di una funzione senza argomenti (la lista e' vuota).

Esempi funzioni

```
def newLine():
    print

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

Cosa succede se l'eseguo?

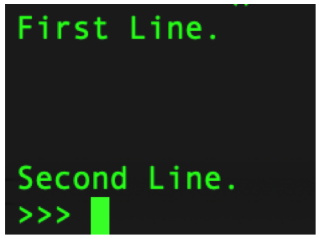


Esempi funzioni

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

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

Cosa succede se l'eseguo?



Osservazioni

- La definizione di funzione termina appena si trova un comando non indentato.
- E' possibile chiamare piu' volte la stessa funzione.
- Una funzione se necessario puo' 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 e' 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.

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 puo' essere chiamata.
- I comandi che la compongono sono effettivamente seguiti solo al momento della chiamata.
- e' 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
...
```

- La chiamata di funzione ha bisogno di un argomento. Si usa la stringa “Mauro”.

```
>>> ciao("Mauro")
```

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

```
Ciao Mauro
```

Esempio di funzione con un parametro

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

```
>>> printTwice('Spam’)
Spam Spam
```

```
>>> printTwice(5)
55
```

```
>>> printTwice(3.14159)
3.14159 3.14159
```