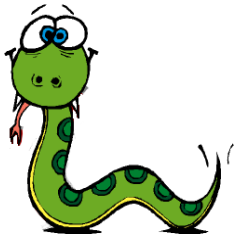
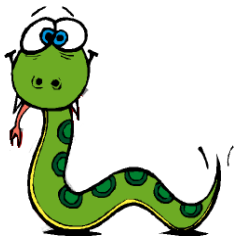
Tecnologie dell'Informazione e della Comunicazione



Capitolo 4
Condizionale e ricorsione

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



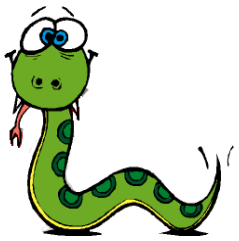
Alcuni operatori utili

- 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
```

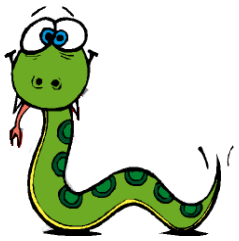
- Si tratta di un operatore molto utile ad esempio per verificare se un numero e' divisibile per un altro: se `x%y` da risultato 0 allora significa che x e' divisibile per y.

Espressioni Booleane



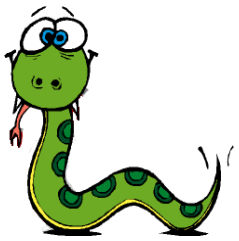
- Un ***espressione booleana*** e' un espressione che puo' essere vera o falsa.
- In python c'e' la seguente convenzione (comune a molti linguaggi di programmazione):
  - **FALSO** ha valore 0
  - **VERO** ha valore 1 oppure **qualsiasi oggetto diverso da 0,** NB. anche un float o una stringa
- Ad esempio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

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



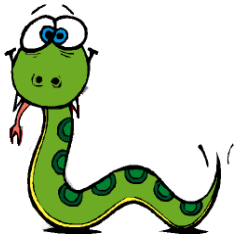
Altri operatori di confronto

```
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
```



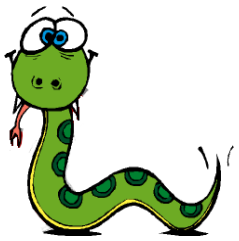
Operatori logici

- 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.



Interprete

- Esempi operatori
- Variabili True e False.



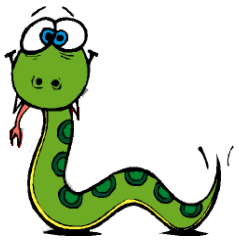
Il condizionale

- Per scrivere programmi utili e' spesso necessario valutare espressioni booleane e cambiare comportamento a seconda del risultato.
- Questo si fa con i ***comandi condizionali***

```
if x > 0:
  print "x is positive"
```

- L'espressione booleana dopo il comando **if** si chiama ***condizione***. Se la condizione e' vera viene eseguito il comando indentato che stampa, altrimenti non accade nulla.

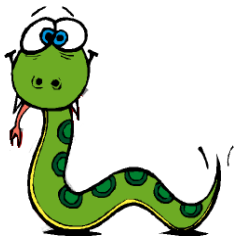
Il concetto di Blocco



- NB. la struttura del comando **if** e' simile per alcuni aspetti alla definizione di funzione.
  - prima c'è un intestazione seguita dai "**:**".
  - poi c'è una sequenza di comandi.
- Struttura generale comune ai due:

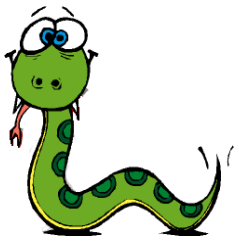
```
HEADER:
  FIRST STATEMENT
  ...
  LAST STATEMENT
```

- HEADER inizia con un newline e finisce con ":".
- I comandi indentati che seguono sono detti ***blocco*** (= ***block***).



Comandi nei blocchi

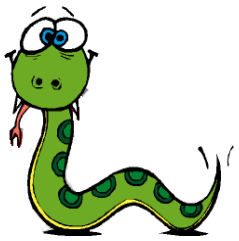
- Non c'è limite al numero di comandi di un blocco.
- Ma ce ne deve essere almeno uno.
- Quindi un blocco vuoto non e' corretto (sintatticamente).
- Possibilità di utilizzare il comando **pass**.



Il comando if then else

- Esiste una forma composta di comando condizionale denominato comunemente ***if_then_else***.

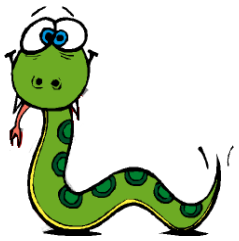
```
if x%2 == 0:
  print x, "is even"
else:
  print x, "is odd"
```



Esempio

```
def printParity(x):
  if x%2 == 0:
    print x, "is even"
  else:
    print x, "is odd"
```

Condizionali concatenati



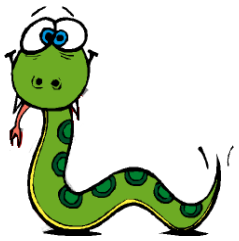
- Quando ci sono piu' continuazioni possibili che dipendono da diverse condizioni si puo' utilizzare un comando ***condizionale concatenato*** a volte detto ***case***.

```
if x < y:
  print x, "is less than", y
elif x > y:
  print x, "is greater than", y
else:
  print x, "and", y, "are equal"
```

- Il comando **elif** e' detto anche “else if”.
- Non c'e' limite al numero delle alternative ma l'ultimo comando deve essere un **else (ma e' opzionale)**.

Semantica condizionali concatenati

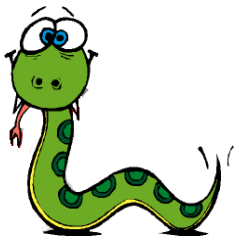
- Le condizioni vengono controllate in ordine.
- Se una di esse e' vera si esegue il blocco corrispondente (anche detto **branch**).
- NB. Viene considerata solo la prima condizione che risulta vera, le condizioni che seguono anche se vere vengono ignorate.



Condizionali innestati

- I condizionali possono essere anche innestati l'uno dentro l'altro.

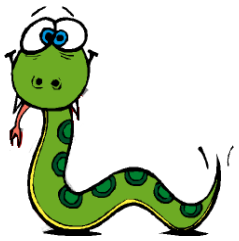
```
if x == y:
  print x, "and", y, "are equal"
else:
  if x < y:
    print x, "is less than", y
  else:
    print x, "is greater than", y
```



Esempio

- A volte gli operatori logici possono servire a semplificare la struttura dei condizionali innestati.

```
if 0 < x:
  if x < 10:
    print "x is a positive single digit."

if 0 < x and x < 10:
  print "x is a positive single digit."

if 0 < x < 10:
  print "x is a positive single digit."
```

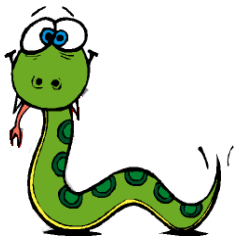


return per uscire da una funzione

- Il comando return permette anche di terminare l'esecuzione di una funzione prima di arrivare alla fine.

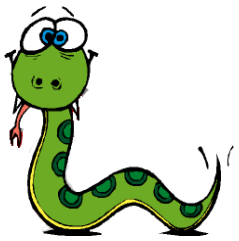
```
import math

def printLogarithm(x):
  if x <= 0:
    print "Positive numbers only, please."
    return

  result = math.log(x)
  print "The log of x is", result
```

Ricorsione



- Abbiamo visto che una funzione puo' chiamare un altra funzione.
- Una funzione puo' anche chiamare se stessa!
- In questo caso la funzione si dice ricorsiva.

```
def countdown(n):
  if n == 0:
    print "Blastoff!"
  else:
    print n
    countdown(n-1)
```

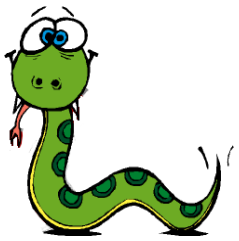


Traccia funzione ricorsiva

```
def countdown(n):
  if n == 0:
    print "Blastoff!"
  else:
    print n
    countdown(n-1)
```

```
>>>countdown(3)
3
2
1
Blastoff!
```

- countdown(3) 3
  - countdown(2) 2
    - countdown(1) 1
      - countdown(0) Blastoff!
      - return
    - return
  - return
- return



Cosa fa' questo programma?

```
def nLines(n):
  if n > 0:
    print
    nLines(n-1)
```

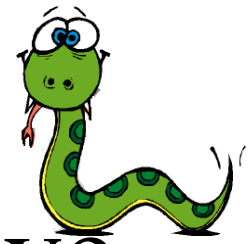
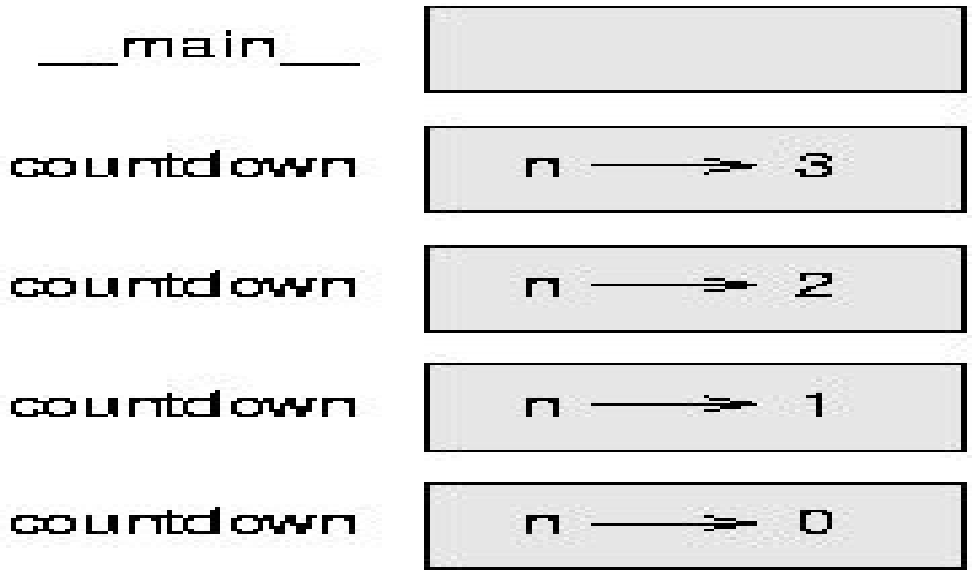
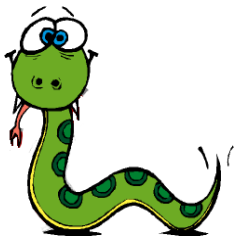


Diagramma a stack: funzioni ricorsive

- Ogni volta che una funzione viene chiamata python crea un nuovo frame che contiene le variabili locali della funzione e i parametri.
- Per le funzioni ricorsive sono attivi sullo stack piu' frame della stessa funzione.

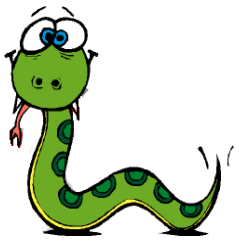




Attenzione alla ricorsione infinita!

- Se una funzione ricorsiva non arriva mai al caso base il programma non termina.

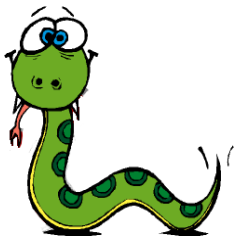
```
def recurse():
  recurse()

>>>recurse()
  File "<stdin>", line 2, in recurse
  (98 repetitions omitted)
  File "<stdin>", line 2, in recurse
RuntimeError: Maximum recursion depth exceeded
```



Definizioni ricorsive

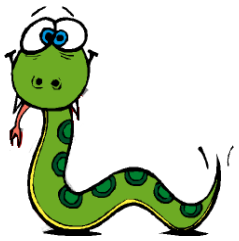
- Studiare attentamente il caso base (terminazione).
- Verificare che la chiamata ricorsiva si avvicina al caso base.
- Verificare i casi base.
- Se tutto questo funziona in genere la funzione ricorsiva funziona bene.

Input da tastiera



- I programmi visti fino ad ora non sono in grado di prendere input dall'utente.
- Python (come tutti i linguaggi) fornisce delle funzioni built-in che permettono di prendere input da una tastiera.
  - **raw_input**: quando viene invocata questa funzione il programma si ferma e aspetta che l'utente digiti qualcosa sulla tastiera. raw_input restituisce quello che l'utente ha scritto convertito in stringa.

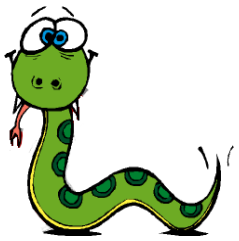
```
>>> input = raw_input ()
What are you waiting for?
>>> print input
What are you waiting for?
```



Input con prompt

- Spesso quando si desidera qualcosa in input e' opportuno stampare anche qualcosa che ci indica questo fatto.
- Si puo' stampare una stringa con una domanda oppure con dei caratteri che indicano che il programma sta aspettando (come fa l'interprete python). Questa stringa si chiama **prompt**.

```
>>> name = raw_input ("What...is your name? ")
What...is your name? Arthur, King of the Britons!
>>> print name
Arthur, King of the Britons!
```



Input di interi

- Se l'input che ci si aspetta e' un intero e' opportuno utilizzare un altra funzione: la **input**.

```
prompt = "Quale e' la velocita' di una 500\n"
speed = input(prompt)
```