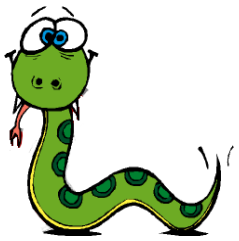
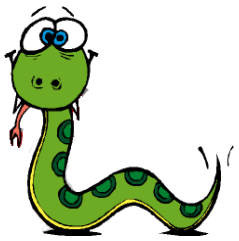
Tecnologie dell'Informazione e della Comunicazione



Capitolo 5
Sviluppo di Funzioni

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



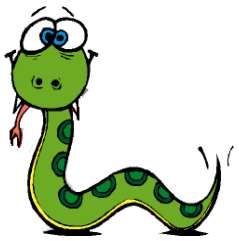
Esempio di funzione area

```
import math

def area(radius):
  temp = math.pi * radius**2
  return temp

# OPPURE in alternativa

def area(radius):
  return math.pi * radius**2
```



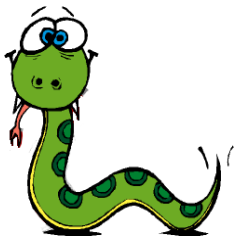
Valore assoluto

Quale dei due funziona?

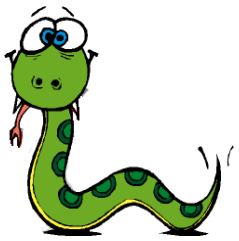
```
def absoluteValue(x):
  if x < 0:
    return -x
  else:
    return x
```

```
def absoluteValue(x):
  if x < 0:
    return -x
  elif x > 0:
    return x
```

Sviluppo incrementale



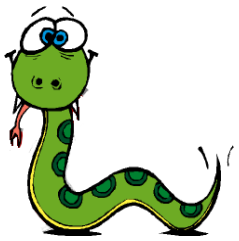
- Per ora abbiamo visto funzioni molto semplici che in genere danno pochi problemi.
- Con problemi piu' difficili aumenta la complessita' delle funzioni e i problemi di programmazione crescono
- E' necessario utilizzare tecniche opportune per la realizzazione di funzioni e programmi.
- **Sviluppo incrementale (= incremental development):**
  per evitare lunghe sessioni di debugging si costruisce un programma aggiungendo e testando di volta in volta piccole porzioni di codice.



Esempio distanza tra due punti

Teorema di Pitagora
- (x1,y1)
- (x2,y2)

$$\text{distanza} = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$



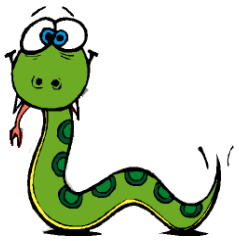
Sviluppo incrementale - esempio

- Si tratta di una funzione con quattro parametri, restituisce un valore floating point che e' la distanza.

```
def distance(x1, y1, x2, y2):
  return 0.0
```

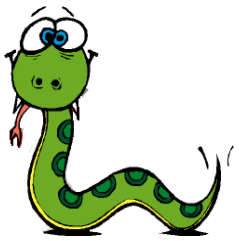
TEST?

```
>>> distance(1, 2, 4, 6)
0.0
```



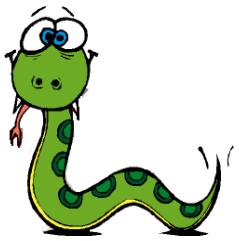
Sviluppo incrementale: la somma

```
def distance(x1, y1, x2, y2):
  dx = x2 - x1
  dy = y2 - y1
  print "dx is", dx
  print "dy is", dy
  return 0.0
```



Sviluppo incrementale: i quadrati

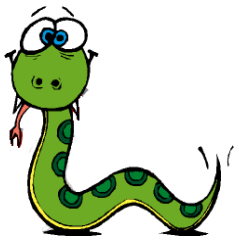
```
def distance(x1, y1, x2, y2):
  dx = x2 - x1
  dy = y2 - y1
  dsquared = dx**2 + dy**2
  print "dsquared is: ", dsquared
  return 0.0
```



Sviluppo incrementale: i quadrati

```
def distance(x1, y1, x2, y2):
  dx = x2 - x1
  dy = y2 - y1
  dsquared = dx**2 + dy**2
  result = math.sqrt(dsquared)
  return result
```

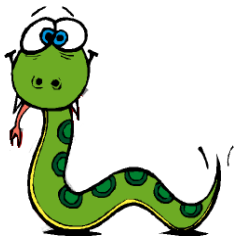
Aspetti principali dello sviluppo incrementale:



1) Partire con un programma che funziona e fare dei piccoli cambiamenti incrementali.

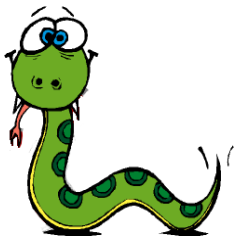
2) Utilizzare variabili locali per memorizzare i valori intermedi e se necessario stamparle.

3) Alla fine si puo' compattare il programma per levare alcune delle cose in piu', ma solo se questo non ne compromette la leggibilita'.



Composizione di funzioni

- Si ottiniene quando si chiama una funzione da un altra funzione.
- Esempio: scrivere una funzione che dati due punti, ne considera uno come centro di una circonferenza e il secondo come punto del perimetro e ne calcola l'area.



Esempio area2

Dati due punti trovare l'area della circonferenza che si ottiene con un punto al centro e l'altro punto nel perimetro.

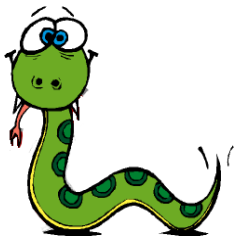
```
radius = distance(xc, yc, xp, yp)

result = area(radius)
return result
```

```
def area2(xc, yc, xp, yp):
  radius = distance(xc, yc, xp, yp)
  result = area(radius)
  return result
```

```
def area2(xc, yc, xp, yp):
  return area(distance(xc, yc, xp, yp))
```

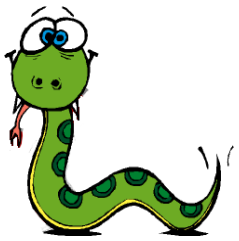
Funzioni booleane



- Sono utili per nascondere test complicati nelle funzioni. Ad esempio all'interno dei condizionali.
- Spesso e' comodo usare per queste funzioni nomi che assomigliano a domande con risposta si/no.

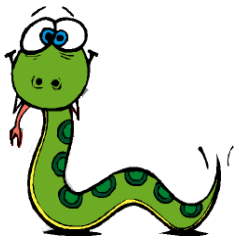
```
def isDivisible(x, y):
  if x % y == 0:
    return 1          # it's true
  else:
    return 0          # it's false

def isDivisible(x, y):
  return x % y == 0
```



Esempio

```
if isDivisible(x, y):
  print "x is divisible by y"
else:
  print "x is not divisible by y"
```



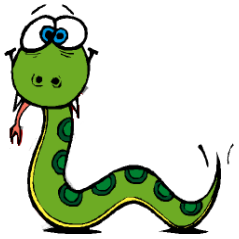
Esempio: Il fattoriale

$0! = 1$

$n! = n\,(n-1)!$

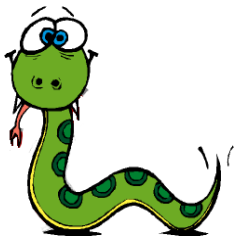
```
def factorial(n):
  if n == 0:
    return 1
```

Il fattoriale (continua)



```
def factorial(n):
  if n == 0:
    return 1
  else:
    recurse = factorial(n-1)
    result = n * recurse
    return result

def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n-1)
```

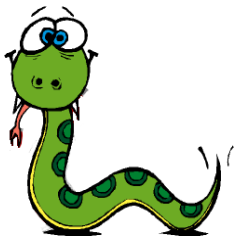


Esempio: Fibonacci

```
fibonacci(0) = 1
fibonacci(1) = 1
fibonacci(n) = fibonacci(n-1) + fibonacci(n-2);
```

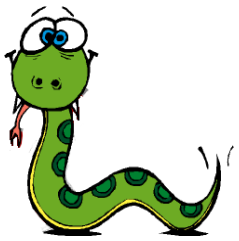
```
def fibonacci (n):
  if n == 0 or n == 1:
    return 1
  else:
    return fibonacci(n-1) + fibonacci(n-2)
```

Controllo dei tipi



- Cosa succede se si chiama il fattoriale con 1.5 come argomento?
- interprete....
- Come risolvere il problema?

```
def factorial (n):
  if type(n) != type(1):
    print "Factorial is only defined for integers."
    return -1
  elif n < 0:
    print "Factorial is only defined for positive integers."
    return -1
  elif n == 0:
    return 1
  else:
    return n * factorial(n-1)
```



Osservazioni

- Ragionando si riesce a dimostrare che ora il programma fattoriale termina sempre.
- Questo si puo' fare grazie alle condizioni che sono state aggiunte dette **guardie** (= **guardians**).
- Le guardie permettono di dimostrare la correttezza del codice e sopratutto permettono di minimizzare gli errori a tempo di esecuzione.