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

Funzioni

Regola valutazione funzioni

- In python le espressioni che vengono passate alle funzioni vengono valutate prima di eseguire la funzione.
- Questa regola si chiama: *valutazione per valore:* i parametri 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

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


>>> subtract(8, 5)
3
>>>
```

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

>>>
```

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
```

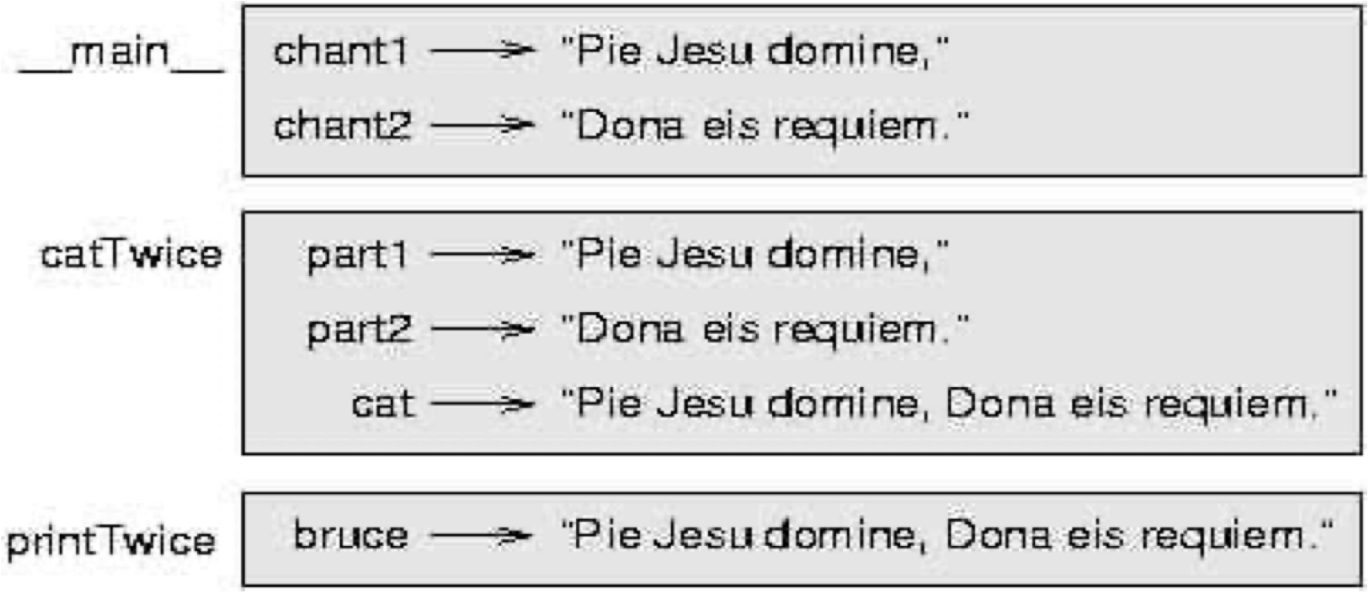
Diagrammi a stack

- 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__.

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

def catTwice(part1, part2):
    cat = part1 + part2
    printTwice(cat)

>>> chant1 = "Pie Jesu domine, "
>>> chant2 = "Dona eis requiem."
>>> catTwice(chant1, chant2)
```



Diagrammi a stack e dubugging

- 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
```

PYTHON

Condizionale

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

- 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.

Semantica condizionali concatenati

- Le condizioni vengono controllate in ordine.
- Se una di esse e' vera si esegue il blocco corrispondente (anche detto branch).
- 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
```

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 builtin 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

- In alternativa si può utilizzare la funzione input.

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

PYTHON

Sviluppo di funzioni

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

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

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

Quale dei due funziona?

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)

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

Esempio distanza tra due punti

- 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
```

Esempio distanza tra due punti

- 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
```

Test?

```
>>> distance(1, 2, 4, 6)
dx is 3
dy is 4
0.0
```

Esempio distanza tra due punti

- 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
```

Test?

```
>>> distance(1, 2, 4, 6)
dsquared is 25
0.0
```

Esempio distanza tra due punti

- 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
```

Test?

```
>>> distance(1, 2, 4, 6)
5
```

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

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.

```
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"
```