Laboratorio 04

Programmazione - CdS Matematica

Michele Donini
19 Novembre 2013

Controllo del flusso

- **Selezione**: ci permette di modificare il normale flusso sequenziale di un programma a seconda della valutazione di una certa condizione sullo stato del programma.
- **Iterazione**: ci permette di esprimere la ripetizione di un insieme di comandi a seconda della valutazione di una certa condizione.

Espressioni a valori booleani

I modi con i quali è possibile creare espressioni da utilizzare per ottenere valori di verità (vero o falso) in Python e quindi utilizzabili per modificare il flusso del programma. In particolare vedremo:

- espressioni basiche,
- identità,
- comparazione,
- appartenenza,
- connettivi logici,
- espressioni condizionali.

Espressioni basiche

Istanze nulle, che vengono valutate a *False*:

- L'oggetto None di tipo NoneType
- Il booleano False
- Gli zeri degli oggetti numerici: 0, 0L, 0.0, 0.0j
- Ogni iterabile vuoto: '', (), [], {}, set([])

Le altre istanze vengono quindi valutate a *True*.

```
>>> x = 1
>>> bool(x-1)
False
>>> bool('')
False
>>> bool(' ')
True
```

Esercizio

Senza eseguire il codice, dire qual è il valore basico di verità di x, y e z dopo aver eseguito le seguenti istruzoni.

```
>>> x = "ciao"
>>> y = x.split('c')
>>> z = y[0]
>>> y = [y[0]]
```

Esercizio

Senza eseguire il codice, dire qual è il valore basico di verità di x, y e z dopo aver eseguito le seguenti istruzoni.

```
>>> x = "ciao"
>>> y = x.split('c')
>>> z = y[0]
>>> y = [y[0]]
```

Soluzione: x e y *True*, z *False*.

```
>>> x
'ciao'
>>> y
['']
>>> z
''
```

Identità

Si utilizza l'operatore **is**.
Dato che il comportamento potrebbe non essere uniforme su diverse piattaforme, è sempre meglio evitare l'utilizzo dell'operatore di identità quando si ha a che fare con oggetti immutabili.

```
>>> x,y = 1,1 # due interi uguali..
>>> x is y # sono in realtà lo stesso oggetto!
True
>>> x,y = 3.14,3.14 # due float uguali..
>>> x is y # invece non sono lo stesso oggetto
False
>>> [] is [] # i tipi mutabili: sempre oggetti diversi
False
```

Esercizio

Sostituire (senza eseguire il codice!) ai punti di domanda il valore booleano corretto *True* o *False*.

```
>>> x = [-1,1]
>>> y = [x,[-1,1]]
>>> x is y[1]
?
>>> x is y[0]
?
>>> [x,[-1,1]] is y
?
>>> [x,y[1]] is y
?
>>> [y[0],y[1]] is y
?
```

Esercizio

Sostituire (senza eseguire il codice!) ai punti di domanda il valore booleano corretto *True* o *False*.

```
>>> x = [-1,1]
>>> y = [x,[-1,1]]
>>> x is y[1]
?
>>> x is y[0]
?
>>> [x,[-1,1]] is y
?
>>> [x,y[1]] is y
?
>>> [y[0],y[1]] is y
?
>>> id(y)  # 35129344
>>> id([y[0],y[1]]) # 35093440 diverso da id(y)
>>> id([y[0],y[1]]) # 35128504 ancora diverso!
```

Risposte: False, True, False, False, False.

Comparazione

In Python abbiamo i seguenti operatori di comparazione:

- $<$ (minore),
- $>$ (maggiore),
- $==$ (uguale),
- $!=$ (diverso),
- $<=$ (minore o uguale),
- $>=$ (maggiore o uguale)

Ricordiamoci che confrontando **tipi diversi**, l'interpretazione dell'**operatore di comparazione cambia**.

Esercizio

Sostituire (senza eseguire il codice!) ai punti di domanda il valore booleano corretto *True* o *False*.

```
>>> "abc" <= "abcd"
?
>>> "abcd" <= "abz"
?
>>> "abc" <= "Abc"
?
>>> x = 2.34
>>> 2.34 == x
?
>>> k = 2.34 * 2
>>> k == 4.68
?
>>> m = 1
>>> m = m/2
>>> m != 0
?
```

Esercizio

Sostituire (senza eseguire il codice!) ai punti di domanda il valore booleano corretto *True* o *False*.

```
>>> "abc" <= "abcd"
?
>>> "abcd" <= "abz"
?
>>> "abc" <= "Abc"
?
>>> x = 2.34
>>> 2.34 == x
?
>>> k = 2.34 * 2
>>> k == 4.68
?
>>> m = 1
>>> m = m/2
>>> m != 0
?
```

Risposte: True, True, False, True, True, False.

Appartenenza

La verifica di appartenenza di un oggetto ad una collezione (stringa, lista, tupla, dizionario, insieme) tramite il comando **in**.

```
>>> paperopoli = ['pippo','pluto','paperino','paperone']
>>> 'nonnapapera' in paperopoli
False
>>> 'pluto' in paperopoli
True
```

Connettivi Logici

- I connettivi logici sono **and**, **or** e l'operatore unario di negazione **not**.
- Similmente ad altri linguaggi di programmazione, viene eseguita con la tecnica cosiddetta **short-circuit** (cortocircuito).
- Le due espressioni coinvolte vengono effettivamente valutate solo se tale valutazione risulti strettamente necessaria.
- I connettivi **and** e **or** in Python non ritornano necessariamente un valore booleano ma quello che è importante è che il valore ritornato viene valutato correttamente come un booleano quando richiesto.

Esempio

Vediamo qualche veloce esempio:

```
>>> 2 and 3
2
>>> 2 and 0
0
>>> "buono" or "cattivo"
'buono'
>>> not ("cattivo" or "buono")
False
>>> 'uno' or 1
'uno'
```

Espressioni condizionali e esercizio

Nella forma: EXPT **if** COND **else** EXPF.
Completare con la corretta espressione condizionale, in modo da creare la lista dei valori da 1 a 40 con azzerati i multipli di 4 e di 6 ma non i multipli di entrambi.

```
>>> a = range(1,41)
>>> [0 ... for p in a]
```

Espressioni condizionali e esercizio

Nella forma: EXPT **if** COND **else** EXPF.
Completare con la corretta espressione condizionale, in modo da creare la lista dei valori da 1 a 40 con azzerati i multipli di 4 e di 6 ma non i multipli di entrambi.

```
>>> a = range(1,41)
>>> [0 ... for p in a]
>>> [0 if (p%4==0 or p%6==0) and not(p%4==0 and p%6==0)
    else p for p in a]
[1, 2, 3, 0, 5, 0, 7, 0, 9, 10, 11, 12, 13, 14, 15, 0,
    17, 0, 19, 0, 21, 22, 23, 24, 25, 26, 27, 0,
29, 0, 31, 0, 33, 34, 35, 36, 37, 38, 39, 0]
```

Script

- Aprire idle dal terminale:

```
idle &
```

- Digitare qualche comando all'interno dell'editor.
- Salvare il file con il nome *primo.py*.
- Da terminale utilizzare il comando:

```
python primo.py
```

per eseguire i comandi inseriti nello script.

IF ELIF ELSE

Vediamo un semplice esempio dell'operatore di selezione:

```
importo = int(raw_input('Inserire importo: '))
if importo >= 100:
    sconto = 10
elif importo >= 50:
    sconto = 15
else:
    sconto = 20
print('Sconto= %d'%(sconto))
```

Notare come i costrutti if, else, elif, richiedano i due punti (:) per indicare che dalla riga seguente inizia il blocco di istruzioni relativo. Ogni blocco più interno in Python deve essere indentato (ovvero, orizzontalmente spostato verso destra) rispetto al relativo costrutto.

Esercizi semplici

- Creare lo script inputAnumber.py che inizializza una variabile n al valore 4, chiede all'utente di inserire un numero maggiore o uguale a n tramite tastiera, stampa un messaggio di rimprovero nel caso che il numero sia minore di n oppure di felicitazioni in caso contrario.
- Creare lo script simulaAnd.py che (senza utilizzare l'operatore and) inizializza due variabili, a e b, all'interno dello script ad un valore a scelta tra False,True. Stampa True se sia a che b valgono True, False altrimenti.
- Creare lo script simulaOr.py che sia l'equivalente di simulaAnd.py per l'operatore or.
- Creare lo script simulaXor.py (VERO se, e solo se, uno solo degli operandi è VERO).

Soluzioni

- Creare lo script inputAnumber.py che inizializza una variabile n al valore 4, chiede all'utente di inserire un numero maggiore o uguale a n tramite tastiera, stampa un messaggio di rimprovero nel caso che il numero sia minore di n oppure di felicitazioni in caso contrario.

Soluzioni

- Creare lo script inputAnumber.py che inizializza una variabile n al valore 4, chiede all'utente di inserire un numero maggiore o uguale a n tramite tastiera, stampa un messaggio di rimprovero nel caso che il numero sia minore di n oppure di felicitazioni in caso contrario.

```
n = 4
m = float(raw_input("Inserisci un valore maggiore o
    uguale di %d:"%n))
if (m>=n):
    print("Complimenti!")
else:
    print("Cattivo!")
```

Soluzioni

- Creare lo script simulaAnd.py che (senza utilizzare gli operatori and, or, not) inizializza due variabili, a e b, all'interno dello script ad un valore a scelta tra False,True. Stampa True se sia a che b valgono True, False altrimenti.

Soluzioni

- Creare lo script simulaAnd.py che (senza utilizzare gli operatori and, or, not) inizializza due variabili, a e b, all'interno dello script ad un valore a scelta tra False,True. Stampa True se sia a che b valgono True, False altrimenti.

```
a = False
b = True
if (a == False):
    print("False")
elif (b == False):
    print("False")
else:
    print("True")
```

Soluzioni

- Creare lo script simulaOr.py che sia l'equivalente di simulaAnd.py per l'operatore or.

Soluzioni

- Creare lo script simulaOr.py che sia l'equivalente di simulaAnd.py per l'operatore or.

```
a = False
b = True
if (a == True):
    print("True")
elif (b == True):
    print("True")
else:
    print("False")
```

Soluzioni

- Creare lo script simulaXor.py (VERO se, e solo se, uno solo degli operandi è VERO).

Soluzioni

- Creare lo script simulaXor.py (VERO se, e solo se, uno solo degli operandi è VERO).

```
a = False
b = True
if (a == True):
    if (b == False):
        print("True")
    else:
        print("False")
elif (b == True):
    print("True")
else:
    print("False")
```

WHILE

Il tipico schema di programmazione iterativa condizionale è:

- Inizializza variabili V presenti in COND.
- Fino a che COND rimane vera (while COND:)
  - fai qualcosa (eventualmente usando V),
  - modifica variabili V.

Esempio:

```
num = int(raw_input("Inserire un numero intero: "))
while num!=5:
    print "Valore: %d" % num
    num = int(raw_input("Inserire un numero intero: "))
```

Esercizio

Creare uno script sottomedia.py che:

- Generi un valore casuale M intero compreso tra 10 e 100.
- Comunichi all'utente il valore di M.
- Chieda all'utente dei valori numerici fintanto che la MEDIA dei valori inseriti non sia maggiore o uguale a M.
- Quando la media dei valori è maggiore o uguale a M lo script conclude stampando la media ottenuta.

Suggerimento: usare randrange da random.

Soluzione

Ecco la soluzione:

Soluzione

Ecco la soluzione:

```
from random import randrange
M = randrange(10,101)
print("Valore di M %d:"%M)
valore = float(raw_input("Inserire un valore:"))
contatore = 1
totale = valore

while(totale/contatore < M):
    valore = float(raw_input("Inserire un valore:"))
    totale = totale + valore
    contatore = contatore + 1
print("Media: %.5f"%(totale/contatore))
```

Iteratori

- Un altro meccanismo di iterazione molto usato in Python è costruito mediante l'iteratore for in un modo molto simile a quello utilizzato per i descrittori di lista.
- L'iteratore **for** attraversa uno ad uno tutti gli elementi di un iterabile.

La sua struttura è la seguente:

```
for X in IT:
    BLOCCO
```

Esempi

Scrivere 4 volte la stringa Ciao!:

```
for v in range(4):
    print("Ciao!")
```

Stampare i caratteri della stringa Giorni:

```
print("caratteri:")
for c in "Giorni":
    print "%c"%c
```

Stampare una rubrica:

```
rubrica = {'mara':'340-1123451','giulio':'329-7678854'}
for nome in rubrica: # come for nome in rubrica.keys()
    print("%s -> tel. %s"%(nome,rubrica[nome]))
```

Esercizi semplici

- Creare uno script factorial.py che chiede all'utente di inserire un numero naturale, controlla che il numero inserito sia maggiore di 0 e stampa il fattoriale del numero, es. 4!=24.
- Creare uno script verifyRand.py che simula n volte il lancio di un dado e stampa le frequenze con cui ciascuna faccia del dado. Veriricare all'aumentare del numero di lanci la convergenza alla probabilità teorica di un sesto. Dopo 10000 tiri, ecco il numero di volte che ogni faccia del dado e' uscita: [1691, 1620, 1703, 1664,1726, 1596]. **Suggerimento**: usare randrange da random.

Soluzioni

- Creare uno script factorial.py che chiede all'utente di inserire un numero naturale, controlla che il numero inserito sia maggiore di 0 e stampa il fattoriale del numero.

Soluzioni

- Creare uno script factorial.py che chiede all'utente di inserire un numero naturale, controlla che il numero inserito sia maggiore di 0 e stampa il fattoriale del numero.

```
n = int(raw_input("Inserire un intero:"))
fattoriale = 1

if (n>0):
    for i in range(n):
        fattoriale = fattoriale * (i+1)
    print("%d!=%d"%(n,fattoriale))
else:
    print("n<=0")
```

Soluzioni

- Creare uno script verifyRand.py che simula n volte il lancio di un dado e stampa le frequenze con cui ciascuna faccia del dado. Veriricare all'aumentare del numero di lanci la convergenza alla probabilità teorica di un sesto.
  Suggerimento: usare randrange da random.

Soluzioni

- Creare uno script verifyRand.py che simula n volte il lancio di un dado e stampa le frequenze con cui ciascuna faccia del dado. Veriricare all'aumentare del numero di lanci la convergenza alla probabilità teorica di un sesto.
  Suggerimento: usare randrange da random.

```
from random import randrange
N = 10000 #numero di lanci

lanci = [0]*6
for i in range(N):
    lanci[randrange(6)] += 1
print(lanci)
```