

# Laboratorio 04

Programmazione - CdS Matematica

Mirko Polato

18 Novembre 2014



- **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

Esistono diversi 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
```

# 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 = 17, 17 # 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 # possono non essere lo stesso oggetto
False
>>> {} is {} # i tipi mutabili: sempre oggetti diversi
False
```

# Comparazione e appartenenza

- In Python abbiamo i seguenti operatori di comparazione:

<, >, ==, !=, <=, >=

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

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

```
>>> gotham = ['batman', 'joker', 'bane']
>>> 'batman' in gotham
True
>>> 'iron man' in gotham
False
```

# 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
3
>>> 2 and 0
0
>>> "buono" or "cattivo"
"buono"
>>> [] or "bazinga" or 42
"bazinga"
>>> (0 or {}) and ("due" or 3)
{}
>>> (0 or "uno") and (2 or "tre")
2
```

# Esercizio

Dati  $x = [0, 17]$ ,  $y = (x, [0, 17])$  determinare, **prima di eseguire**, il valore ritornato dalle seguenti espressioni.

```
>>> x[1] == y[1][0]
>>> x[1] > y[1][0]
>>> x[1] <= y[1][1]
>>> x is y[0]
>>> x is y[1]
>>> x[0] and y[0][0] or y[1][1]
>>> x[0] and (y[0][0] or y[1][1])
```

# Esercizio

Dati  $x = [0, 17]$ ,  $y = (x, [0, 17])$  determinare, **prima di eseguire**, il valore ritornato dalle seguenti espressioni.

```
>>> x[1] == y[1][0]
>>> x[1] > y[1][0]
>>> x[1] <= y[1][1]
>>> x is y[0]
>>> x is y[1]
>>> x[0] and y[0][0] or y[1][1]
>>> x[0] and (y[0][0] or y[1][1])
```

Risposte: False, True, True, True, False, 17, 0.

# Espressioni condizionali e esercizio

EXPT **if** COND **else** EXPF.

Prima viene valutata COND, se questa risulta vera allora esegue EXPT, altrimenti esegue 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)
```

# Espressioni condizionali e esercizio

EXPT if COND else EXPF.

Prima viene valutata COND, se questa risulta vera allora esegue EXPT, altrimenti esegue 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 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]
```

- Aprire idle dal terminale:

```
idle &
```

- Aprire l'editor dal menu File → New window
- 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.  
Oppure premere F5 all'interno dell'editor.

# Operatore di selezione

```
if COND:
    BLOCCO
elif COND_1:
    BLOCCO_1
#...
else:
    BLOCCO_N
```

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))
```

# Esercizi semplici

- Creare lo script `simulaAnd.py` che (senza utilizzare l'operatore `and`) inizializza due variabili, `a` e `b`, ad un valore a scelta tra `False` e `True`. Stampa `True` se sia `a` che `b` valgono `True`, `False` altrimenti.
- Creare lo script `simulaXor.py` (VERO se, e solo se, uno solo degli operandi è VERO).
- Creare lo script `morra.py` che simula un round di morra cinese: chiede all'utente di inserire la sua scelta tra: "sasso", "carta" o "forbice". Poi genera casualmente (importare modulo `random`) la scelta del computer e infine restituisce all'utente una stringa per notificare il risultato. (Stampare "input errato" se la stringa inserita dall'utente non è valida)

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

- Creare lo script `simulaAnd.py` che (senza utilizzare gli operatori `and`, `or`, `not`) inizializza due variabili, `a` e `b`, ad un valore a scelta tra `False` e `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")
```

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

- 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")
```

# Soluzione - Morra cinese

```
import random
valide = ["sasso", "carta", "forbice"]
utente = raw_input("inserisci la tua scelta: ")
if utente not in valide:
    print "input errato"
else:
    cpu = valide[random.randint(0,2)]
    print "scelta cpu: %s" %cpu
    if cpu == utente:
        print "pareggio"
    elif cpu == "sasso":
        print "vinto" if utente == "carta" else "perso"
    elif cpu == "forbice":
        print "vinto" if utente == "sasso" else "perso"
    else:
        print "vinto" if utente == "forbice" else "perso"
```

# Cicli - WHILE

```
while COND:  
    BLOCCO #modifica variabili in COND
```

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:

```
from math import sqrt  
num = int(raw_input("Inserire un numero intero: "))  
while not sqrt(num).is_integer():  
    print "Valore: %d" % num  
    num = int(raw_input("Inserire un numero intero: "))
```

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:

```
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))
```

# Esercizio

Generare uno script `yahtzee.py` che simula il lancio di tre dadi finché o si ottiene uno yahtzee, ovvero tutti e 3 i lanci restituiscono la faccia 6, o si arriva a 100 lanci. Restituire il numero di tentativi effettuati per ottenere lo yahtzee oppure, nel caso non si sia ottenuto, un messaggio “triste” 😞. **Suggerimento:** usare `randint` da `random`, `randint(1, 6)`.

# Soluzione

Ecco la soluzione:

```
from random import randint
d1, d2, d3 = 0, 0, 0
count = 0
while (d1+d2+d3 < 18 and count <= 100):
    d1 = randint(1,6)
    d2 = randint(1,6)
    d3 = randint(1,6)
    count += 1
    print "%d, %d, %d" %(d1, d2, d3)
if count <= 100:
    print "ci sono voluti %d tentativi" %count
else:
    print "no yahtzee :("
```

# Iteratori - FOR

- 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

Iterare su range di valori. Esempio, stampare tabellina del 2:

```
for v in range(2,21,2):  
    print (v)
```

# Esempi

Iterare su range di valori. Esempio, stampare tabellina del 2:

```
for v in range(2,21,2):  
    print(v)
```

Iterare per riferimenti. Esempio, 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]))
```

# Esempi

Iterare su range di valori. Esempio, stampare tabellina del 2:

```
for v in range(2,21,2):  
    print (v)
```

Iterare per riferimenti. Esempio, 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]))
```

Iterare per indice. Esempio, scorrere una lista:

```
movies = ["Memento", "Inception", "Interstellar"]  
for i in range(len(movies)):  
    print ("film in indice #%d e' %s" %(i, movies[i]))
```

# Esempi

Iterare su range di valori. Esempio, stampare tabellina del 2:

```
for v in range(2,21,2):  
    print(v)
```

Iterare per riferimenti. Esempio, 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]))
```

Iterare per indice. Esempio, scorrere una lista:

```
movies = ["Memento", "Inception", "Interstellar"]  
for i in range(len(movies)):  
    print("film in indice #d e' s" %(i, movies[i]))  
  
for i, movie in enumerate(movies):  
    print("film in indice #d e' s" %(i, movie))
```

# 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 di ciascuna faccia del dado. Verificare all'aumentare del numero di lanci la convergenza alla probabilità teorica di un sesto.  
**Suggerimento:** usare `randrange` da `random`.
- Creare uno script `divisors.py` che chiede all'utente di inserire un numero naturale, controlla che il numero inserito sia maggiore di 0 e stampa la lista di **tutti** i suoi divisori, senza l'utilizzo del descrittore di lista.

- 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.

- 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")
```

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

- Creare uno script `verifyRand.py` che simula  $n$  volte il lancio di un dado e stampa le frequenze con cui ciascuna faccia del dado. Verificare 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)
```

- Creare uno script `divisors.py` che chiede all'utente di inserire un numero naturale, controlla che il numero inserito sia maggiore di 0 e stampa la lista di **tutti** i suoi divisori in ordine crescente, senza l'utilizzo del descrittore di lista.

- Creare uno script `divisors.py` che chiede all'utente di inserire un numero naturale, controlla che il numero inserito sia maggiore di 0 e stampa la lista di **tutti** i suoi divisori in ordine crescente, senza l'utilizzo del descrittore di lista.

```
n = int(raw_input("inserire numero: "))
if (n > 0):
    l = []
    for i in range(1, n/2+1):
        if n%i == 0:
            l += [i]
    l += n
    print l
else:
    print "n <= 0"
```

## Esercizio - While o For?

- Creare uno script fibo.py che richiede all'utente di inserire 2 interi  $< 10$  e genera la sequenza dei primi 1000 numeri di "Fibonacci" (ogni elemento è la somma dei due precedenti) usando come valori iniziali quelli inseriti dall'utente, es. 1, 3 inseriti dall'utente daranno 1, 3, 4, 7, 11, ....

Confrontare infine il valore restituito dalla divisione degli ultimi due valori generati con il numero aureo  $\frac{1+\sqrt{5}}{2}$ . Cosa notate?

**Suggerimento:** usare sqrt da modulo math.

# Soluzione - "Fibonacci" like

```
from math import sqrt
n1 = int(raw_input("Inserire primo numero: "))
n2 = int(raw_input("Inserire secondo numero: "))
print "%d\n%d" %(n1, n2)
for i in range(1000):
    if i%2:
        n1 += n2
        print n1
    else:
        n2 += n1
        print n2
aurea = (1 + sqrt(5)) / 2
lasts = float(n1) / n2
print aurea
print lasts
print aurea - lasts
```

# Esercizio - While o For?

- Creare uno script `morraComplete.py` che simula un match completo a morra cinese (potete riusare parte dello script `morra.py`), decretando un vincitore solo dopo che uno tra utente e computer raggiunge i 5 punti (1 punto per ogni vittoria, 0 per pareggio o sconfitta). Ignorare i turni con input errati.

# Soluzioni - Morra cinese completa

```
from random import randint
valide = ["sasso", "carta", "forbice"]
cu, cp = 0, 0
while (cu < 5 and cp < 5):
    utente = raw_input("Inserisci la tua scelta: ")
    if utente in valide:
        cpu = valide[randint(0,2)]
        print "scelta cpu: %s" %cpu
        if cpu != utente:
            if cpu == "sasso":
                (cu, cp) = (cu+1, cp) if u == "carta"
                else (cu, cp+1)
            elif cpu == "forbice":
                (cu, cp) = (cu+1, cp) if utente=="sasso"
                else (cu, cp+1)
            else:
                (cu, cp) = (cu+1, cp) if utente=="forbice"
                else (cu, cp+1)
print "hai vinto" if cu > cp else "hai perso"
```