

# Esecuzione condizionale ed espressioni logiche

Informatica@SEFA 2018/2019 - Lezione 6

**Massimo Lauria** <massimo.lauria@uniroma1.it>  
<http://massimolauria.net/courses/infosefa2018/>

**Venerdì, 5 Ottobre 2018**

# Ripartiamo dal nostro esempio

$$Ax^2 + Bx + C = 0$$

```
import math 1
2
def eqsecondogrado(A,B,C): 3
    """Risolve equazioni di 2o grado A x^2 + B x + C = 0""" 4
    Delta = B*B - 4*A*C 5
    if Delta < 0: 6
        print("Nessuna soluzione") 7
    else: 8
        if A==0: 9
            print("Non è un equazione propria di 2o grado") 10
        else: 11
            # Utilizzo la formula standard 12
            sol1 = ( -B - math.sqrt(Delta) ) / 2*A 13
            sol2 = ( -B + math.sqrt(Delta) ) / 2*A 14
            print("Soluzioni: ",sol1,sol2) 15
```

# Esecuzione

eqsecondogrado(1, 0, 0)	1
eqsecondogrado(0, 3, 1)	2
eqsecondogrado(1, 2, 1.0)	3
eqsecondogrado(2, 1, 2)	4
eqsecondogrado(2, 1, -3)	5
eqsecondogrado(2, 1, -2)	6
print('')	7
help(eqsecondogrado)	8

```
Soluzioni: 0.0 0.0
Non è un equazione propria di 2o grado
Soluzioni: -1.0 -1.0
Nessuna soluzione
Soluzioni: -6.0 4.0
Soluzioni: -5.123105625617661 3.1231056256176606
```

Help on function eqsecondogrado in module \_\_main\_\_:

```
eqsecondogrado(A, B, C)
    Risolve equazioni di 2o grado  $A x^2 + B x + C = 0$ 
```

# Estendiamo la docstring

- ▶ i parametri della funzione
- ▶ il valore restituito
- ▶ altri effetti e comportamenti

```
import math 1
2
def eqsecondograde(A,B,C): 3
    """Risolve equazioni di 2o grado  $A x^2 + B x + C = 0$  4
    5
    Stampa le due soluzioni dell'equazione, se esistono. Se 6
    la soluzione è unica viene stampata due volte. 7
    Non accetta equazioni di grado inferiore a 2. 8
    9
    Parametri: 10
    - A : coefficiente del termine di secondo grado 11
    - B : coefficiente del termine di primo grado 12
    - C : coefficiente del termine costante 13
    14
    Restituisce: nulla. 15
    """ 16
```

# Vediamo il risultato

```
help(eqsecondogrado)
```

1

Help on function eqsecondogrado in module \_\_main\_\_:

eqsecondogrado(A, B, C)

Risolve equazioni di 2o grado  $A x^2 + B x + C = 0$

Stampa le due soluzioni dell'equazione, se esistono. Se la soluzione è unica viene stampata due volte. Non accetta equazioni di grado inferiore a 2.

Parametri:

- A : coefficiente del termine di secondo grado
- B : coefficiente del termine di primo grado
- C : coefficiente del termine costante

Restituisce: nulla.

# Prendere decisioni

# Scegliere le istruzioni da eseguire

```
pioggia = False           1
nuvoloso = True           2
if pioggia or nuvoloso:   3
    print("1. Prenderò l'ombrello") 4
    print("1. Prenderò le scarpe chiuse") 5
                                6
nuvoloso = False          7
if pioggia or nuvoloso:   8
    print("2. Prenderò l'ombrello") 9
    print("2. Prenderò le scarpe chiuse") 10
```

```
1. Prenderò l'ombrello
1. Prenderò le scarpe chiuse
```

# Sintassi del costrutto `if`

<code>if</code> condizione:	1
istruzione1	2
istruzione2	3
istruzione3	4
...	5

- ▶ **condizione** espressione dal valore booleano (**vero/falso**)
- ▶ **istruzione1** indentata rispetto alla riga precedente
- ▶ **le altre istruzioni** allineate con **istruzione1**



# Due alternative (If-then-else)

```
pioggia = False           1
nuvoloso = False          2
if pioggia or nuvoloso:    3
    print("Prenderò l'ombrello") 4
else:                      5
    print("Prenderò i sandali")    6
```

Prenderò i sandali

# Sintassi del costrutto if else

if condizione:	1
blocco1	2
blocco1	3
blocco1	4
else:	5
blocco2	6
blocco2	7

**oppure (anche se fa un po' schifo)**

if condizione:	1
blocco1	2
blocco1	3
blocco1	4
blocco1	5
else:	6
blocco2	7
blocco2	8

**L'indentazione dei due blocchi non deve essere uguale**

# Aumentiamo il numero di opzioni

`elif` è un'abbreviazione di `else if`

```
def commenti_voto(voto):                                1
    print("Il voto e' " + str(voto) + ".")              2
    if voto < 18:                                        3
        print("Mi dispiace...")                        4
    elif voto == 18:                                    5
        print("Appena sufficiente.")                   6
    elif voto < 24:                                     7           # tra 19 e 24
        print("OK, ma potevi fare meglio.")            8
    elif voto == 30:                                    9
        print("Congratulazioni!")                      10
    else:                                               11           # tra 25 e 29
        print("Bene!")                                 12
```

**Le condizioni vengono testate a cascata. E.g. il test nella linea 7 viene effettuato solo se quelli alle linee 3 e 5 sono falliti.**

# Aumentiamo il numero di opzioni (II)

<code>commenti_voto(15)</code>	1
<code>commenti_voto(18)</code>	2
<code>commenti_voto(23)</code>	3
<code>commenti_voto(27)</code>	4
<code>commenti_voto(30)</code>	5

Il voto e' 15.  
Mi dispiace...  
Il voto e' 18.  
Appena sufficiente.  
Il voto e' 23.  
OK, ma potevi fare meglio.  
Il voto e' 27.  
Bene!  
Il voto e' 30.  
Congratulazioni!

# elif aiuta la leggibilità del codice

## Una versione equivalente scritta senza elif

```
def commenti_voto(voto):  
    print("Il voto e'", voto)  
    if voto < 18:  
        print("mi dispiace")  
    else:  
        if voto == 18:  
            print("appena sufficiente")  
        else:  
            if voto < 24:  
                print("OK, ma potevi fare meglio")  
            else:  
                if voto == 30:  
                    print("congratulazioni!")  
                else:  
                    print("bene!")
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

# Rivediamo l'esempio con elif

```
import math 1
2
def eqsecondograde(A,B,C): 3
    """Risolve equazioni di 2o grado  $A x^2 + B x + C = 0$ """ 4
    Delta = B*B - 4*A*C 5
    if A==0: 6
        print("Non è un'equazione propria di 2o grado") 7
    elif Delta < 0: 8
        print("Nessuna soluzione") 9
    else: 10
        # Utilizzo la formula standard 11
        sol1 = ( -B - math.sqrt(Delta) ) / 2*A 12
        sol2 = ( -B + math.sqrt(Delta) ) / 2*A 13
        print("Soluzioni: ",sol1,sol2) 14
```

# Miglioriamo l'esempio

- ▶ gestione dell'approssimazione
- ▶ gestione dei casi

```
import math 1
import sys 2
3
def eqsecondogrado(A,B,C): 4
    """Risolve equazioni di 2o grado  $A x^2 + B x + C = 0$ """ 5
    Epsilon=sys.float_info.epsilon 6
    Delta = B*B - 4*A*C 7
    if A==0: 8
        print("Non è un'equazione propria di 2o grado") 9
    elif Delta <= -Epsilon: 10
        print("Nessuna soluzione") 11
    elif -Epsilon < Delta < Epsilon: 12
        sol = -B / 2*A 13
        print("Soluzione unica: ",sol) 14
    else: 15
        # Utilizzo la formula standard 16
        sol1 = ( -B - math.sqrt(Delta) ) / 2*A 17
        sol2 = ( -B + math.sqrt(Delta) ) / 2*A 18
        print("Soluzioni: ",sol1,sol2) 19
```

# Quanto ti è chiaro il codice?

[bit.ly/INFO2018-06a](http://bit.ly/INFO2018-06a)

```
import math 1
import sys 2
3
def eqsecondogrado(A,B,C): 4
    """Risolve equazioni di 2o grado  $A x^2 + B x + C = 0$ """ 5
    Epsilon=sys.float_info.epsilon 6
    Delta = B*B - 4*A*C 7
    if A==0: 8
        print("Non è un equazione propria di 2o grado") 9
    elif Delta <= -Epsilon: 10
        print("Nessuna soluzione") 11
    elif -Epsilon < Delta < Epsilon: 12
        sol = -B / 2*A 13
        print("Soluzione unica: ",sol) 14
    else: 15
        # Utilizzo la formula standard 16
        sol1 = ( -B - math.sqrt(Delta) ) / 2*A 17
        sol2 = ( -B + math.sqrt(Delta) ) / 2*A 18
        print("Soluzioni: ",sol1,sol2) 19
```



# Espressioni Vero/Falso

# Variable booleana

Python ha due valori, True e False, di tipo booleano.

```
print( type(True) )           1
print( type(False) )         2
bocciato = False               # variabile inizializzata a False 3
print( type(bocciato) )       4
print( str(False) )          5
print( str(True) )           6
print( false )                # False con l'iniziale maiuscola 7
```

```
<class 'bool'>
<class 'bool'>
<class 'bool'>
False
True
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "/tmp/babel-pqZjtq/python-dKmyAI", line 7, in <module>
      print( false )           # False con l'iniziale maiuscola
NameError: name 'false' is not defined
```

# Operatori per espressioni booleane

## Confronti tra valori

- ▶ `==` (uguale) `!=` (diverso)
- ▶ `<`, `>`, `<=`, `>=`

## Operazioni logiche

- ▶ `not`, `and`, `or`

## Altre...

# Esempi (I)

I valori booleani possono essere usati per rappresentare il risultato di relazioni logiche

```
print(1 >= 2)
```

1

False

```
print( 1 == (2 - 1) )
```

1

True

```
print ('ia' in 'ciao')  
print ('io' in 'ciao')
```

1

2

True  
False

## Esempi (II)

Delta = -2.1	1
A = 4	2
print( Delta < 0 )	3
print( A == 0)	4
print( 1 > -1)	5
print( -7 < A < 10)	6

True  
False  
True  
True

# Uso di espressioni booleane

## Effettuare calcoli e assegnamenti

voto = 23	1
promosso = voto >= 18	2
print("Lo studente è stato promosso:",promosso)	3

Lo studente è stato promosso: True

## Condizioni per if / elif

sessione_finita = True	1
if promosso and sessione_finita:	2
print("Verbalizzazione")	3

Verbalizzazione

# Uguaglianza e assegnamenti

- ▶ L'operatore == determina se due operandi sono uguali
- ▶ Il simbolo = indica un assegnamento di variable

```
variabile = "valore assegnato"      1  
variabile == "altra stringa"       2  
print(variabile)                   3
```

```
valore assegnato
```

# Catene di confronti

In python è possibile scrivere

```
a1 op1 a2 op2 a3 op3 ... aN
```

dove  $op1, op2, \dots$  sono operatori di confronto. Ad esempio

```
N=3.4 1
if 0 < N < 10: 2
    print("Nell'intervallo di sicurezza") 3
```

è “equivalente” a

```
N=3.4 1
if 0 < N and N < 10: 2
    print("Nell'intervallo di sicurezza") 3
```



# Altri esempi di catene di confronti

```
x = 5
print(1 < x < 10)      # 1 < x and x < 10
print(10 < x < 20 )    # 10 < x and x < 20
print(x < 10 < x*10 < 100) # x<10 and 10< x*10 and x*x<100
print(10 > x <= 9)     # 10 > x and x <= 9
print(5 == x > 4)      # 5 == x and x>4
```

```
True
False
True
True
True
```

# Confronti tra stringhe

```
print('Mario' == 'Bruno')           1
print('Mar' < 'Mario' and 'Mar' < 'Marco') 2
print('A' < 'B')                     3
print('Z' < 'a')                     4
print('0' < '9' < 'A' < 'Z' < 'a' < 'z') 5
print('Mario' > 'Bruno')             6
```

```
False
True
True
True
True
True
```

## Quando `stringa1 < stringa2` ?

- ▶ **se, nella prima posizione in cui differiscono, il carattere di `stringa1` è più piccolo di quello di `stringa2`.**

```
print('xxxxxAyyyyy' < 'xxxxxBrrr')
```

1

True

- ▶ **se `stringa1` è un prefisso di `stringa2`**

```
print('xxxx' < 'xxxxyyy')
```

1

# Conversione verso bool

La funzione `bool(x)` converte `x` ad un valore booleano.

```
def veroofalso(x): 1
    if bool(x): 2
        print(repr(x) + ' è come True') 3
    else: 4
        print(repr(x) + ' è come False') 5
6

veroofalso('') 7 # stringa vuota è falso, le altre vere
veroofalso(0) 8 # 0 è falso, gli altri interi sono veri
veroofalso(-3) 9 # 0 è falso, gli altri interi sono veri
veroofalso('0') 10
veroofalso(0.0) 11
veroofalso(0.00000001) 12
```

```
' ' è come False
0 è come False
-3 è come True
'0' è come True
0.0 è come False
1e-08 è come True
```

# Condizione if e elif non booleana

```
def veroofalso(x):
    if x:
        print(repr(x) + ' è come True')
    else:
        print(repr(x) + ' è come False')

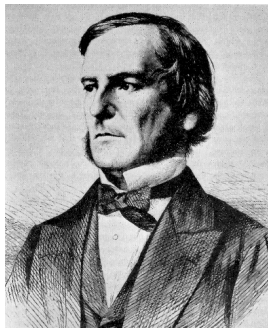
veroofalso('')      # stringa vuota è falsa, le altre vere
veroofalso(0)       # 0 è falso, gli altri interi sono veri
veroofalso(-3)      # 0 è falso, gli altri interi sono veri
veroofalso('0')
veroofalso(0.0)
veroofalso(0.00000001)
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12

```
' ' è come False
0 è come False
-3 è come True
'0' è come True
0.0 è come False
1e-08 è come True
```

# La logica booleana

# George Boole (1815–1864)



Fondatore della logica matematica

- ▶ studio formale dei ragionamenti usati in matematica
- ▶ uso di manipolazioni algebriche per concetti logici

# Operatori logici

Operatori che combinano espressioni booleane.

	Matematica	Python
negazione	$\neg x$	not x
congiunzione	$x \wedge y$	x and y
disgiunzione	$x \vee y$	x or y



# Negazione logica $\neg x$

Assume il valore opposto della variable  $x$

x	not x
False	True
True	False

```
porta_chiusa = False           1
porta_aperta = not porta_chiusa 2
print(porta_aperta)           3
```

True

Domanda: a cosa è uguale `not not x`?

# Congiunzione logica $x \wedge y$

La congiunzione è vera quando  $x$  e  $y$  sono entrambi veri.

x	y	x and y
False	False	False
True	False	False
False	True	False
True	True	True

**Esercizio:** Quando vale `True` l'espressione seguente?

`a1 and a2 and a2 and a4 and a5`

1

# Esempio di congiunzione logica

<code>vento = True</code>	1
<code>neve = True</code>	2
<code>tormenta = vento and neve</code>	3
<code>print(tormenta)</code>	4

True

# Disgiunzione logica $x \vee y$

La disgiunzione è vera quando almeno uno tra  $x$  e  $y$  è vero.

x	y	x or y
False	False	False
True	False	True
False	True	True
True	True	True

**Esercizio:** Quando vale `True` l'espressione seguente?

```
a1 or a2 or a2 or a4 or a5
```

1

# Esempio di disgiunzione logica

```
nuvoloso   = True           1
pioggia    = False          2
brutto_tempo = pioggia or nuvoloso  3
print(brutto_tempo)         4
```

**True**

# Associatività e Commutatività

Un operatore tra due operandi, chiamiamolo  $\circ$ , si dice

- ▶ associativo, quando  $(a \circ b) \circ c = a \circ (b \circ c)$
- ▶ commutativo, quando  $a \circ b = b \circ a$

**Esercizio:** dimostrare che se un operatore  $\circ$  è associativo e commutativo, allora comunque vengano messe le parentesi o ordinati gli operandi nella seguente espressione

$$a_1 \circ a_2 \circ a_3 \cdots a_{n-1} \circ a_n$$

il valore dell'espressione non cambia.

# Differenze con il linguaggio naturale

Nel linguaggio naturale si usa `or` in modo diverso

vado al mare o in montagna

intendendo alternative esclusive.

Invece l'`or` logico funziona in maniera differente, ne senso che il risultato è vero anche se entrambe le opzioni sono vere.

## Or esclusivo $x \oplus y$

L'or esclusivo (XOR) è vero quando esattamente uno tra  $x$  e  $y$  è vero. Lo XOR è denotato anche come  $x \oplus y$ .

$x$	$y$	$x \oplus y$
False	False	False
True	False	True
False	True	True
True	True	False

**Esercizio:** Quando vale True l'espressione seguente?

$a1 \oplus a2 \oplus a2 \oplus a4 \oplus a5$

1



## Il not precede and che precede or

E.g. lo XOR tra  $x$  e  $y$  si può anche scrivere come

$$x \oplus y = ((\neg x) \wedge y) \vee (x \wedge (\neg y))$$

```
def exclusive_or(x,y):           1
    return not x and y or x and not y    2

print(exclusive_or(False,False))      3
print(exclusive_or(True,False))        4
print(exclusive_or(False,True))        5
print(exclusive_or(True,True))         6
                                         7
```

# Esercizi

**Esercizio:** Addizione e moltiplicazione sono commutativi e associativi. Verificate.

**Esercizio:** XOR,  $\wedge$  e  $\vee$  sono commutativi e associativi. Verificate.

# Tabelle di verità

**Formula booleana: formula di variabili booleane e operatori booleani.**

$$(x \vee \neg y) \vee (\neg x \wedge y)$$

x	y	(x or (not y)) or ( (not x) and y)
False	False	True
True	False	True
False	True	True
True	True	True

# Regole di de Morgan

$\neg(x \vee y)$  è uguale a  $\neg x \wedge \neg y$

ed anche

$\neg(x \wedge y)$  è uguale a  $\neg x \vee \neg y$

**Esercizio:** verificare usando le tabelle di verità

- scrivere le tabelle delle quattro formule
- ogni formula ha due variabili: la tabella ha 4 righe

# Distributività

$x \wedge (y \vee z)$  è uguale a  $(x \wedge y) \vee (x \wedge z)$

ed anche

$x \vee (y \wedge z)$  è uguale a  $(x \vee y) \wedge (x \vee z)$

**Esercizio: verificare usando le tabelle di verità**

- ▶ scrivere le tabelle delle quattro formule
- ▶ ogni formula ha tre variabili: la tabella ha 8 righe