approfondimento-sui-booleani

June 5, 2023

0.1 Il vero e il falso

Quando abbiamo introdotto il costrutto `if`, abbiamo visto che la condizione di scelta è un'espressione che può essere vera o falsa. Questo introduce un nuovo tipo di dato, detto dato **booleano** (tecnicamente `bool`) che può assumere due valori: `True` e `False`.

[1]:
```
4 > 7
```

[1]:
```
False
```

[2]:
```
a = 12
a + 10 < a*3
```

[2]:
```
True
```

[3]:
```
esito = a >= 7
esito
```

[3]:
```
True
```

[4]:
```
esito + "aaa"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [4], in <cell line: 1>()
----> 1 esito + "aaa"

TypeError: unsupported operand type(s) for +: 'bool' and 'str'
```

Visto? Un valore booleano può essere memorizzato in una variabile, come qualunque altro valore. Nell'ultimo comando avete visto che Python segnala `TypeError` visto che non si può sommare un `bool` con una `str`. Che succede se sommate un `bool` e un `int`?

0.1.1 Confronti: uguale o diverso?

Gli operatori `==` e `!=` determinano rispettivamente se due espressioni hanno o meno lo stesso valore. Permettono di creare un'espressione booleana da due espressioni di tipo non necessariamente booleano.

- l'espressione `expr1 == expr2` vale `True` se `expr1` ha lo stesso valore di `expr2`, e vale `False` altrimenti.
- l'espressione `expr1 != expr2` è la negazione logica di `expr1 == expr2`

[ ]:
```
print( "casa" == "ca" + "sa" )
print( "gatto" != 4)
print( "5" == 5)
print( "a"*4  != 'aaaa')
print( "apici"  == 'apici')  # gli apici non contano
```

[5]:
```
x = "testo breve"
risultato = len(x) < 100

if risultato:
    print("Il testo in x è corto.")
```

```
Il testo in x è corto.
```

Ricordate: L'operatore boolean di uguaglianza è `==`, mentre l'assegnamento è denotato con `=`. Usano quello sbagliato, otterrete un errore o peggio un comportamento errato del programma.

0.1.2 Interludio: uguaglianza di due `float`

Se si hanno due espressioni `float` e si vuole capite se sono uguali, non è mai consigliabile confrontarle direttamente con `!=` e `==`. - Le operazioni sui `float` sono approssimate. - Espressioni che logicamente hanno lo stesso valore, potrebbero non averlo in Python.

[6]:
```
piccolo = 0.0000001
grandep = 10 ** 20
granden = -grandep

x = piccolo + grandep + granden
y = grandep + granden + piccolo

print("x è :", x)
print("y è :", y)
```

```
x è : 0.0
y è : 1e-07
```

[7]:
```
if x == y:
    print("Approssimazione sufficiente.")
else:
    print("Approssimazione insufficiente.")
```

```
Approssimazione insufficiente.
```

[8]:
```
tolleranza = 0.000001
if abs(x - y) < tolleranza:
```

```
    print("Approssimativamente uguali.")
else:
    print("Diversi.")
```

```
Approssimativamente uguali.
```

0.1.3 Confronti: maggiore e minore?

Abbiamo già visto confronti tra numeri con `<` e `>`. In realtà Python ha quattro operatori per confronti del genere - `>` maggiore di; - `<` minore di; - `>=` maggiore di o uguale a; - `<=` minore di o uguale a;

Il loro significato non dovrebbe essere difficile da immaginare. Vediamo degli esempi.

[10]:
```
x = 7
print( x - 5 > 2)
print( x - 5 >= 2)
```

```
False
True
```

[12]:
```
y = int(input())
if y < 0:
    print("Numero negativo.")
elif y > 0:
    print("Numero positivo.")
else:
    print("Zero.")
```

```
 12
```

```
Numero positivo.
```

[11]:
```
a = 10
b = 20
if a <=b:
    print("La coppia (a,b) è ordinata")
```

```
La coppia (a,b) è ordinata
```

Possiamo confrontare solamente i numeri? No! In realtà le stringhe in Python sono confrontabili rispetto all'ordine lessicografico. Senza dare troppi dettagli su questo tipo di ordine, possiamo dire che - I caratteri delle cifre `0-9` sono minori dei caratteri `A-Z` che sono minori di `a-z`. Il carattere spazio `' '`è anche minore di `"0"`. - La stringa di lunghezza zero `""` è la stringa più piccola di tutte

[15]:
```
"0" < "1"
```

[15]:
```
True
```

[16]:
```
"9" < "A"
```

[16]: True

[17]:
```
"A" < "B"
```

[17]: True

[18]:
```
"Z" < "a"
```

[18]: True

[19]:
```
"a" < "b"
```

[19]: True

[20]:
```
"gatto" >= "casa"
```

[20]: True

[21]:
```
"abaco" < "ala"
```

[21]: True

[22]:
```
"AdaAfd687fytch8d7382tu" < "ciao"
```

[22]: True

[23]:
```
"Casa" < "casa"  # maiuscole
```

[23]: True

[24]:
```
" spazio" >= "spazio" # spazio
```

[24]: False

[25]:
```
"casa" < "casale"  # prefisso
```

[25]: True

[26]:
```
"" < " "
```

[26]: True

Non si possono invece confrontate numeri e stringhe. Se lo fate Python segnalerà un `TypeError`.

[28]:
```
"quattro" < 5
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
```

```
Input In [28], in <cell line: 1>()
----> 1 "quattro" < 5

TypeError: '<' not supported between instances of 'str' and 'int'
```

0.1.4 Operatori Logici: Negazione, Or, And

Le espressioni a valore booleano possono anche essere combinate tra loro con gli operatori logici:

Negazione logica `not` Se `expr` è un'espressione booleana, allora `not expr` è l'espressione booleana che è vera se e solo se `expr` è falsa.

[29]:
```
print(True)
print(not True)
print(False)
print(not False)
print( not 10 < 3)
```

```
True
False
False
True
True
```

[30]:
```
x = 12
res = x < 10
print( not res)
```

```
True
```

Disgiunzione logica `or` Consideriamo due espressioni booleane `expr1` e `expr2`: il valore di `expr1 or expr2` è `True` quando **almeno una** delle due espressioni componenti è `True`.

Notate che questo è differente dal linguaggio di tutti i giorni. In italiano se diciamo "questo **o** quello" indichiamo due scelte mutuamente esclusive. Non è così in Python.

È possibile concatenare tanti `or` scrivendo

```
expr1 or expr2 or expr3 or ... or exprN
```

e anche in questo caso l'espressione finale è `True` quando **almeno una** delle espressioni componenti è `True`. Se invece **tutte** le espressioni componenti sono `False`, il risultato è `False`.

[33]:
```
print( False or False )
print( True  or False )
print( False or True  )
print( True  or True  )
```

```
False
True
```

```
True
True
```

[32]:
```
piove = False
nuvoloso = True

if piove or nuvoloso:
    print("Prendo l'ombrello.")
```

```
Prendo l'ombrello.
```

[31]:
```
verifica = 2 < 1 or 3 > 5 or 12 % 2 == 0 or 7*2 == 1
print(verifica)
```

```
True
```

Congiunzione logica `and` Consideriamo due espressioni booleane `expr1` e `expr2`: il valore di `expr1 and expr2` è `True` quando **entrambe** le due espressioni sono `True`.

È possibile concatenare tanti `and` scrivendo

```
expr1 and expr2 and expr3 and ... and exprN
```

e anche in questo caso l'espressione finale è `True` quando **tutte** le espressioni componenti sono `True`. Se invece **anche una sola** delle espressioni componenti è `False`, il risultato è `False`.

[37]:
```
print( False and False )
print( True  and False )
print( False and True  )
print( True  and True  )
```

```
False
False
False
True
```

[38]:
```
piove = False
ventoso = True

if piove and ventoso:
    print("È tempesta")
```

Attenzione alle maiuscole e minuscole: le grafie di `and`, `or`, `not`, `True` e `False` devono essere esattamente queste. Altre grafie non sono ammesse.

[34]:
```
True AND False
```

```
  Input In [34]
    True AND False
         ^
```

```
SyntaxError: invalid syntax
```

[35]:
```
not true
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [35], in <cell line: 1>()
----> 1 not true

NameError: name 'true' is not defined
```

[36]:
```
1 > 3 OR 3 > 1
```

```
  Input In [36]
    1 > 3 OR 3 > 1
          ^
SyntaxError: invalid syntax
```

0.1.5 Riassunto

Abbiamo visto

- valori booleani `True` e `False`;
- espressioni booleane con confronto di valori `==` e `!=`;
- che non si dovrebbero usare con in `float`;
- espressioni booleane con confronti `<`, `>`, `<=`, `>=`;
- confronti tra stringe e ordine lessicografico;
- operatori logici `not`, `and`, `or`.