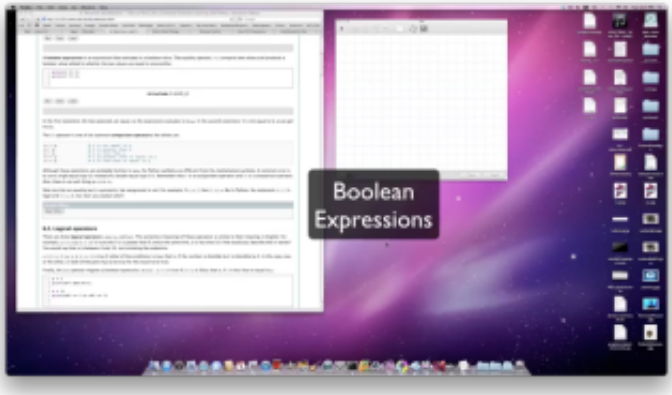
Boolean Values and Boolean Expressions



The Python type for storing true and false values is called `bool`, named after the British mathematician, George Boole. George Boole created *Boolean Algebra*, which is the basis of all modern computer arithmetic.

There are only two **boolean values**. They are `True` and `False`. Capitalization is important, since `true` and `false` are not boolean values (remember Python is case sensitive).

```
1 print(True)
2 print(type(True))
3 print(type(False))
4
```

ActiveCode: 1 (ch05_1)

Run

```
True
<type 'bool'>
<type 'bool'>
```

Note

Boolean values are not strings!

It is extremely important to realize that True and False are not strings. They are not surrounded by quotes. They are the only two values in the data type `bool`. Take a close look at the types shown below.

```
1 print(type(True))
2 print(type("True"))
3
```

ActiveCode: 2 (ch05_1a)

Run

```
<type 'bool'>
<type 'str'>
```

A **boolean expression** is an expression that evaluates to a boolean value. The equality operator, `==`, compares two values and produces a boolean value related to whether the two values are equal to one another.

```
1 print(5 == 5)
2 print(5 == 6)
3
```

ActiveCode: 3 (ch05_2)

Run

```
True
False
```

In the first statement, the two operands are equal, so the expression evaluates to `True`. In the second statement, 5 is not equal to 6, so we get `False`.

The `==` operator is one of six common **comparison operators**; the others are:

```
x != y               # x is not equal to y
x > y                # x is greater than y
x < y                # x is less than y
x >= y               # x is greater than or equal to y
x <= y               # x is less than or equal to y
```

Although these operations are probably familiar to you, the Python symbols are different from the mathematical symbols. A common error is to use a single equal sign (`=`) instead of a double equal sign (`==`). Remember that `=` is an assignment operator and `==` is a comparison operator. Also, there is no such thing as `=<` or `=>` .

Note too that an equality test is symmetric, but assignment is not. For example, if `a == 7` then `7 == a` . But in Python, the statement `a = 7` is legal and `7 = a` is not. (Can you explain why?)

Check your understanding

sel-1: Which of the following is a Boolean expression? Select all that apply.

- [x] a) True
- [x] b) 3 == 4
- [ ] c) 3 + 4
- [x] d) 3 + 4 == 7
- [ ] e) "False"

Check Me | Compare Me

Correct!
a: True and False are both Boolean literals.
b: The comparison between two numbers via == results in either True or False (in this case False), both Boolean values.
d: 3+4 evaluates to 7. 7 == 7 then evaluates to True, which is a Boolean value.

Logical operators

There are three **logical operators**: `and` , `or` , and `not` . The semantics (meaning) of these operators is similar to their meaning in English. For example, `x > 0 and x < 10` is true only if `x` is greater than 0 *and* at the same time, x is less than 10. How would you describe this in words? You would say that x is between 0 and 10, not including the endpoints.

`n % 2 == 0 or n % 3 == 0` is true if *either* of the conditions is true, that is, if the number is divisible by 2 *or* divisible by 3. In this case, one, or the other, or both of the parts has to be true for the result to be true.

Finally, the `not` operator negates a boolean expression, so `not x > y` is true if `x > y` is false, that is, if `x` is less than or equal to `y` .

```
1 x = 5
2 print(x>0 and x<10)
3
4 n = 25
5 print(n%2 == 0 or n%3 == 0)
6
```

ActiveCode: 4 (chp05_3)

Run

Save Load

```
True
False
```

Common Mistake!

There is a very common mistake that occurs when programmers try to write boolean expressions. For example, what if we have a variable `number` and we want to check to see if its value is 5,6, or 7. In words we might say: “number equal to 5 or 6 or 7”. However, if we translate this into Python, `number == 5 or 6 or 7`, it will not be correct. The `or` operator must join the results of three equality checks. The correct way to write this is `number == 5 or number == 6 or number == 7`. This may seem like a lot of typing but it is absolutely necessary. You cannot take a shortcut.

Check your understanding

sel-2: What is the correct Python expression for checking to see if a number stored in a variable x is between 0 and 5.

- a) x > 0 and < 5
- b) 0 < x < 5
- c) x > 0 or x < 5
- d) x > 0 and x < 5

Check Me Compare Me

Correct!! Yes, with an and keyword both expressions must be true so the number must be greater than

0 an less than 5 for this expression to be true.

Precedence of Operators

We have now added a number of additional operators to those we learned in the previous chapters. It is important to understand how these operators relate to the others with respect to operator precedence. Python will always evaluate the arithmetic operators first (** is highest, then multiplication/division, then addition/subtraction). Next comes the relational operators. Finally, the logical operators are done last. This means that the expression `x*5 >= 10 and y-6 <= 20` will be evaluated so as to first perform the arithmetic and then check the relationships. The `and` will be done last. Although many programmers might place parenthesis around the two relational expressions, it is not necessary.

The following table summarizes the operator precedence from highest to lowest. A complete table for the entire language can be found in the Python Documentation (http://docs.python.org/py3k/reference/expressions.html#expression-lists).

Level	Category	Operators
7(high)	exponent	**
6	multiplication	*,/,//,%
5	addition	+,-
4	relational	==,!=,<=,>=,>,<
3	logical	not
2	logical	and
1(low)	logical	or

Note

This workspace is provided for your convenience. You can use this activecode window to try out anything you like.

```
1
2
```

ActiveCode: 5 (scratch_06_01)

Run

Check your understanding

sel-3: Which of the following properly expresses the precedence of operators (using parentheses) in the following expression: 5*3 > 10 and 4+6==11

- a) ((5*3) > 10) and ((4+6) == 11)
- b) (5*(3 > 10)) and (4 + (6 == 11))
- c) ((((5*3) > 10) and 4)+6) == 11
- d) ((5*3) > (10 and (4+6))) == 11

Check Me | Compare Me

Correct!! Yes, * and + have higher precedence, followed by > and ==, and then the keyword "and"