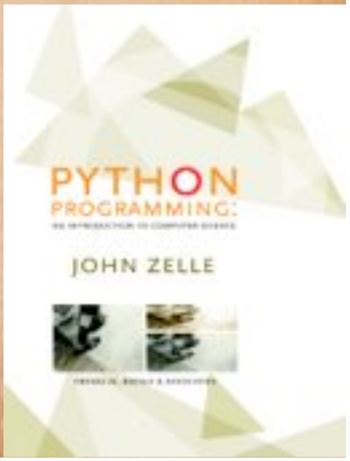
Python Programming: An Introduction To Computer Science

Chapter 8
Booleans



Coming up: Computing with Booleans

Computing with Booleans

- `if` and `while` both use Boolean expressions.
- Boolean expressions evaluate to `True` or `False`.
- So far we've used Boolean expressions to compare two values, e.g. (`while x >= 0`)

Boolean Operators

- The Boolean operators `and` and `or` are used to combine two Boolean expressions and produce a Boolean result.
- `<expr> and <expr>`
- `<expr> or <expr>`

Expressions versus Statements

- In the last slide we used the term “expression”.
- The difference between an expression and a statement is:
  - Expressions are something (they evaluate to a value)
    - e.g. x*7+(y**3) , t==True or v1 != v2
  - Statements do something
    - print “hello”
    - x = x * 7

Statement or Expression

x = 9	Statement
45 % 78 == 0	Expression
myFunction(‘potato’)	Statement
x, y = 5, 6	Statement
print “%20s” %(‘potato’)	Statement
“%20s” %(‘potato’)	Expression

Boolean Operators

- The Boolean operators `and` and `or` are used to combine two Boolean expressions and produce a Boolean result.
- `<expr> and <expr>`
- `<expr> or <expr>`

Boolean Operators

- The `and` of two expressions is true exactly when both of the expressions are true.
- We can represent this in a *truth table*.

P	*Q*	*P and Q*	*P or Q*
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

Boolean Expressions

- The only time and is true is when both expressions are true
- The only time `or` is false is when both expressions are false.
- Also, note that `or` is true when both expressions are true. This isn't how we normally use "or" in language.

Boolean Operators

- Consider `a or not b and c`
- How should this be evaluated?
- The order of precedence, from high to low, is `not`, `and`, `or`.
- This statement is equivalent to `(a or ((not b) and c))`
- Since most people don’t memorize the the Boolean precedence rules, use parentheses to prevent confusion.

Boolean Operators

- To test for the co-location of two points, we could use an `and`.

-
  ```
  if p1.getX() == p2.getX() and
     p2.getY() == p1.getY():
      # points are the same
  else:
      # points are different
  ```
- The entire condition will be true *only* when both of the simpler conditions are true.

Boolean Operators

- Say you’re writing a racquetball simulation. The game is over as soon as either player has scored 15 points.
- How can you represent that in a Boolean expression?
- `scoreA == 15 or scoreB == 15`
- When either of the conditions becomes true, the entire expression is true. If neither condition is true, the expression is false.

Coming up: Boolean Operators

Boolean Operators

- We want to construct a loop that continues as long as the game is **not** over.
- You can do this by taking the negation of the game-over condition as your loop condition!

-
  ```
  while not(scoreA == 15 or scoreB == 15):
      #continue playing
  ```

Boolean Operators

- Some racquetball players also use a shutout condition to end the game, where if one player has scored 7 points and the other person hasn’t scored yet, the game is over.

-
  ```
  while not(scoreA == 15 or scoreB == 15 or \
    (scoreA == 7 and scoreB == 0) or \
    (scoreB == 7 and scoreA == 0):
        #continue playing
  ```

Boolean Operators

- Let’s look at volleyball scoring. To win, a volleyball team needs to win by at least two points.
- In volleyball, a team wins at 15 points
- If the score is 15 – 14, play continues, just as it does for 21 – 20.
- `(a >= 15 and a - b >= 2) or (b >= 15 and b - a >= 2)`
- `(a >= 15 or b >= 15) and abs(a - b) >= 2`

Boolean Algebra

- The ability to formulate, manipulate, and reason with Boolean expressions is an important skill.
- Boolean expressions obey certain algebraic laws called *Boolean logic* or *Boolean algebra.*

Boolean Algebra

Algebra	Boolean algebra
a * 0 = 0	*a* and false == false
a * 1 = *a*	*a* and true == *a*
a + 0 = *a*	*a* or false == *a*

- `and` has properties similar to multiplication
- `or` has properties similar to addition
- `0` and `1` correspond to false and true, respectively.

Boolean Algebra

Anything `or`*ed with true is true:*

```
a or true == true
```

What is anything and'd with False?

Both `and` *and* `or` *distribute:*

```
a or (b and c) == (a or b) and (a or c)
a and (b or c) == (a and b) or (a and c)
```

Similar to algebra!
a and (b or c)
a*(b+c) ==
(a*b) + (a*c)

Double negatives cancel out:

```
not(not a) == a
```

DeMorgan's laws:

```
not(a or b) == (not a) and (not b)
not(a and b) == (not a) or (not b)
```

Short Circuit

```
x = 7
y = 8
if x < 10 or y > 9:
    print “Hello”
```

Question: Does Python need to check if y > 9?

No! Once it knows that x < 10 is True, anything Or’d with True is True!

Short Circuit

```
x = 57
y = 8
if x < 10 and y > 9:
    print “Hello”
```

Question: Does Python need to check if y > 9?

No! Once it knows that x < 10 is False, anything And’d with False is False!

Short Circuit

This is called “short circuiting”. If possible, only the first part of a boolean expression will be executed. This has consequences!

```
x = 88
if x < 10 and getAnswer() == ‘go’:
   print “Hello”
```

getAnswer is NOT called at all in this code!

Boolean Algebra

- We can use Boolean rules to simplify our Boolean expressions.
-
  ```
  while not(scoreA == 15 or scoreB == 15):
      #continue playing
  ```
- This is saying something like “While it is not the case that player A has 15 or player B has 15, continue playing.”
- Applying DeMorgan’s law:

  ```
  while (not scoreA == 15) and (not scoreB == 15):
      #continue playing
  ```

Boolean Algebra

- This becomes:

```
while scoreA != 15 and scoreB != 15
    # continue playing
```

- Isn’t this easier to understand? “While player A has not reached 15 and player B has not reached 15, continue playing.”

Applying DeMorgan's Laws

- Negate each element
- change *and* to *or*
- change *or* to *and*

Simplify:

not(x < 8 and y > 7)

--- not(x < 8) or not(y > 7)

--- x >= 8 or y <= 7

Boolean Algebra

- Sometimes it's easier to figure out when a loop should stop, rather than when the loop should continue.
- In this case, write the loop termination condition and put a `not` in front of it. After a couple applications of DeMorgan's law you are ready to go with a simpler but equivalent expression.

Dan’s Final Word

- When in doubt, simplify as much as you can, then add comments and explain the reasoning behind the Boolean statement!

  Keep is simple!