Introduction to Python Programming

Andrew J. Pounds, Ph.D.

Exercise Four: Conditional Expressions

As you know computers basically work off of “Boolean” logic -- true/false values represented electronically by signals being either on or off respectively. Real computer programming starts when you learn to manipulate these signals to control the flow of logic through a program.

1. **Boolean Constants**

   Python contains two Boolean constants: True and False (note that they are capitalized).

2. **Relational (or Comparison) Operators**

   If is often necessary to compare the values of two quantities using operators like…

   $$<, \leq, >, \geq, =, \neq$$

   With the result of the operation being a boolean value of True or False. For example the following expression

   $$3 < 4$$

   would be True. Python provides the following relational operators

Operator	Description	Result with $a < b$
==	Are operands equal	False
!=	Are operands not equal	True
<>	Are operands not equal	True
>	Is left operand greater than right operand	False
<	Is left operand less than right operand	True
>=	Is left operand greater than or equal to right operand	False

<=	Is left operated left than or equal to right operand	True

3. Logical Operators

It is also frequently necessary to compare boolean statements with each other. For example, the following mathematical statement

$$(3 < 4) \wedge (\neg(4 < 2) \vee (2 > 3))$$

Asks the question “is 3 less than 4 AND if either 4 is not less than 2 OR if 2 is greater than three. This expression would result in True being returned. In python the above statement would be written as

```
(3<2) and ( not( 4 < 2 ) or ( 2 > 3) )
```

The operators and, or, and not are the three logical operators provided by Python. As seen in the example, they can be combined to evaluate complicated expressions. The following table summarizes their possible results.

A	B	A and B	A or B	not(A and B)	not(A or B)
True	True	True	True	False	False
True	False	False	True	True	False
False	True	False	True	True	False
False	False	False	False	True	True

3. Program Flow Control

The reason computer programs need these types of operators is so that the expressions in a computer program can be executed pursuant to specific conditions being met. As an example, imagine you are writing a program to sound an alarm to remind you to go to class. You would only want that alarm executing on the days that you had a specific class. In other words -- IF it is a class day, sound the alarm.

To allow for this type of programming, python has the IF statement that takes a boolean (logical) argument. For example

```
A = 5
B = 3
if ( A > B ):
   print "A is greater than B"
   print A
```

Will result in the output…

```
A is greater than B
5
```

Take particular note of the indention in the code segment above. Inside of the if clause all of the text that is indented will be executed.

You make more complicated decision structures by using the “else if” statement like so…

```
if ( boolean expressions one):
  … do all this indented stuff
else if ( another boolean expression):
  … do all this indented stuff
else if ( a different  boolean expressions):
  … do all this indented stuff
else:
  … do this indented stuff if none of the other conditions are met
```

Is is also possible to NEST if statements (put if blocks inside of other if blocks). This can become very difficult to debug, but in some cases it is the only way to parse very difficult logic structure.

It is worth noting that the indented statements corresponding to the FIRST true boolean expression are executed and the rest are not. For example.

```
A = 5
B = 6
C = 7
if ( A < B ):
  print “first block”
else if ( ( A < B) and (B < C)):
  print “second block”
else:
  print “third block”
```

Will only print “first block” because even though the second boolean statement is true, the first block was triggered first.