3.2 Booleans

The *Boolean* data type consists of just two values: True and False. In Python this is implemented as a sub-type of the integers, where 0 corresponds to True and any non-zero value corresponds to False. You can actually use integer operations on Boolean values if you choose (such as True + False), but this will just make your programs harder to read. The most common ways to produce Boolean values is through the comparison operators (such as < or ==), and through the membership search operations on strings and lists.

Sometimes you need to build compound conditions out of simpler ones. For example, if you want to check that a number x is between 1 and 10, you can do so with

```
if x >= 1 and x <= 10:
```

In this example we took two complete Boolean conditions and connected them with the operator **and**.

There are three Boolean operators: **and**, **or**, and **not**.

Symbol	Meaning	Example	Result
and	x and y is True only when both sides are True; if either side is False the result is False.	$x = 23$	
		$x < 10$ and $x > 0$	False
		$x > 10$ and $x < 100$	True
or	x or y is True when either side is True; it is only Fals when both sides are False.	$x = 23$	
		$x < 10$ or $x > 0$	True
		$x > 10$ or $x < 100$	True
		$x < 10$ or $x < 20$	False
not	not x has the opposite value of x: it is False when x is True and True when x is False.	$x = 23$	
		not $x < 10$	True
		not $x > 20$	False
		not $x == 23$	False

You can build very complex expressions using these operators. In general, if you aren't sure how Python will group together sub-expressions, it is best to use parentheses to force the expression to be parsed in the way you intend. For example, the following condition describes the numbers 6, 7, 8, 9, 21, 22, 23;

```
if ((x > 5) and (x < 10)) or ((x >20 and (x < 24)):
```

One comforting thing about the way Python implements the Boolean operators is that it stops evaluating as soon as it knows the answer. For example, the expression a **and** b is False if either of its operands is False, so if a is False there is no need to evaluate b at all. If s is a string and n is an integer might have the code

```
if n < len(s) and s[n] == 'b':
        print( "The string contains letter 'b'." )
```

If n is *not* less than the length of the string, the right side of the expression s[n] == 'b' would normally cause the program to crash. It doesn't in this case because Python never evaluates the right side of the expression; it knows the result just from the left side, n < **len**(s). Note that if we wrote the expression in the opposite order,

```
if s[n] == 'b' and n < len(s):
        print( "The string contains letter 'b'." )
```

then the program will crash when n gets a value large than the length of s. The **or** operator is implemented in the same way: if its left operand evaluates to True it doesn't bother to evaluate the right operand since it knows the result of the full expression is True.

As a final example we will develop a program that inputs three numbers and *sorts* them, which means it puts them into increasing order. There are many sorting algorithms, but with just three numbers we can take a common-sense approach. First we need to input the three numbers, which we will store in variables a, b, and lstinlinec. This calls for three input statements, each of which will look like

```
a = eval( input("Enter a number: ") )
```

Next, we need to determine the ordering of lstinlinea, b, and lstinlinec. There are factorial (n) ways to order n numbers (There are lstinlinen different choices for the first number; for each of these there are n−1 choices for the second number, and so forth....); since factorial (3) is 6, we know that there are 6 possible orderings for our numbers. One way to write the program would be to have 6 variations of a statement such as

```
if a <=b and b <= c:
    print(a, b, c)
```

We will take an alternative approach that finds the smallest number and then orders the other two. This uses three variations of the following statement:

```
if a <= b and a <= c: # a is the smallest
    if b <= c:
        print(a, b, c)
    else:
        print(a, c, b)
```

Note the use of **else** here. If a is the smallest of the three numbers and b is not less than or equal toc, then c must be smaller than b.

Here is the entire program

```
# This inputs 3 numbers and prints
# then out in inceasing order.

def main():
    a = eval( input("Enter a number: ") )
    b = eval( input("Enter another number: ") )
    c = eval( input("Enter a third number: ") )
    print( "The correct ordering of those is ", end="")
    if a <= b and a <= c: # a is the smallest
        if b <= c:
            print(a, b, c)
        else:
            print(a, c, b)
    elif b <= a and b <= c:  # b is the smallest
        if a <= c:
            print(b, a, c)
        else:
            print(b, c, a)
    else: # c is the smallest
        if a <= b:
            print( c, a, b)
        else:
            print(c, b, a)

main()
```

Program 3.2.1: Sorting three numbers