**Conditionals and boolean logic**

Before we get to conditionals in python 3 it is a good idea to address boolean logic. We can test a statement to see if it is true or false using a variety of operators, used in a similar fashion to the mathematical operators we saw last week.

**Booleans.py**

print(1 == 1)

print(1 == 0)

print(10 % 2 == 0)

print(14 % 2 == 1)

print(2 > 3)

print(8 < 16)

print(19 >= 19.0)

print(4 <= 3)

number = 10

print(5 < number < 15)

print(20 < number < 30)

print((number > 5) and (number < 15))

print((number == 10) or (number == 20))

print(number != 10)

print(number != 15)

print("apple" == "apple")

print("orange" == 7)

This little script demonstrates the various comparisons we can do. We can even compare floats with integers and strings with strings. You’ll see that when you run this script each operation will return either ‘True’ or ‘False’. With this we are able to carry out complex decision making and execute different blocks of code depending on the result of one of these operations.

**Stock\_checker.py**

apples = 15

if(apples < 10):

print("We do not have enough apples.")

elif(10 <= apples < 20):

print("We have enough apples.")

else:

print("We have too many apples.")

An if statement at its most basic requires just an ‘if’ but it’s good practice to add an ‘else’ to handle the opposing condition. If we need to test for alternative outcomes we can use one or more ‘elif’ statements. Be sure to add a colon at the end of each ‘if’, ‘elif’ and ‘else’ statements to let python know that we’re opening a new block of code.

It is also worth mentioning line 5. This is an alternative way of writing the following and looks somewhat cleaner.

elif(apples >= 10) and (apples < 20):

**Indentation in python**

At this point it would be a good idea to talk about how python handles blocks of code. In other languages, such as JavaScript, C++ or Java, the code within an if statement would be contained within curly brackets. Python uses indentation instead to represent a new block of code which has two benefits. It forces you to format your code properly which leads to code that is easier to read and there is less superfluous syntax cluttering your code. This concept is used elsewhere in python so be sure to indent your code properly, otherwise you may get strange errors.

Let’s write something a bit more interesting using what we just learning, but this time we’ll ask the user for some information.

**Guess\_the\_number.py**

import random

random\_number = random.randint(1, 100)

player\_choice = int(input("Choose a number between 1 and 100: "))

if(player\_choice > 100) or (player\_choice < 1):

print("That number is out of range")

else:

if(player\_choice > random\_number):

difference = player\_choice - random\_number

print("That number was too big.")

print("Number was {}, {} off".format(random\_number, difference))

elif(player\_choice < random\_number):

difference = random\_number - player\_choice

print("That number was too small.")

print("Number was {}, {} off".format(random\_number, difference))

else:

print("You guessed right!")

This little game is pretty basic for now but we’ll be sure to return to it later to improve it.

In order to generate the random number for the user to guess we ‘import’ the random module. This is a built-in module that gives us various tools for generating random numbers. We invoke one of these tools, or functions, on the next line. Here we create a random number between 1 and 100.

Then we ask the user to input a number between 1 and 100. We then check if the number is within range and if not we let the user know, otherwise we check to see if the number matches our randomly generated one. If the number is too big or small we let the user know how close to the random number they were.