Session 3
Control Structures

- ✓ Introduction
- ✓ If Else Conditions
- ✓ Loops

Introduction

Control structures are keywords you can use in programs to change the flow of the instructions. For example you may want to run a certain piece of code in one case but not another. In the previous session we talked about filling variables with different data and how boolean expressions can be used to compare them; in this session we will use those tools and combine them with if-else conditions as well as loops to run pieces of code in different ways than from top to bottom.

If Else Conditions

As discussed in the previous chapter **if-else** conditions are fundamental keywords for controlling the flow of a program. The if-else condition is combined with boolean conditions to allow for certain instructions to be ran if a certain condition is true. This is extremely useful, for example you will only want to log a user into your system if their password matches the one stored in your database, else you will tell the user that their password is incorrect. This level of control differs from what was previously said about programs being a set of instructions that are followed in order, now one program can behave differently based on different conditions. Let's look at an example of the if-else construct in Python before using it in some exercises. Try modifying the example program to trigger the if and the else conditions.

```
x = 20

if x is > 10:
     print "X is greater than 10"
else:
     print "X is less than 10"
```

In the above example you see two print keywords, only one of these can ever be printed because only one can be true, X cannot be greater than ten and less than ten at the same time. If you change the X variable to be less than ten you will trigger the else condition as it will fail the if condition by being false. A key part of the if-else construct in Python is the colon after each condition, without this your program will not run.

When doing if-else conditions any value that is zero is treated as false and anything that is not zero is treated as true; so for example if x is one and you do if x then it would be true, but if x is zero then the condition is false.

If-else conditions can be chained together in if-elseif blocks. This allows the programmer to test against multiple conditions, the final part of an if-elseif block is an optional else which defines what to do if all of the conditions return false. So for example and chain of if-elseifs would look like this.

```
x = 20

if x > 10:
    print "x is greater than 10"
elif x > 100:
    print "x is greater than 100"
else
    print "x is less than 10 and less than 100"
```

The above piece of code can only have one true statement so only one print line is shown.

Note

The else if is shortened to elif in Python. The final else is not a requirement to an if-else block, it is possible to just have a single if.

Exercise

This exercise will see you combining keywords, arithmetic operators and if else conditions to create a program that can say whether or not a year is a leap year. For this program you will use a special operator called the modulo operator (%). The modulo operator will divide a number and return the remainder, this makes it very useful for this exercise. The instructions go in order so a year must be tested against all of the following conditions.

The rules for determining if a year is a leap year are as follows -

- *If a year is not divisible by four then it is a regular year.*
- *Else if the year is not divisible by 100 then it is a leap year.*
- *Else if the year is not divisible by 400 then it is a regular year.*
- *Else it is a leap year.*

As an example to get you started and to show you the modulo operator the first rule can be written like this -

```
if year % 4:
     print "{0} is a regular year".format(year)
```

Save your code as Excercise_3.py. As a test, 2004 was a leap year but 2003 was not, 2232 will be a leap year.

Loops

The ability to loop in a program is crucial; for example you may have a file containing one thousand names and addresses to be entered into a database, to do that by hand would be extremely difficult and time consuming but for a computer it would take less than a second!

There are different types of loop, some are more common than others.

For Loop

The for loop is one of the most commonly used loops. It is generally used when there is a known range of values to loop over, by a range of values we mean that the number of **loop iterations** is known before the loop starts. An example of this is if we know we want to loop ten times or if we know we have ten entries in a database to loop over.
In Python the for loop is slightly different to other languages as it is closer to a **for each** loop. The for each loop is still a for loop except it reads slightly closer to English in that it loops for each item in the input variable.

```
for i in range(0, 10):
    print "Number {0}".format(i)
```

When that piece of code is run you will see a it prints a different number ten times starting from zero and ending at nine. This is because the loop assigns each element in the range zero to ten to the variable i.

Exercise

Use the for loop to print the first ten numbers in the Fibonacci Sequence. Save your program as Exercise_4.py, when it is run you should see the following output -

1
2
3
5
8
13
21
34
55
89

While Loop

If you wish your loop to run as long as a certain condition is true then you should use

the **while loop**, it can be thought of as a repeating if statement. A while loop is great when you need a loop to continue for an unknown amount of time. For example you may use a while loop to read a file of unknown size first by reading a bit of data, checking if there is any more, reading it and so on until you reach the end of the file.

Note

A while loop will not run if it's condition is already false, for example if you run -

```
while False:
    // some code
```

Nothing will happen in this case, this is why a while loop is known as a pre-test loop.

Let's have a look at an example while loop in Python. It is a simple loop that counts from one to nine much like our for loop.

```
i = 10
while i > 1:
    print "Loop number {0}".format(i)
    i -= 1
```

Note

Note the -= operator in the previous example, this simply subtracts whatever amount from the variable on the left. Alternatively there is the += operator that adds the right hand side to the left and saves the variable.

Exercise

Use a while loop to write out the song '99 bottles of beer on the wall'. The song repeats the following sentences but with decreasing number of beers!

99 bottles of beer on the wall

99 bottles of beer

take one down and pass it around

98 bottles of beer on the wall

Save your code as Exercise_5.py.

Do-While Loop

There is an extension to the while loop called a **do-while loop**. The difference between a while loop and a do-while loop is that a do-while loop will always execute at least once, this is because the while condition comes after the block to be executed.

 *Note*

Python does not support a do while loop but it is present in other languages such as C++ and Java.