7 'while' statements

In the previous notebook you have seen how boolean statements with the _if_ statement can be used to allow branching according to different conditions being met in the programme. Once the condition has been met the programme executes the appropriate code and moves on. In many coding situations you find you wish to repeatedly test a condition and execute the same code until the condition has been met.

In this notebook you will learn how to:

- use while statements to execute loops until a condition, as defined by a boolean statement, is met,
- use the *while* loop to do a simple *sin* series summation,
- use the a *break* to terminate a loop, especially in the form of a 'repeat ... until' type structure.

7.1 The 'while' statement

7.1.1 while statements and infinite loops

A *while* loop takes the form

while *boolean expression* :
...indented code

If the boolean expression is False the programme will skip the indented code after the *while* statement and continue execution of the programme. If the boolean expression is True, the indented code is executed and when the indented block is completed the programme returns to the *while* statement and follows the same procedure.

VERY IMPORTANT. From this description you can see that, unless, as result of executing the code in the loop, we reach a condition when the boolean statement is false, then the loop will continue executing endlessly!. The only way you can stop the programme from running is to interrupt the kernel (or \< CTRL C> if you are using for example anaconda or Idle).

Here is a trivial example of an infinite loop. Before you run the code(!) note how the prompt becomes *In [*]*. The * shows the code is still executing. You can interrupt the kernel by using the 'Kernel' tab on the menu at the top of the notebook.

In [ ]:

```
a=True
while a :
  b=1

print("end of programme")
```

At least the 'keyboard interrupt' shows you where you were in the code when it was stopped.

Now we are aware of this pitfall let's see a simple example of a *while* loop in practise. (To avoid getting into the loop on restarting this notebook you may wish to change the code above to start with *a=False* !). Look at carefully at and run the code below.

In [ ]:

```
i=10                          # Calculate factorial of i
fact=1
while (i > 0) :
    fact = fact*i
    i = i-1

print (fact)
```

We have already seen how we could calculate a factorial using a for loop (which is preferable) but let's see how this version works.

Firstly, we are going to set the number for factorial calculation in _i_. We have also set our starting value for the factorial outside the loop. We can see already that the loop will execute at least once as _i_ is greater that zero. Note also why we need to set an initial value of 1 for fact. The important point here is that we are changing the value of _i_ in the loop. If we don't then we guarantee we get an infinite loop! We also note that at some point we will get to the point where _i_ becomes less than or equal to zero so that the loop finishes. It might take sometime to execute if _i_ was very large but it will terminate at some point.

7.1.2 why use while statements?

As we noted above, we could most easily have written our factorial calculation above using a *for* loop. The key is that before we execute a *for* loop we know or can calculate the precise number of steps needed.

However, in many cases we may not know *a priori* the number of times we wish to execute the loop. Let's take for example the calculation of the sine of a number using the series expansion

$$\sin(x) \simeq x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

We might guess that for very small x we hardly need to go beyond the first term, whereas, for larger x we may need to include many more terms in the series. One way of deciding when to stop the series would be when the size of the last term added reaches a pre-determined and small value (the is not the best way but gives an idea of the principle). We will use this idea in the code fragment below. Look at and read the code below carefully. There are a large number of comments to help you.

In [ ]:

```
from math import *

def fact(n):                    # we need to calculate factorials so let's define a function a
    ans =1                      # if you look you might find a function to calculate factorial
    for i in range(n):
        ans=ans*(i+1)
    return (ans)

dangle = 10                     # enter angle in degrees
angle = dangle*2*pi/(360.)      # now convert to radians (again you might find function in pyt

tolerance = 0.0001              # set the limit for the size of new term below which will term
newterm = 1                     # this doesn't really matter as long as it is > tolerance (to
x = angle                       # we know the first term in the series is the angle in radians
i = 3                           # we have the first term in term in the series (i=1) so we'll
j=-1                            # our terms alternate in sign starting negative for i=3.
nterms = 1
while (abs(newterm) > tolerance):       # we have made sure the loop executes at least once.
    newterm = x**i/fact(i)*j            # calculate he next term NB. j is the sign. When new
    x = x+newterm                       # add the new term to our approximation
    i = i + 2                           # we don't have any even terms so increment i by 2
    j=j*-1                              # changes the sign for the next term
    nterms = nterms + 1                 # keeps a record of the number of terms we've inclua

print("sin(",dangle,") is approximately",x)
print(nterms," terms in the series were used")
print("sin(",dangle,") is ",sin(angle))
```

Note the programme always uses at least the first two terms in the series (the loop executes at least once). The value in the variable 'tolerance' controls when we will exit the loop. The smaller it is the more terms will be included. You might try changing its value (but whatever you do don't make it negative - why?). Also check to see as you make the angle larger that more terms are included.

I have included my own function for calculating factorials. See if you can find the python function in the math packages to do the same. This is not the only way in which you might do the calculation - feel free to experiment!

7.1.3 how do I code a 'repeat ... until' structure

In some languages and books on programming you may see reference to a *repeat ... until* loop. Python does not have a direct implementation. So what is a *repeat ... until* loop? As we noted our while loop will not execute at all if the boolean expression is false when the while statement is first called. A *repeat ... until* loop executes the loop at least once and tests at the end of the loop. As we have seen in the example above this can be achieved with a while loop by forcing the condition to be true at the entry point to the while loop.

An alternative way to do this is as follows

In [ ]:

```
i = 5
while True :
   i =i -1
   if i == 0 :
        print ("the condition i == 0 has been met")
        break

print("end")
```

As you can see the condition (i == 0) here is tested at the end of the loop if the condition is met then *break* means exit the loop immediately and proceed with the code following.

7.1.4 the *break* statement

break may actually be used at any position in a *while* or *for* loop. However, you should think very carefully about how you use it and what the implications are for the code following when you break from a partially completed loop.

7.2 Summary

In this notebook you have learnt how to:

- use a *while* statement to repeatedly execute a fragment of code until a boolean condition is met,
- anticipate and avoid infintite loops when using *while* statements,
- learn how to break out of for or *while* loop,
- shown how a *while* loop may be used to sum a sine series to a pre-set tolerance.