Lesson 5: Loops (Iteration, Repetition): `while` statement

So far, all our programs have carried out one major task such as converting a single quantity of metres to centimetres.

Frequently, we want to repeat such a calculation.

Say we have thirty values for metres which we want to convert to centimetres.

Using the program from a previous lesson, we would have to run it 30 times to achieve the desired result.

Programming languages provide loops to allow us repeat part of the program as many times as we wish.

For example, in the conversion program we can write the program to repeat the process of reading a value to be converted and displaying the result, 30 times or any number of times.

This is called looping (repetition or iteration).

The ***while*** loop

There are a number of loop statements, but all program looping can be performed using a **`while`** statement.

Loops are another form of conditional statement.

In the case of a loop, we use the condition to decide whether to repeat a statement or not.

```
while condition:
    action statement(s)          # loop body

rest of program statements
```

The **action statements of a loop** are called the **loop body -** can be a ***single*** or *group* of statements

The **loop body is executed** only if the **condition evaluates to true.** After executing the loop body, the condition is then re-evaluated to test if it is still true.

If it is true, we repeat execution of the loop body and test the condition again.

This process continues **until the condition evaluates to false**.

Example L5.1

Write a calculator program to sum pairs of numbers, until the user enters 0 as one of the numbers. We read in the two numbers to be summed, calculate the sum and display the result. We repeat these steps until the user enters 0.

We use a `while` loop to repeat the necessary statements:

```
# calc4.py: Repeat adding 2 numbers until user enters 0

n1 = 1        # Assign non-zero so that we can start the loop
n2 = 1

while (n1 != 0) and (n2 != 0) :

        n1 = float(input('\nEnter first number [0 to quit]: '))
        n2 = float(input('\nEnter second number [0 to quit]: '))
        sum = n1 + n2
        print(f'\nThe sum of {n1} and {n2} is {sum} \n\n')

print ('\n\nFinished summing\n')
```

The loop body is highlighted in blue. The loop body statements are repeated until the user enters 0 for one of the numbers.

When the loop condition evaluates to false, the loop terminates and the first statement after the loop body is executed – here it is a `print` to display that the program is finished.

Example L5.1 output

```
Enter first number:  4
Enter second number: 6
The sum of 4.0 and 6.0 is 10.0

Enter first number:  20
Enter second number: 30
The sum of 20.0 and 30.0 is 50.0

Enter first number:  0
Enter second number: 6
The sum of 0.0 and 6.0 is 6.0

Finished summing
```

Example L5.2

Modify the L5.1 to sum **three** pairs of numbers. Read in the two numbers to be summed, calculate the sum and display the result, **three** times.

We sometime call such a loop a **counting** loop.

```
# calc5.py: Calculator program to add 2 numbers, 3 times

count = 1

while count <= 3:
      n1 = float(input('\nEnter first number: '))
      n2 = float(input('\nEnter second number: '))

      sum = n1 + n2
      print(f'\nThe sum of {n1} and {n2} is {sum} \n\n')

      count = count + 1              # end of loop

print ('Finished summing\n')
```

Example L5.2 outputs:

```
Enter first number: 1
Enter second number: 2

The sum of 1.0 and 2.0 is 3.0

Enter first number: 3
Enter second number: 4

The sum of 3.0 and 4.0 is 7.0

Enter first number: 4
Enter second number: 5

The sum of 4.0 and 5.0 is 9.0

Finished summing
```

What would happen if we omitted the statement

```
count = count + 1
```

from the loop body?

This is a very common error to make with counting loops.

If we omit the statement to increment `count`, the **loop will never terminate**, **as `count` will always be less than 4**. It is an example of an **infinite** or **endless loop**.

An endless loop may be terminated by interrupting the program or switching off the computer, both of which terminate the program as. To interrupt a program, a combination of keys is pressed, such as pressing the control key and the C key simultaneously (denoted by Ctrl/C).

Such an error is a **logical or runtime error**. These differ from syntax errors because the program can be executed but does not behave as expected.

For this reason, they are more serious than syntax errors. In large programs, it is very difficult to ensure that there are no logical errors.

Thorough testing of programs may increase our confidence that a program is correct, but such **testing on its own, can never establish the correctness of a program**.

It is important to bear this fact in mind and it is worthwhile investigating the area of program correctness.

Example L5.3

Write a program to sum the integers 1 to 99 (i.e. calculate the sum of 1+2+3+...+99) and display the result.

```
# sum.py: calculate 1+2+3+.....+99

sum = 0       # contains the sum we wish to compute
i = 1         # the loop counter

while i <= 99:
    sum = sum + i
    i = i + 1

print(f'\nSum of 1 to 99 is: {sum}\n' )
```

Executing this program produces:

```
Sum of 1 to 99 is: 4950
```

The loop body is executed only if the condition `(i <= 99)` evaluates to true.

Since we have initialised `i` to 1, the condition is true and the loop body is executed.

In the loop body,

a running total for `sum` is calculated by adding the value of `i` to `sum`.
the variable `i` is then increased by 1.
the condition is tested again.

The variable `i` now has the value 2 and the condition `(i <= 99)` remains true so we execute the loop body again assigning `sum` the value 3 (1+2) and increasing `i` to 3.

Next time around the loop, `sum` becomes 6 (3+3) and `i` becomes 4.

We test the condition again and continue in this manner until `i` eventually reaches the value 100.

When we test the condition in this case, it is now false (i.e. `i > 99`) and so the loop body is not executed.

We now continue at the first statement after the loop body i.e. the `print` statement.

Example L5.4
Sometimes it is useful to put a `print` in the loop body so you can see what's happening

```
# L5.4 sum2.py: calculate the sum of 1 to 9

sum = 0
i = 1

while i <= 9:
    sum = sum + i
    print(f'Sum = {sum} i = {i}) # display what's happening
    i = i + 1

print(f'\nSum of 1 to 9 is: {sum}\n' )
```

Executing this program produces as output:

```
Sum =  1  i =  1
Sum =  3  i =  2
Sum =  6  i =  3
Sum =  10  i =  4
Sum =  15  i =  5
Sum =  21  i =  6
Sum =  28  i =  7
Sum =  36  i =  8
Sum =  45  i =  9

Sum of 1 to 9 is: 45
```

The `break` statement

Sometimes we wish to terminate a `while` loop without having to wait for the loop condition to become false. We use the **`break`** statement to do this. It stops the loop and the program continues at the first statement after the loop body.

Example L5.7
A guessing game program. The user guesses a "secret" word built into the program.

```
# guess.py: Guess the secret word

secret = 'blue'
guess = ''

while (guess != secret) and (guess != 'quit'):

   guess = input('Guess the word:[quit to finish]  ')

   if guess == 'quit':
       break                          # Exit the loop

   if guess != secret:
       print(f'\nWrong guess: {guess}')

   else:
       print(f'\nWell done !')     # end of loop

if (guess == 'quit'):
                print(f'\nThe secret word was: {secret}')
```

If the user enters `'quit'` then the **`break`** statement terminates the loop and the first statement after the loop body is executed – displaying the secret word.

Note: The loop in this program can terminate in **two** ways. It will terminate if the loop condition is false (for example the user guesses the word) OR if the user enters `'quit'`.

This means that when the loop terminates, we need to check if it was because the user entered `'quit '` and display the appropriate message in that case.

```
Running guess3.py:

Guess the secret word:  man
Wrong guess:  man
Guess the secret word:  dog
Wrong guess:  dog
Guess the secret word:  quit
The secret word was:  blue

Running guess3.py:

Guess the secret word:  black
Wrong guess:  black
Guess the secret word:  blue
Well done !
```

Example L5.8
Guessing game program to limit number of guesses. The user has **only 3 chances** to guess the "secret" word.

Algorithm for this guessing game program

We explained the concept of an algorithm earlier. It is the set of set of steps to solve a problem. We usually write algorithms in what is called pseudo code.

This is a cross between English and programming language statements. There is no defined version of pseudo code, so you can make up your own version.

In my pseudo code, I use **`repeat until`** … **`end repeat`** for a loop.

It can be read as "repeat the statements from `repeat` to `end repeat` while the condition is true.

In the example below the loop body is highlighted in blue.

Because we are now using conditionals (`if` and `while`) our programs are becoming longer and more complex.

So it is a good idea to develop an algorithm for your program before writing the actual code.

```
# Guessing game algorithm

Set number of guesses to 1
Set guess to blank
Set the secret word in the program

Repeat until guess is correct, or quit or number of guesses > 3

        Ask the user to guess the word or quit
        If guess is 'quit'
           Exit the loop
        If guess is incorrect then
           Display error message
           Add 1 to number of guesses
        Else
           Display Correct guess message

End repeat

If guess is quit
      Display quit message
else
      Display too many guesses message

Program terminates
```

We implement the algorithm in Python:

```
# L5.8: guess3.py: Guess the secret word in 3 guesses

secret = 'blue'
guess = ''
num_g = 1   # number of guesses

while (guess != secret) and (guess != 'quit') and (num_g < 4):

    guess = input('Guess the secret word:[quit to finish]  ')

    if guess == 'quit':
        break               # Exit the loop

    if guess != secret:
        print(f'\nWrong guess: {guess}')
        num_g = num_g + 1

    else:
        print(f'\nWell done !') # end of loop

if (guess == 'quit'):
    print(f'\nThe secret word was: {secret}')

else:
    print(f'\n Sorry you have used 3 guesses')
    print(f'\n\nThe secret word was: {secret}')
```

Note: **We insert blank lines in our code to make it easier to read**.

```
Running guess3.py:

Guess the secret word:  cat
Wrong guess:  cat
Guess the secret word:  dogs
Wrong guess:  dog
Guess the secret word:  red
Sorry you have used 3 guesses')

The secret word was: blue

Running guess3.py:
Guess the secret word:  black
Wrong guess:  black
Guess the secret word:  blue
Well done !
```

Nested Loops

A loop may contain as part of its loop body any statement including another loop.

A loop inside the body of another loop is called an **inner loop** or **nested loop.**

Example 5.9: Write a program to read in the marks for a group students and display the average mark for each student. There are 3 marks for each student. The programs allows the user enter as many students as they wish, finishing when the name '***quit***' is entered.

Algorithm

```
Read name

Repeat until name is quit                # outer loop

    Set sum to 0
    Set number of marks to 1

    Repeat until number of marks > 3 # nested loop
        Read mark
        Add mark to sum
        Add 1 to number of marks
    End repeat                           # end of nested loop

    Compute average  = sum / 3
    Display average mark for name
    Read next name

End repeat                               # end of outer loop

Display finished message
```

```
# L5.9 average.py: Compute average mark for students
# There are 3 marks for each student

name = input('\nEnter name: [or quit] :')

while ( name != 'quit' ):

    nm = 1          # number of marks entered
    sum = 0.0

    while ( nm <= 3 ):
        mark = float(input(f'Enter mark {nm}: '))
        sum = sum + mark
        nm = nm + 1          #end of inner loop

    average = sum / 3

    print(f'Average mark for {name} : {average:.2f}' )

    name = input('\nEnter name: [quit] : ') # end of outer loop


print(f'\nFinished \n')
```

L5.9 runs as follows

```
Enter name: [quit] : Joe
Enter mark 1: 50
Enter mark 2: 60
Enter mark 3: 70
Average mark for Joe : 60.00

Enter name: [quit] : Mary
Enter mark 1: 70
Enter mark 2: 80
Enter mark 3: 85
Average mark for Mary : 78.33

Enter name: [quit] : quit

Finished
```

Note the use of an **f-string** in the `input` statement:

```
mark = float(input(f'Enter mark {nm}: '))
```

This allows us display which of the three marks is being entered (1, 2, or 3) as shown in the output above.

Time to practice !

- Copy all the examples from the slides above and get them to run in your Python environment.
- Then complete the exercises from the Handbook and get them to run.
- Finally carry out the assignments from the Handbook and get them to run.