

# Conditional statements

Monday, December 13, 2021 6:37 PM

## IF STATEMENTS

Statement are a group of conditions in Python which can be used in several different ways. The most commonly used is the if statement and is the first most people will learn.

See the following example:

```
In [29]: a = int(33)
         b = int(200)
         if b > a:
             print("b is greater than a")

Out[30]: b is greater than a
```

Note that there must be an indentation, otherwise a syntax error will be raised.

`elif`, meaning "else if" is Python's way of saying that "if the previous conditions were not true then try this next condition".

See the following example which shows that because the first condition is false it will try the second condition in the `elif` clause which is true.

```
In [31]: a = int(33)
         b = int(33)
         if b > a:
             print("b is greater than a")
         elif a == b:
             print("a is equal to b")

Out[32]: a is equal to b
```

Else conditions exist to catch anything that wasn't caught by the previous conditions in the statement. In the following example a is greater than b so condition 1 is false, the second condition is also false. The else condition however is true which completes the statement:

```
In [33]: a = int(200)
         b = int(33)
         if b > a:
             print("b is greater than a")
         elif a == b:
             print("a is equal to b")
         else:
             print("a is greater than b")

Out[34]: a is greater than b
```

Note, it is possible to have an else without an `elif` condition.

For statements in Python differ to what they are in other high level programming languages.

# FOR LOOPS

Looping something over, a for loop is called an iterable in Python.

Rather than iterating over a progression of numbers or giving the user the ability to define the iteration step and the halting condition, for statements in Python iterate over the items of any sequence in order that they appear in the sequence.

See the following example which shows that the for loop lacks the need for an indexing variable to be set beforehand:

```
In [35]: words = ["cat", "dog", "windows"]
         for w in words:
             print(w)
```

```
Out[36]: cat
         dog
         windows
```

Break statements are used to break out of the innermost enclosing for loop before it has looped through all the items. The following example shows a break statement which exists the loop when w is "dog", it's important to note that the break can come before the `print()` function as well:

```
In [37]: words = ["cat", "dog", "windows"]
         for w in words:
             if w == "dog":
                 break
             print(w)
```

```
Out[38]: cat
```

The continue statement can stop the current iteration of the loop and continue with the next, the following example shows a continue statement where it does not print "dog":

```
In [39]: words = ["cat", "dog", "windows"]
         for w in words:
             if w == "dog":
                 continue
             print(w)
```

```
Out[40]: cat
         windows
```

```
In [1]: for byte in b'Binary':
         print(byte)
```

```
Out[2]: 66
         105
         110
         97
         114
         121
```

With for loops, it is possible to loop over the contents one byte at a time. In this above example, the byte values represent the ASCII codes for each letter.

It is important to note that integers are not iterable objects and thus cannot be iterated in a for loop.

This however can be done by constructing string versions of the integer.

## WHILE LOOPS

While loops are used to repeat a block of code, which allows the creation of interesting programs. Unlike if conditions that check only once, while loops check the condition repeatedly until the condition is false. If there is no condition for false then a while loop will continue until system memory is depleted, an "infinite loop".

While loops start with the word while, followed by the condition name then a colon. The next line sees the body of the loop which follows indentation.

In the following example, a while condition is evaluated as a Boolean expression that if true, the body of the loop is executed. To prevent a loop from running endlessly a condition must be placed that stops the loop after running x amount of times.

```
In [9]: count = 0

while count < 5:
    print("I am inside a loop")
    count = count + 1
```

```
Out[10]: I am inside a loop
I am inside a loop
I am inside a loop
I am inside a loop
I am inside a loop
```

Because counts initial value is 0, the loop is run 5 times due to count being increased by +1 in each iteration of the loop.

While loops can also take user input, in the next example this while loop prints the multiplication table of a number taken by user input from 1 to 10.

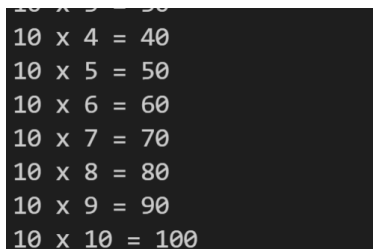
This is done by creating a loop that runs 10 times. In the body of the loop, a condition that multiplies the number by the count which then can be displayed by using a `print()` method.

```
number = int(input("Please enter a number: "))

count = 1
while count <= 10:
    product = number * count
    print(number, "x", count, "=", product)
    count = count + 1
```

The times tables can be printed in a nice format by first printing the number inputted by the user followed by a "x" string, the count, a "=" followed by the calculated product. The example shows the output.

```
Please enter a number: 10
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
```



10 x 4 = 40  
10 x 5 = 50  
10 x 6 = 60  
10 x 7 = 70  
10 x 8 = 80  
10 x 9 = 90  
10 x 10 = 100