Python For Loops

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

In [1]:
```
# Example
# Print each fruit in a fruit list:

fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

```
apple
banana
cherry
```

The for loop does not require an indexing variable to set beforehand.

Looping Through a String

Even strings are iterable objects, they contain a sequence of characters:

In [2]:
```
# Example
# Loop through the letters in the word "banana":

for x in "banana":
    print(x)
```

```
b
a
n
a
n
a
```

The break Statement

With the break statement we can stop the loop before it has looped through all the items:

In [3]:
```
# Example
# Exit the loop when x is "banana":

fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
    if x == "banana":
        break
```

```
apple
banana
```

In [5]:
```
# Example
# Exit the loop when x is "banana", but this time the break comes
# before the print:

fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        break
    print(x)
```

```
apple
```

The continue Statement

With the continue statement we can stop the current iteration of the loop, and continue with the next:

In [7]:
```
# Example
# Do not print banana:

fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
```

```
apple
cherry
```

The range() Function

To loop through a set of code a specified number of times, we can use the range() function, The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

In [8]:
```
# Example
# Using the range() function:

for x in range(6):
    print(x)
```

```
0
1
2
3
4
5
```

Note that range(6) is not the values of 0 to 6, but the values 0 to 5.

The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6):

In [9]:
```
# Example
# Using the start parameter:

for x in range(2, 6):
    print(x)
```

```
2
3
4
5
```

The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3):

In [10]:
```
# Example
# Increment the sequence with 3 (default is 1):

for x in range(2, 30, 3):
    print(x)
```

```
2
5
8
11
14
17
20
23
26
29
```

Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:

In [11]:
```
# Example
# Print all numbers from 0 to 5, and print a message when the loop has ended:

for x in range(6):
    print(x)
else:
    print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```

Note: The else block will NOT be executed if the loop is stopped by a break statement.

In [12]:
```
# Example
# Break the loop when x is 3, and see what happens with the else block:

for x in range(6):
    if x == 3: break
    print(x)
else:
    print("Finally finished!")
```

```
0
1
2
```

Nested Loops

A nested loop is a loop inside a loop.

The "inner loop" will be executed one time for each iteration of the "outer loop":

In [13]:
```
# Example
# Print each adjective for every fruit:

adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
    for y in fruits:
        print(x, y)
```

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
```

The pass Statement

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

In [17]:
```
# Example
for x in [0, 1, 2]:
    pass
```

In [ ]:
```

```