Python While Loops
Python Loops
Python has two primitive loop commands:

while loops
for loops
The while Loop
With the while loop we can execute a set of statements as long as a condition is true.

ExampleGet your own Python Server
Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```
Note: remember to increment i, or else the loop will continue forever.

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

The break Statement
With the break statement we can stop the loop even if the while condition is true:

ExampleGet your own Python Server
Exit the loop when i is 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```
ADVERTISEMENT

The continue Statement
With the continue statement we can stop the current iteration, and continue with the next:

ExampleGet your own Python Server
Continue to the next iteration if i is 3:

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

The else Statement
With the else statement we can run a block of code once when the condition no longer is true:

ExampleGet your own Python Server
Print a message once the condition is false:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

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.

ExampleGet your own Python Server
Print each fruit in a fruit list:

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

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:

ExampleGet your own Python Server
Loop through the letters in the word "banana":

```
for x in "banana":
  print(x)
```
The break Statement
With the break statement we can stop the loop before it has looped through all the items:

ExampleGet your own Python Server
Exit the loop when x is "banana":

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```
ExampleGet your own Python Server
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)
```
ADVERTISEMENT

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

ExampleGet your own Python Server
Do not print banana:

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

ExampleGet your own Python Server
Using the range() function:

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

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):

ExampleGet your own Python Server
Using the start parameter:

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

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):

ExampleGet your own Python Server
Increment the sequence with 3 (default is 1):

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

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

ExampleGet your own Python Server
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!")
```

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

ExampleGet your own Python Server
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!")
```

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":

ExampleGet your own Python Server
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)
```

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.

ExampleGet your own Python Server

```
for x in [0, 1, 2]:
  pass
```