Chapter 6: Python Loops

Python has two primitive loop commands: `while` loops and `for` loops

while Loop

With the `while` loop we can execute a set of statements as long as a condition is true.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Note: remember to increment i, or else the loop will continue forever.

break Statement

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

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

continue Statement

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

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

else Statement

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

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

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).

```
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:

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

break Statement

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

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

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)
```

continue Statement

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

Do not print banana:

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

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.

```
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 (not including):

```
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:

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:

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.

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":

```
colors = ["green", "yellow", "red"]
fruits = ["apple", "banana", "cherry"]

for clr in colors:
  for frt in fruits:
    print(clr, frt)
```