The for Loop

This lesson highlights the main features of the 'for' loop.

We'll cover the following

A `for` loop uses an **iterator** to traverse a sequence, e.g. a range of numbers, the elements of a list, etc. In simple terms, the iterator is a variable that goes through the list.

The iterator starts from the beginning of the sequence. In each iteration, the iterator updates to the next value in the sequence.

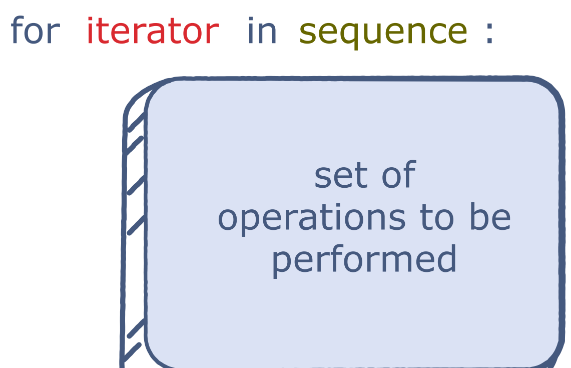
The loop ends when the iterator reaches the end.

Structure

In a `for` loop, we need to define three main things:

1. The name of the iterator
2. The sequence to be traversed
3. The set of operations to perform

The loop always begins with the `for` keyword. The body of the loop is indented to the right:



The `in` keyword specifies that the iterator will go through the values *in* the sequence/data structure.

Looping Through a Range

In Python, the built-in `range()` function can be used to create a sequence of integers. This sequence can be iterated over through a loop. A range is specified in the following format:

```
range(start, end, step)
```

The `end` value is not included in the list.

If the `start` index is not specified, its default value is `0`.

The `step` decides the number of steps the iterator jumps ahead after each iteration. It is optional and if we don't specify it, the default `step` is `1`, which means that the iterator will move forward by one step after each iteration.

Let's take a look at how a `for` loop iterates through a range of integers:

```
for i in range(1, 11):  # A sequence from 1 to 10
    if i % 2 == 0:
        print(i, " is even")
    else:
        print(i, " is odd")
```

As we can see above, rather than individually checking whether every integer from `1` to `10` is even or odd, we can loop through the sequence and compute `i % 2 == 0` for each element.

The iterator, `i`, begins from `1` and becomes every succeeding value in the sequence.

Let's see how a loop changes when the `step` component of a range is specified:

```
for i in range(1, 11, 3):  # A sequence from 1 to 10 with a step of 3
    print(i)
```

Looping Through a List/String

A list or string can be iterated through its indices.

Let’s double each value in a list using a `for` loop:

```
float_list = [2.5, 16.42, 10.77, 8.3, 34.21]
print(float_list)

for i in range(0, len(float_list)
               ):  # Iterator goes traverses to the last index of the list
    float_list[i] = float_list[i] * 2

print(float_list)
```

We could also traverse the elements of a list/string directly through the iterator. In the `float_list` above, let’s check how many elements are greater than `10`:

```
float_list = [2.5, 16.42, 10.77, 8.3, 34.21]
count_greater = 0

for num in float_list:  # Iterator goes traverses to the last index of the list
    if num > 10:
        count_greater += 1

print(count_greater)
```

In this example, `num` is the iterator. An important thing to keep in mind is that in the case above, altering `num` will **not alter the actual value in the list**. The iterator makes a copy of the list element.

In the next lesson, we’ll learn how to nest `for` loops.