Loops

This lesson will introduce loops in Python.

WE'LL COVER THE FOLLOWING

Why loops?

Many times, during programming, we have to repeat the same task a number of times. For instance, we have a list of numbers and we want to double every number in it. Here, we want to perform the same task for every item on our list. This is where loops come in handy. **Loops** are used to execute the same block of code a fixed number of times.

There are two kinds of loops in Python.

- The `for` loop
- The `while` loop

The `for` loop

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.

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

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

- The name of the iterator
- The sequence to be traversed
- The set of operations to perform

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

```
for iterator in sequence:
   body
```

The `in` keyword specifies that the iterator will go through the values in the sequence/list.

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,stop,step)
```

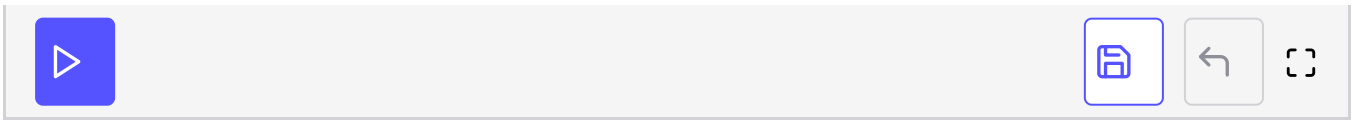
If the `start` index is not specified, its default value is `0`.

The `stop` value determines the end of the list and is not included in the list. It also tells where the iterator should stop.

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 loop that iterates over a list
for i in range(1,11):
  sq = i**2
  print('The square of',i, '=',sq)
```



We use the `for` loop structure we defined above in the first line. `range(1,11)`

gives us a list of integers `[1,2,3,4,5,6,7,8,9,10]`. In the first iteration, `i` gets assigned the value `1`. The execution moves to **line 3**. Here a variable `sq` is created and assigned the value of `i` squared. Then in **line 4**, the results are printed. After **line 4**, the iterator `i` is assigned the next value from the list which is `2`. **Lines 3 and 4** repeat. Then `i` is updated to `3` and so on. The loop stops when all the elements of the list have been assigned to `i` turn by turn.

The `break` statement

Sometimes, we need to exit a loop before it reaches the end. This can happen if we have found what we were looking for and don't need to make any more computations in the loop.

A good example would be if we are searching in a list for an item. We will exit the loop once we have found it.

```
names = ['Monica','Ross','Chandler','Joey','Pheobe','Rachel']
found = False
for i in names:
  if i == 'Joey':
    print('Joey does not share food!' )
    found = True
    break

if found == False:
  print('Joey not found')
```

In **line 1**, we create a list, `names`, with 6 names in it. We want to search the list and print a message when we find `Joey` in the list. We will also print a message if we do not find it. Therefore, we create a Boolean variable `found` and set it to `False`. We will set it to `True` when we find `Joey`.

In **line 3**, we write the `for` loop. Iterator `i` will be assigned every name in the list one by one. When the condition on **line 4** is satisfied, a message is printed in **line 5**. Then `found` is assigned `True` in **line 6**. In **line 7**, we exit the loop because of the `break` statement. In **line 9**, we check if `Joey` was not found and we print a message.

The `while` loop

The `while` loop keeps iterating over a block of code as long as a certain **condition** holds `True`. It operates using the following logic:

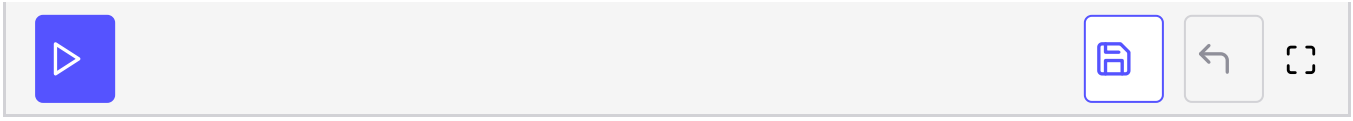
While this condition is true, keep the loop running.

In a `for` loop, the number of iterations is fixed since we know the size of the sequence. On the other hand, a `while` loop is not always restricted to a fixed range. Its execution is based solely on the condition associated with it.

```
while condition:
  body
```

Below we write code for printing the squares of first 10 integers.

```
i = 1
# while loop
while i < 11:
  sq = i**2
  print('The square of',i, '=',sq)
  i = i+1
```



In the first line, we create variable `i` and assign it a value of `1`. In **line 3**, we write the keyword `while`, and specify our condition. Our condition is `i < 11`. As long as this condition is `True`, the code inside the loop will keep executing. Inside the loop, we just take the square of `i` in **line 4** and print it in **line 5**. In **line 6**, we update the value of `i`. If we do not write **line 6**, the `while` loop will keep running indefinitely.

We can also use the `break` statement with a `while` loop and exit it when a certain condition is fulfilled.

We are now familiar with the very basic features of the language. In the next lesson, we will look at *libraries* and *packages* in Python.