Iteration & Loops

In this chapter, we are going to explore iteration and loops in Python.

WE'LL COVER THE FOLLOWING

Loops are used in computer programming to automate repetitive tasks.

Types of Loops in Python

In python, there are two types of loops.

For Loop

In Python, the most common form of iteration is the “for” loop.

A `for` loop is used for iterating over a sequence of items.

The general syntax for creating a `for` loop is:

```
for index in sequence:
    statement
```

Here, a sequence can be a list, tuple, dictionary, set, or string.

> **Note:** The for loop does not require an iterating/ indexing variable to set

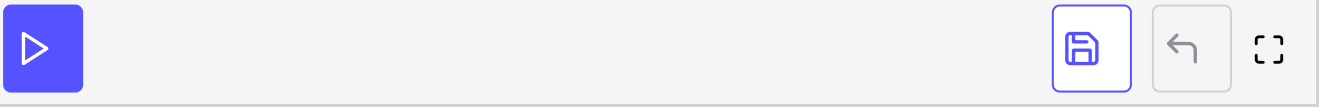
beforehand. The indexing can be defined in a loop.

Looping Through a List

The “for” loop allows you to iterate over all items of a list, such that you can do whatever you want with each item.

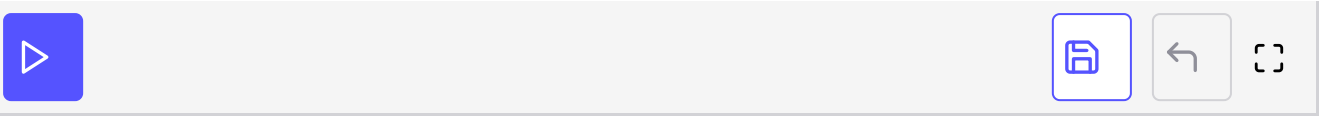
For instance, let’s create a list and print the square value of each element.

```
for value in [0, 1, 2, 3, 4, 5]:
  print(value * value)
```



It’s quite easy but very powerful! The “for” loop is the basis of many things in programming. For instance, you already know about the “sum(list)” function which sums all the elements of a list, but here’s an example using the “for” loop:

```
mylist = [1,5,7]
sum = 0
for value in mylist:
  sum = sum + value
print(sum)
```

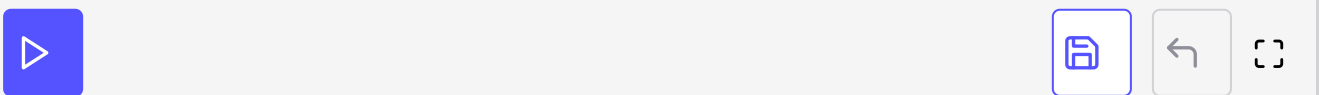


Basically, you create the variable “sum” and keep adding each value as it comes from the list.

Looping Using the `range`

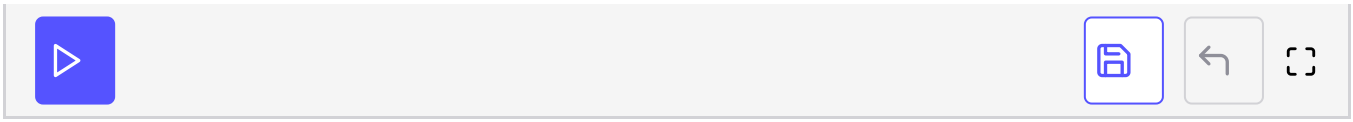
The `range()` function allows looping through a set of code a specified number of times. The range(n) function returns a sequence of numbers, starting from 0 by default and incrementing by 1 (by default), and ends at a specified number(n) minus 1.

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



Sometimes, instead of the values of a list, you may need to work with the indexes; i.e., not with the values, but the positions of the values in the list. Here's an example that iterates over a list and returns the indexes and the values for each index:

```
mylist = [1,5,7]
for i in range(len(mylist)):
  print ("Index:", i, "Value:", mylist[i])
```



> **Note:** You can see that we are not iterating over the list itself, but iterating over the "range" of the length of the list. The range function returns a special list.

So, when you use "range" you are not iterating over "mylist"; you're iterating but over a list with some numbers that you'll use as indexes to access individual values on "mylist".

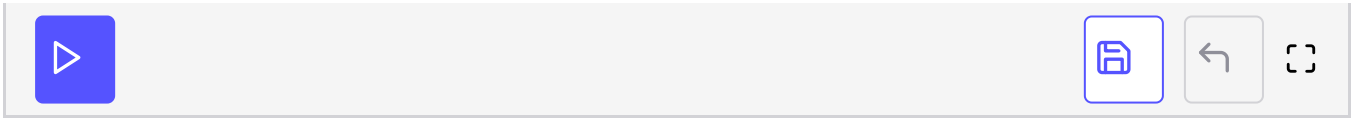
There's more about the range function in the Python documentation.

Looping Using `enumerate`

Sometimes you may need both things (indexes and values), and you can use the `"enumerate"` function:

```
mylist = [1, 5, 7]

for i, value in enumerate(mylist):
  print "Index:", i, "Value:", value
```

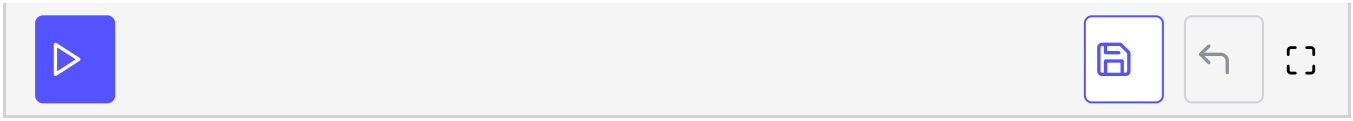


> **Note:** The first value on a Python list is always at index 0, while it is greater than 0.

Looping Through a String

Strings contain a sequence of characters so that they can be iterated.

```
for x in "Full Speed Python":
  print(x)
```



While Loop

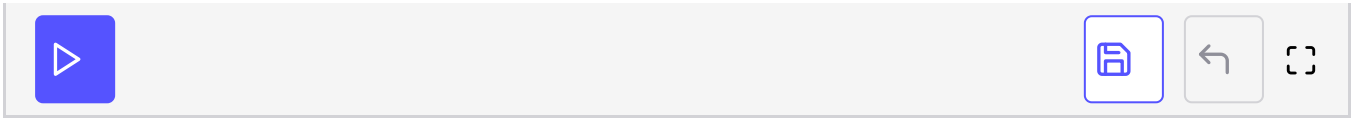
Finally, we also have the `"while"` statement that allows us to repeat a sequence of instructions while a specified condition is true.

The general syntax for creating a while loop is:

```
while(condition):
    statement
```

For instance, the following example starts “n” at 10 and **while “n” is greater than 0**, it keeps subtracting 1 from “n”. When “n” reaches 0, the condition “n > 0” is false, and the loop ends:

```
n = 10
while n > 0:
  print(n)
  n = n-1
```



Notice that it never prints 0.

Now that the idea of ’Iteration and Loops’ in python is clear, let’s check your knowledge in the upcoming exercises before moving on to ‘Dictionaries’ chapter.