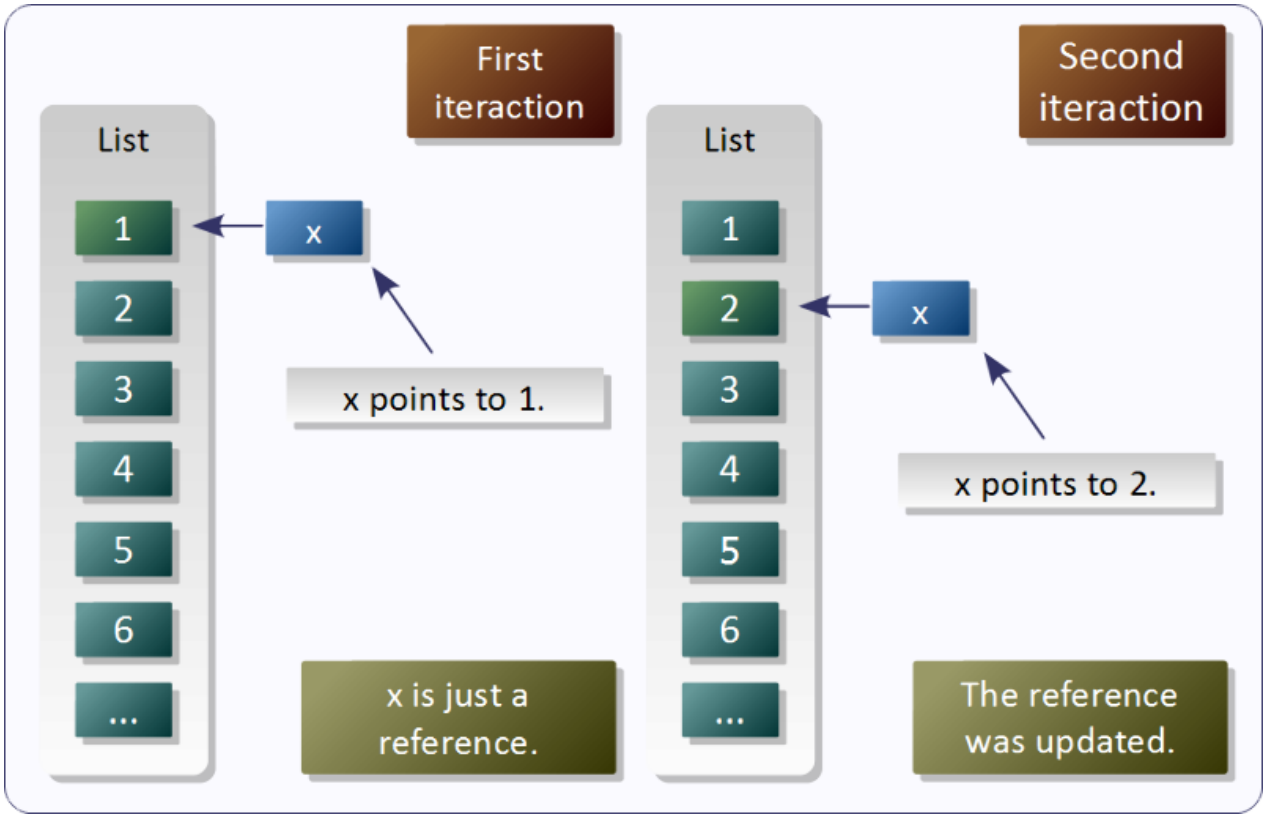
Chapter 4: Loops

Loops are repetition structures, generally used to process data collections, such as lines of a file or records of a database that must be processed by the same code block.

For

It is the repetition structure most often used in Python. The statement accepts not only static sequences, but also sequences generated by iterators. Iterators are structures that allow iterations, i.e. access to items of a collection of elements, sequentially.



During the execution of a *for* loop, the reference points to an element in the sequence. At each iteration, the reference is updated, in order for the *for* code block to process the corresponding element.

The clause *break* stops the loop and *continue* passes it to the next iteration. The code inside the `else` is executed at the end of the loop, except if the loop has been interrupted by *break*.

_*Syntax_8:

```
for <reference> in <sequence>:
    <code block>
    continue
    break
else:
    <code block>
```

Example:

```
# Sum 0 to 99
s = 0
for x in range(100, 1, -5):
    print(x)
    s = s + x
print("sum of 0 to 99 is", s)
print("-" * 20)
for x in []:
    print("Hello")
else:
    print("Sorry")
```

```
  File "<ipython-input-6-2255673f0bfc>", line 3
    for x in range 100, 1, -5:
                   ^
SyntaxError: invalid syntax
```

```
cols = ["Red", "Green", "Yellow", "White"]
for color in cols:
    print(color)
```

```
Red
Green
Yellow
White
```

```
color = {"c1": "Red", "c2": "Green", "c3": "Orange"}
for value in color.values():
    print(value)
print("-"*10)
for col in color:
    print(col, color[col])
```

```
Green
Orange
Red
----------
c2 Green
c3 Orange
c1 Red
```

```
color = {"c1": "Red", "c2": "Green", "c3": "Orange"}
x_test = [[1,2],[3,4],[5,6]]

for value in  color.values():
    print( value)


for col in color:
    print(col, color[col])

for x in x_test:
    print(x)

for x, y in x_test:
    print(x, y)
```

```
Green
Orange
Red
c2 Green
c3 Orange
c1 Red
[1, 2]
[3, 4]
[5, 6]
1 2
3 4
5 6
```

The function `range(m, n, p)`, is very useful in loops, as it returns a list of integers starting at `m` through smaller than `n` in steps of length `p`, which can be used as the order for the loop.

While

Executes a block of code in response to a condition.

Syntax:

```
while <condition>:
    <code block>
    continue
    break
else:
    <code block>
```

The code block inside the *while* loop is repeated while the loop condition is evaluated as true.

Example:

```
# Sum 0 to 99
s = 0
x = 1

while x < 100:
    s = s + x
    x = x + 1
print ("Sum of 0 to 99", s)

while x < 0:
    print("Hello")
else:
    print("Sorry")
```

```
Sum of 0 to 99 4950
Sorry
```

The *while* loop is appropriate when there is no way to determine how many iterations will occur and there is a sequence to follow.

```
s = 0
x = 100

while x < 100:
    s = s + x
    x = x + 1
else:
    print("x is already equal or greater than 100")
print(s)
```

```
x is already equal or greater than 100
0
```

```
x = 1;
s = 0
while (x < 10):
     s = s + x
     x = x + 1
     if (x == 5):
          break
else:
     print('The sum of first 9 integers : ',s)
print('The sum of ',x,' numbers is :',s)
```

```
The sum of  5  numbers is : 10
```

Break

The break statement is used to exit a for or a while loop. The purpose of this statement is to end the execution of the loop (for or while) immediately and the program control goes to the statement after the last statement of the loop. If there is an optional else statement in while or for loop it skips the optional clause also

```
num_sum = 0
count = 0
for x in range(1, 9):
    print(x)
    num_sum = num_sum + x
    count = count + 1
    if count == 5:
        break
print("Sum of first ",count,"integers is : ", num_sum)
```

```
1
2
3
4
5
Sum of first  5 integers is :  15
```

Continue Statement

The continue statement is used in a while or for loop to take the control to the top of the loop without executing the rest statements inside the loop. Here is a simple example.

```
for x in range(8):
    if (x == 3 or x==6):
        print("\tSkipping:", x)
        continue
        print("This should never print")
    print(x)
```

```
0
1
2
    Skipping: 3
4
5
    Skipping: 6
7
```

Excersise

- Print all the characters in sentance "The continue statement is used in a while or for loop"
-