

Session 2

2. Control Flow Structures

2.2. Loops

Introduction to Loops

Sometimes, we need to repeat a set of instructions several times in the same way.

Python provides two main looping structures: **for** and **while**.

While Loops

Syntax :

```
1 while CONDITION:  
2     #instructions
```

All instructions inside the while loop are executed repeatedly until the condition becomes False.

Example :

```
1 n = 0  
2 while n < 5:  
3     print(n)  
4     n = n + 1  
5 print('End')
```

Note : Each time the block of instructions inside the loop runs, it's called an iteration.

While Loops

If the stopping condition never becomes False, the loop will run forever.

```
1 x = 0
2 while x >= 0:
3     print(x)
4     x = x + 1
5 print("End")
```

--> This creates an infinite loop.

While Loops

- **break**: stops the loop completely

```
1 # Look for N in [0, 100] where N ** 2 = 36
2 x = 0
3 while x < 100:
4     print("x =", x)
5     if x**2 == 36:
6         print("N is:", x)
7         break
8     x = x + 1
```

While Loops

- **continue** : skips the current iteration and goes to the next one.

```
1 # Print the square of every even number below 16
2 n = 0
3 while n < 16:
4     if n % 2 == 1:
5         n = n + 1
6         continue
7     print(n ** 2)
8     n = n + 1
```

Real Example: Simple Chatbot

```
1 # Simple Chatbot
2 print("Hello, I'm your Chatbot!")
3 user_query = input("What do you want: ")
4
5 while True:
6     if user_query == "END":
7         break
8     if user_query == "":
9         user_query = input("What's next: ")
10    continue
11    print("Okay, I will do:", user_query)
12    user_query = input("What's next: ")
13
14 print("Bye!")
```

For loops

In the previous example, we didn't know when the user would stop, so a while loop was suitable.

But when we know in advance how many iterations are needed (for example, summing numbers from 1 to n), a for loop is more convenient and easier to use.

Syntax:

```
1 for i in range(start, end, step):  
2     # instructions
```

For loops

Example 1:

```
1 for i in range(0, 5, 1):
2     print(i)
3
4 # Default start=0 and step=1
5 for i in range(5):
6     print(i)
```

For loops

Example 2: Let's calculate the term of a sequence:

$$U_n = \sum_{k=1}^n k^2$$

```
1 n = 20
2 result = 0
3
4 for k in range(1, n + 1):
5     result = result + k ** 2
6
7 print("U(" + str(n) + ") = " + str(result))
```

3. Lists

3. Lists

3.1. What is a List in Python?

What is a List in Python?

A list is a collection (or sequence) of elements.

```
1 clubs = ['AppsClub', 'CRRT', 'CreArt']
```

In python:

1. A list can contain elements of **different** data types.
2. A list can contain **nested lists** (lists inside lists).
3. Lists are **dynamic**, meaning we don't need to resize them when adding new items.

3. Lists

3.2. List Manipulation

List manipulation

```
1 names = ["Mohamed", "Ahmed", "Ali", "Khalid"]
2
3 # Accessing Elements
4 print(names[0])    # First element (index starts at 0)
5 print(names[-1])   # Last element
6
7 # Modifying a List Element
8 names[2] = "Lahoucine"
9
10 # Checking if an Element Exists
11 check = "Mohamed" in names
12 print(check) # True
13
```

List manipulation

Traversing a List

- Method 1 (read-only):

```
1 for name in names:  
2     print(name)
```

- Method 2 (can modify elements):

```
1 for i in range(len(names)):  
2     print(names[i])
```

List manipulation

List Slicing: Sometimes we need just a part of the list — we can easily copy it using slicing.

Syntax

```
1 list[start:end:step]
```

Examples:

```
1 numbers = [9, 1, 5, 3, 7]
2 print(numbers[0::2])  # [9, 5, 7]
3 print(numbers[1:3])  # [1, 5]
```

3. Lists

3.3. List Methods

List methods

```
1 numbers = [9, 1, 2, 7, 6, 4]
2
3 # Add one element
4 numbers.append(10)
5
6 # Add multiple elements
7 numbers.extend([5, 7, 2])
8
9 # Remove the first occurrence of 9
10 numbers.remove(9)
11
12 # Remove the element at index 0
13 numbers.pop(0)
14
15 # Sort the list
16 numbers.sort()
```

4. String

4. String

4.1. String vs List

String vs List

A **string** is a sequence of characters.

```
1 fruit = "banana"
```

Strings are **not** lists, but they share many features:

```
1 letter = fruit[1]
2 print(letter)

3
4 print('b' in fruit)

5
6 sliced_word = fruit[0:3]

7
8 for char in fruit:
9     print(char)
```

String vs List

Strings are **immutable**, which means you can't change an existing string :

```
1 fruit = "banana"  
2 fruit[0] = 'v' #Error
```

The best you can do is create a **new string** that is a variation on the original:

```
1 fruit = "banana"  
2 new_fruit = 'v' + fruit[1:]  
3 print(new_fruit)
```

4. String

4.2. String Methods

String methods

```
1 fruit = "banana"
2
3 index = fruit.find('b')          # Returns index of 'b' which is 0
4
5 characters = list(fruit)        # ['b', 'a', 'n', 'a', 'n', 'a']
6
7 full_name = "Mohamed Ali Mahmoud"
8 names = full_name.split(" ")   # ['Mohamed', 'Ali', 'Mahmoud']
9
10 words = ['look', 'for', 'a', 'job']
11 sentence = " ".join(words)    # 'look for a job'
```