Lists

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

List Definition and Usage

Definition

- A **list** is a **collection** that is **index supported** and **changeable (mutable)**
- It allows duplicate members
- In Python lists are written with square brackets

```
list_example = ["apple", "banana", "cherry"]
```

List Element

Usage in Programming

- Lists are very useful for storing **multiple elements**
- They can **expand** and **shrink**
- In Python a single list can store **elements** with **different data types**
- Lists are the **basis** for the other abstract data types like **queues**, **stacks**, and their variations

Storing Data

Data in Python Lists

- In Python, a **list** can store data of any data type like:
  - integers
  - floats
  - strings
  - objects
  - other lists
  - mixed data

Examples

```
todo_list = ["Do the dishes", "Clean my room"]
```

List of strings

```
favourite_numbers = [7, 21, 65]
```

List of integers

```
random_list = [7, "Peter", 9.99]
```

List of mixed data

Creating Lists

Creating Lists

- Lists in Python can be created by just placing the **sequence** inside the **square brackets**

```
my_list = [1, 2, 3]
```

- Or using the **`list`** function

```
empty_list = list()
```

- A list may contain duplicate values

```
my_list = [1, 2, 3, 2, 3, 3]
```

Splitting Strings to List

- You can use the **split** function to split a string and create a list

```
some_text = "a b c d"
my_list = some_text.split(" ")
print(my_list) # ['a', 'b', 'c', 'd']
```

separator

- You can split by different separators

```
some_text = "a, b, c, d"
my_list = some_text.split(", ")
print(my_list) # ['a', 'b', 'c', 'd']
```

Joining Lists into a String

- You can create a string from a list using **`string.join()`**

```
my_list = ["a", "b", "c"]
print("-".join(my_list)) # a-b-c
```

String separator

- The result of the join function is always a **`string`**
- **Note**: In Python, you can only join a **list of strings**

```
print(" ".join([1, 2, 3])) # error
```

This will not work

[index]

Accessing Elements

Using Indices

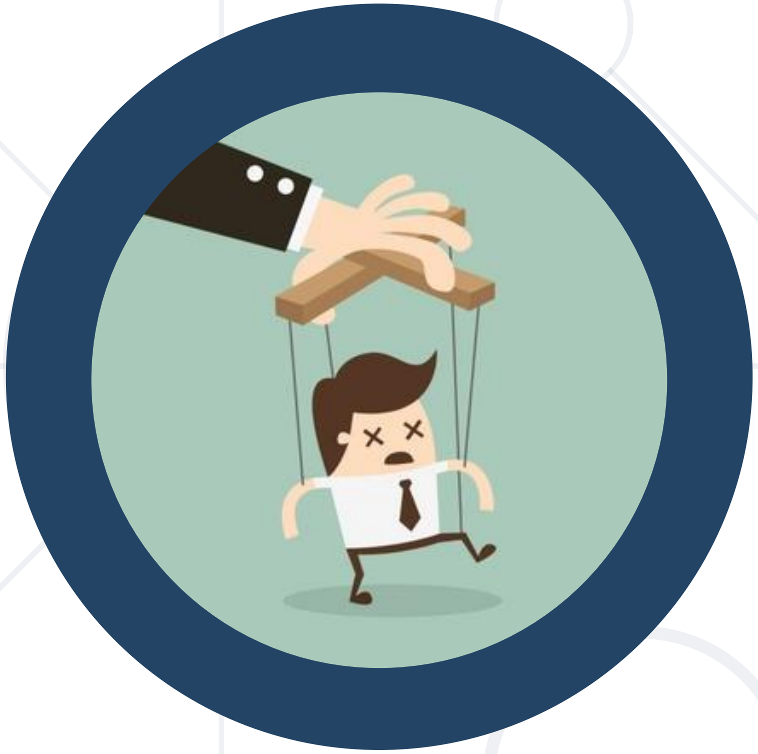
- Use **square brackets** to get an element by an index
- Indices describe the **position** of an element
- We always **start** counting indices from **0**

```
list_of_numbers = [1, 5, 7]
print(list_of_numbers[0]) # 1
print(list_of_numbers[1]) # 5
print(list_of_numbers[2]) # 7
```

Using the "-" Sign

- In Python you can use the negative sign to access an element
- The negative sign will start counting from the end of the list

```
my_pets = ["cat", "dog", "parrot"]
print(my_pets[-1]) # parrot
print(my_pets[-2]) # dog
print(my_pets[-3]) # cat
```



Lists Manipulation

Adding to a List

- Use the **append** function to add a new element

```python
empty_list = []
empty_list.append(2)
empty_list.append(3)
print(empty_list)
# [2, 3]
```

Removing from a List

- Use the **remove** function to remove a particular element

```
list_of_numbers = [1, 2, 3, 4, 5]
list_of_numbers.remove(3)
list_of_numbers.remove(1)
print(list_of_numbers)
# [2, 4, 5]
```

Looping Through Lists

Using for Loop

- There are **two ways** you can loop through a list using **for** loops
  - Iterating over the elements

```
my_list = ["dog", "cat", "fish"]
for element in my_list:
    print(element, end=" ") # dog cat fish
```

  - Using generated list with **range**

```
for index in range(len(my_list)):
    print(my_list[index], end=" ") # dog cat fish
```

Using while Loop

- You can also use **while** loops to iterate through a list
  - In the first example, we iterate until we reach the end of the list
  - In the second example, we iterate until there are no more elements in the list

```
my_list = ["dog", "cat", "fish"]
i = 0
while i < len(my_list) :
    print(my_list[i], end=" ")
    i += 1
```

```
my_list = ["dog", "cat", "fish"]
while my_list:
    print(my_list[0], end=" ")
    current_element = my_list[0]
    my_list.remove(current_element)
```

Searching for Elements

in Keyword

- Use the keyword "**in**" to check if an element is in a list

```
my_list = [1, 2, 3, 4]
if 3 in my_list:
    print("The number 3 is in the list")
```

- Usually the "**in**" keyword is used with **if-else** statements

not in Keywords

- The "**not in**" keywords are used to check if an element is **NOT** in a list

```
my_list = [1, 2, 3, 4]
if 5 not in my_list:
    print("The number 5 is not in the list")
```

- The "**not in**" keywords are also mainly used with **if-else** statements

Summary

- We learned:
  - What lists are in Python
  - How to create lists
  - How to add and remove elements from lists
  - How to loop through lists and access its elements

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License