AIO25

GRID137

June 14, 2025

This documentation is created solely for the purpose of tracking my learning path at AIO 2025

Contents

List of Figures

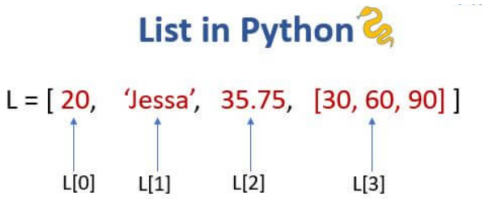
List of Tables

Chapter 1

Module 1 - Week 2

1.1 List definition

A list is a built-in dynamic sized array. We can store all types of items (including another list) in a list.



1.2 List Properties

Mutable	The elements of the list can be modified. We can add or remove items to the list.
Ordered	The items in the lists are ordered. Each item has a unique index value. The new items will be added at the end of the list.
Heterogenous	The list can contain different kinds of elements i.e.; they can contain elements of string, integer, float etc.
Duplicates	The list can contain duplicates i.e., lists can have two items with the same value.

Table 1.1: Properties of Python lists

1.3 List operations

1.3.1 Creating a Python list

We can create a List in Python using 2 ways:

- Using list() constructor.
- Using square bracket([]).

```python
my_list1 = list((1, 2, 3))
print(my_list1)
# Output [1, 2, 3]
```

Figure 1.1: Creating list using constructor

```python
my_list2 = [1, 2, 3]
print(my_list2)
# Output [1, 2, 3]
```

Figure 1.2: Creating list using square bracket

1.3.2 Length of a List

Definition: Length of a List = the number of items present in a list.

We can get the length of a List using len() function.

```python
my_list = [1, 2, 3]
print(len(my_list))
# output 3
```

Figure 1.3: Calculate the length of a list using len()

1.3.3 Accessing items of a List

Items in a List could be accessed by:

- Indexing.
- Slicing.

Indexing

List is 0-indexed, which means if we want to access the 3rd element, for example, we need to use 2 since the index value starts from 0.

There are two types of indexing technique:

- Positive indexing.
- Negative indexing.

Figure 1.4: Positive and negative indexing

Common errors:

- Access an item with an index more than Lists length, it will throw the 'Index Error'.
- If we give any other type except for integer for index values, then it will throw Type Error.

```
my_list = [10, 20, 'Jessa', 12.50, 'Emma']
# accessing 2nd element of the list
print(my_list[1])  # 20
# accessing 5th element of the list
print(my_list[4])  # 'Emma'
```

Figure 1.5: Examples of positive indexing

Negative Indexing: The elements in the list can be accessed from right to left by using negative indexing. The negative value starts from -1 to -length of the list. It indicates that the list is indexed from the reverse/backward.

```
my_list = [10, 20, 'Jessa', 12.50, 'Emma']
# accessing last element of the list
print(my_list[-1])
# output 'Emma'
```

Figure 1.6: Examples of negative indexing

Slicing

Definition: Slicing a list implies, accessing a range of elements in a list.For example, if we want to get the elements in the position from 3 to 7, we can use the slicing method. We can even modify the values in a range by using this slicing technique.

Syntax for slicing:

```
listname[start_index : end_index : step]
```

- The `start_index` denotes the index position from where the slicing should begin and the `end_index` parameter denotes the index positions till which the slicing should be done.
- The `step` allows you to take each nth-element within a `start_index:end_index` range.

Figure 1.7: Slicing syntax

```
my_list = [5, 8, 'Tom', 7.50, 'Emma']

# slice first four items
print(my_list[:4])
# Output [5, 8, 'Tom', 7.5]

# print every second element
# with a skip count 2
print(my_list[::2])
# Output [5, 'Tom', 'Emma']

# reversing the list
print(my_list[::-1])
# Output ['Emma', 7.5, 'Tom', 8, 5]

# Without end_value
# Stating from 3nd item to last item
print(my_list[3:])
# Output [7.5, 'Emma']
```

Figure 1.8: Examples of slicing

1.3.4 Iterating a List

The objects in the list can be iterated over one by one, by using a for a loop.

```
my_list = [5, 8, 'Tom', 7.50, 'Emma']

# iterate a list
for item in my_list:
    print(item)
```

Figure 1.9: List iterating example

1.3.5 Adding elements to the List

Method	Syntax	Usage
append()	mylist.append('Emma')	Only accept one parameter and add it at the end of the list
insert()	insert(index, object)	Add the object/item at the specified position in the list
extend()	mylist.extend([25, 75, 100])	Accept the list of elements and add them at the end of the list, We can even add another list.

Table 1.2: Methods for adding elements in List

1.3.6 Modify the items of a List

- Modify the individual item.

```
my_list = list([2, 4, 6, 8, 10, 12])

# modify single item
my_list[0] = 20
print(my_list)
# Output [20, 4, 6, 8, 10, 12]
```

Figure 1.10: Modify a single item.

- Modify the range of items.

```
# modify range of items
# modify from 1st index to 4th
my_list[1:4] = [40, 60, 80]
print(my_list)
# Output [20, 40, 60, 80, 10, 12]
```

Figure 1.11: Modify range of items.

- Modify all items.

```
my_list = list([2, 4, 6, 8])

# change value of all items
for i in range(len(my_list)):
    # calculate square of each number
    square = my_list[i] * my_list[i]
    my_list[i] = square

print(my_list)
# Output [4, 16, 36, 64]
```

Figure 1.12: Modify a single item.

1.3.7 Removing items from a List

Method	Description
remove(item)	To remove THE FIRST OCCURRENCE of the item on the list.
pop(index)	Remove and return the item at the given index
clear()	To remove all item from the list. The output will be an empty list
del listname	Delete entire list

Table 1.3: List Item Removal Methods in Python

1.3.8 Finding an element in the list

We can use the index() function to find an item in a list.

Syntax: The index() func will accept the value of the element as a parameter and returns the first occurrence of the element or returns ValueError if the element does not exist.

```python
my_list = list([2, 4, 6, 8, 10, 12])

print(my_list.index(8))
# Output 3

# returns error since the element does not exist in the list.
# my_list.index(100)
```

Figure 1.13: Examples of using index() func to get find the element in a list.

1.3.9 Concatenation of two lists

- Using the + operator.
- Using the extend() method to append a new list at the end.

```python
my_list1 = [1, 2, 3]
my_list2 = [4, 5, 6]

# Using + operator
my_list3 = my_list1 + my_list2
print(my_list3)
# Output [1, 2, 3, 4, 5, 6]

# Using extend() method
my_list1.extend(my_list2)
print(my_list1)
# Output [1, 2, 3, 4, 5, 6]
```

Figure 1.14: Lists concatenation.

1.3.10 Copying a list

Using assignment operator (=)

This is called Deep Copying, the changes made to the original list will be reflected in the new one.

```
my_list1 = [1, 2, 3]

# Using = operator
new_list = my_list1
# printing the new list
print(new_list)
# Output [1, 2, 3]

# making changes in the original list
my_list1.append(4)

# print both copies
print(my_list1)
# result [1, 2, 3, 4]
print(new_list)
# result [1, 2, 3, 4]
```

Figure 1.15: Copying list using assignment operator.

Using the copy() method

The copy method can be used to create a copy of a list. This will create a new list and any changes made in the original list will not reflect in the new list. This is shallow copying.

```
my_list1 = [1, 2, 3]

# Using copy() method
new_list = my_list1.copy()
# printing the new list
print(new_list)
# Output [1, 2, 3]

# making changes in the original list
my_list1.append(4)

# print both copies
print(my_list1)
# result [1, 2, 3, 4]
print(new_list)
# result [1, 2, 3]
```

Figure 1.16: Copying list using copy() method.

1.3.11 Sort List using sort()

The sort function sorts the elements in the list in ascending order.

```
mylist = [3,2,1]
mylist.sort()
print(mylist)
```

Figure 1.17: Sort a list in ascending order.

Output: [1, 2, 3]

1.3.12 Reverse a list using reverse()

Output: [1, 6, 5, 4, 3]

1.4 Built-in functions with List

1.4.1 max() and min()

The max function returns the maximum value in the list while the min function returns the minimum value in the list.

```
mylist = [3, 4, 5, 6, 1]
mylist.reverse()
print(mylist)
```

Figure 1.18: Reversing a List.

```
mylist = [3, 4, 5, 6, 1]
print(max(mylist)) #returns the maximum number in the list.
print(min(mylist)) #returns the minimum number in the list.
```

Figure 1.19: Calculate the maximum and minimum of a List

1.4.2 sum()

The sum function returns the sum of all the elements in the list.

```
mylist = [3, 4, 5, 6, 1]
print(sum(mylist))
```

Figure 1.20: Calculate the sum of a List

Output: 19

1.4.3 zip()

Takes iterable containers and returns a single iterator object, having mapped values from all the containers.

Output: [('John', 85), ('Alice', 90), ('Bob', 78), ('Lucy', 92)]

1.4.4 reversed()

Returns a reversed iterator object.

```
names = ['John', 'Alice', 'Bob', 'Lucy']
scores = [85, 90, 78, 92]

res = zip(names, scores)
print(list(res))
```

Figure 1.21: Example of using zip() function

```
a = [1, 2, 3, 4, 5]

# Use reversed() to create an iterator
# and convert it back to a list
rev = list(reversed(a))
print(rev)
```

Figure 1.22: Example of using reversed() function

Output: [5, 4, 3, 2, 1]

1.4.5 sorted()

Returns an arranged list.
Syntax: sorted(iterable, key=key, reverse=reverse)

- iterable - required. The list to sort.
- key - optional. A Function to execute to decide the order. Default is None
- reverse - optional. A Boolean. False will sort ascending, True will sort descending. Default is False

1.4.6 enumerate()

Adds a counter to an iterable and returns it as an enumerate object (iterator with index and the value)
Syntax: enumerate(iterable, start=0)

- Iterable: any object that supports iteration
- Start: the index value from which the counter is to be started, by default it is 0

1.5 2D List

1.5.1 2D List Definition

1	2	3
4	5	6
7	8	0

Figure 1.23: From a graphical perspective, a 2D list represents a table.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Figure 1.24: From a mathematical perspective, a 2D list represents a matrix.

-> From a programming perspective, it's lists inside another list.

1.5.2 Working with 2D List

2D List indexes

When working with 2D List, its indexes are commonly divided into 2 categories:

- Row Index.
- Column Index.

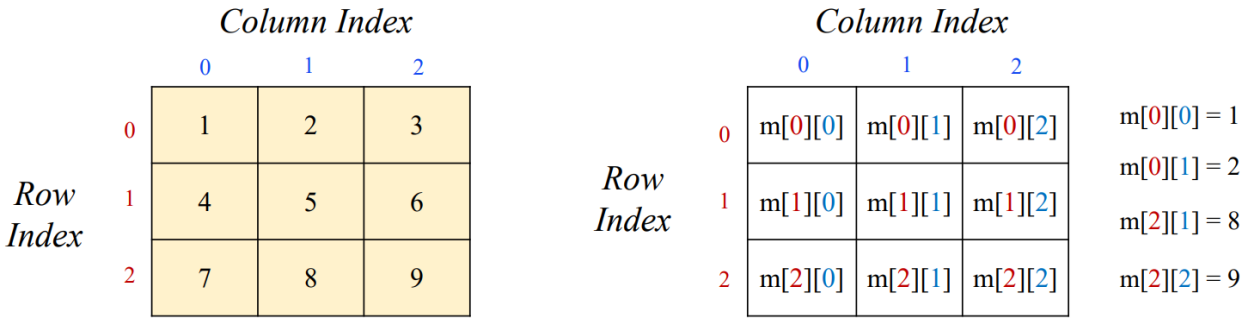


Figure 1.25: Row indexes and Column indexes

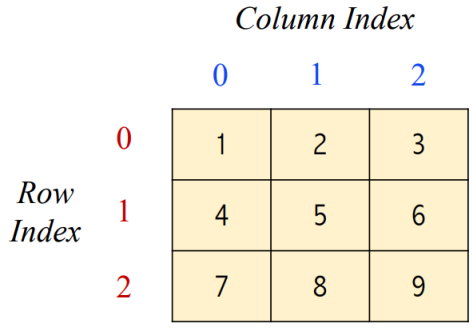
Create 2D List

$$m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$$

Accessing Elements

m[r][c]: the value at row r and column c.

Iterating over a 2D Matrix



- Sol 1:

```
for row in m:
 for element in row:
  print(element, end=' ')
 print()
```

- Sol 2:

```
for r in range(num_rows):
 for c in range(num_cols):
  print(m[r][c], end=' ')
 print()
```

Update elements in 2D matrix

```
matrix[r][c] = new_value
```