Day5_List_Built-in_DS_1

May 21, 2025

1 Day 5: Lists in Python

Today, I explored lists in Python — one of the most important and versatile data structures. I learned that lists are used to store multiple items in a single variable, and they can contain integers, floats, strings, or even other lists (nested lists).

I practiced how to create lists using square brackets [], how to access elements using indexing and slicing, and how to update or delete elements. I also learned how to use built-in list methods like append(), insert(), remove(), pop(), sort(), reverse(), and clear().

A major part of today's learning was understanding how flexible lists are—allowing mixed data types, duplicate values, and dynamic modification. Lists are especially useful in tasks like data storage, iteration, and real-world problem solving.

2 What is a data structure:

A data structure is a way to store and organize data. It helps to use data easily and efficiently. It is useful for storing multiple values and managing data in a better way.

2.0.1 Data type:

A data type stores only one value. Example: a = 1 b = 2.2 c = 'Hello'

2.0.2 Data structure:

A data structure is a collection of one or more data types. Example: my_list = [1, 2, 3] my_tuple = (4, 5, 6) my_set = {1, 2, 3} my_dict = {'name': 'Akshay'}

Matrix: A matrix is a collection of rows and columns. It is used to store 2D data.

3 Types of data structures in Python:

1. Built-in data structures: These are already available in Python. Examples: list, tuple, set, dictionary
2. User-defined data structures: These are created by the user. Examples: stack, queue, linked list, tree, heap, graph, array

4 List in Python

We define a list using square brackets [ ]. For example, l = [ ]

In Jupyter Notebook, to see all functions related to a list, type the list name then a dot (.) and press Tab. For example, l. then Tab

A list can store duplicate values. For example, l = [1, 2, 2, 3]

A list is growable, which means we can add more elements anytime.

A list is mutable, so we can change, add, or remove elements.

A new element is always added at the end by default using the append() function. For example, l.append(5)

List supports indexing. The first element is at index 0. For example, l[0] will give the first item.

List supports slicing to get a part of the list. For example, l[1:3] will give elements from index 1 to 2.

List can hold different data types together. For example, l = [1, “hello”, 3.5]

list inside list called nasted list

4.0.1 List Creation

[1]:
```
list1 = [] # Empty List
print(type(list1))
```

```
<class 'list'>
```

[2]:
```
list2 = [10,20,30,90,80,70,50] # List of Integer Numbers
list3 = [10.5,8.6,99.99,4.6] # List of Float Numbers
list4 = ['One','Two','Three','four','Five'] # List of Strings
list5 = ['Akshay',25,[50,90],[5.6,9.5],['AA','hhh']] # Nested list & mixed data␣
 ↪types
list6 = ['Asif', 25 ,[50, 100],[150, 90] , {'John' , 'David'}]
```

[3]:
```
print('Integer list :', list2)
print('Float list :', list3)
print('String list :', list4)
print('Nested and mixed list :', list5)
print('Another mixed list :', list6)
```

```
Integer list : [10, 20, 30, 90, 80, 70, 50]
Float list : [10.5, 8.6, 99.99, 4.6]
String list : ['One', 'Two', 'Three', 'four', 'Five']
Nested and mixed list : ['Akshay', 25, [50, 90], [5.6, 9.5], ['AA', 'hhh']]
Another mixed list : ['Asif', 25, [50, 100], [150, 90], {'John', 'David'}]
```

[4]:
```
# Length of each list
print('Length of Integer list:', len(list2))
```

```
print('Length of Float list:', len(list3))
print('Length of String list:', len(list4))
print('Length of Nested and mixed list:', len(list5))
print('Length of Another mixed list:', len(list6))
```

```
Length of Integer list: 7
Length of Float list: 4
Length of String list: 5
Length of Nested and mixed list: 5
Length of Another mixed list: 5
```

4.0.2 List Index

List index starts with 0.

Index can be divided into 3 types:

Forward index goes from left to right and starts with 0.

Backward index goes from right to left and starts with -1.

Slice uses three values: start, stop, and step. Step means how many items to jump while slicing.

4.0.3 Indexing examples

[6]:
```
# Indexing examples
print(list4[0])        # 'one' - first element
print(list4[0][0])     # 'o' - first character of first element
print(list4[-1])       # 'Five' - last element
print(list5[-1])       # ['AA', 'hhh'] - last element (nested list)
```

```
One
O
Five
['AA', 'hhh']
```

4.0.4 List slicing

[7]:
```
# List slicing
mylist = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']

print(mylist[0:3])    # ['one', 'two', 'three']
print(mylist[2:5])    # ['three', 'four', 'five']
print(mylist[:3])     # ['one', 'two', 'three']
print(mylist[:2])     # ['one', 'two']
print(mylist[-3:])    # ['six', 'seven', 'eight']
print(mylist[-2:])    # ['seven', 'eight']
print(mylist[-1])     # 'eight'
print(mylist[:])      # whole list
```

```
['one', 'two', 'three']
['three', 'four', 'five']
['one', 'two', 'three']
['one', 'two']
['six', 'seven', 'eight']
['seven', 'eight']
eight
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

4.0.5 Adding items

[8]:
```
# Adding items
mylist.append('nine')      # add at end
mylist.insert(9, 'ten')    # add at index 9

print(mylist)
```

```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

4.0.6 Changing list items

[9]:
```
# Changing list items
mylist[0] = 1
mylist[1] = 2
mylist[2] = 3
print(mylist)  # [1, 2, 3, 'four', 'five', ...]
```

```
[1, 2, 3, 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

4.0.7 Inserting and removing items

[ ]:
```
# Inserting and removing items
mylist.insert(1, 'ONE')
print(mylist)

mylist.remove('ONE')
print(mylist)

mylist.pop()     # remove last item
print(mylist)
```

4.0.8 Remove specific index

[10]:
```
# Remove specific index
if len(mylist) > 8:
    mylist.pop(8)
print(mylist)
```

```
[1, 2, 3, 'four', 'five', 'six', 'seven', 'eight', 'ten']
```

4.0.9 Delete item at index 7 if exists

[11]:
```
# Delete item at index 7 if exists
if len(mylist) > 7:
    del mylist[7]
print(mylist)
```

```
[1, 2, 3, 'four', 'five', 'six', 'seven', 'ten']
```

4.0.10 Clear entire list

[12]:
```
# Clear entire list
mylist.clear()
print(mylist)  # []
```

```
[]
```

[13]:
```
# Recreate list for next steps
mylist = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

4.0.11 Copying lists

[14]:
```
# Copying lists
mylist1 = mylist           # Reference copy (both point to same list)
mylist2 = mylist.copy()    # Actual copy with different address
```

[15]:
```
print(id(mylist), id(mylist1))  # same id
print(id(mylist2))
```

```
2566231843200 2566231843200
2566231915840
```

Change original to show effect on copies

[ ]:
```
# Change original to show effect on copies
mylist[0] = 1
print(mylist)   # [1, 'two', 'three', ...]
print(mylist1)  # [1, 'two', 'three', ...] same because reference copy
print(mylist2)  # ['one', 'two', 'three', ...] unaffected copy
```

4.0.12 Joining lists

[16]:
```
# Joining lists
list1 = ['one', 'two', 'three', 'four']
list2 = ['five', 'six', 'seven', 'eight']
list3 = list1 + list2
print(list3)
```

```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

4.0.13 Reverse list

[17]:
```
# Reverse list
list1.reverse()
print(list1)
```

```
['four', 'three', 'two', 'one']
```

4.0.14 Or using slicing

[19]:
```
# Or using slicing
list1 = list1[::-1]
print(list1)
```

```
['one', 'two', 'three', 'four']
```

4.0.15 Sorting lists

[20]:
```
# Sorting lists
mylist3 = [9, 5, 2, 99, 12, 88, 34]
mylist3.sort()             # Ascending sort
print(mylist3)

mylist3.sort(reverse=True)  # Descending sort
print(mylist3)

mylist4 = [88, 65, 33, 21, 11, 98]
print(sorted(mylist4))  # returns sorted list without modifying original
print(mylist4)
```

```
[2, 5, 9, 12, 34, 88, 99]
[99, 88, 34, 12, 9, 5, 2]
[11, 21, 33, 65, 88, 98]
[88, 65, 33, 21, 11, 98]
```

4.0.16 Loop through list

[21]:
```
# Loop through list
for i in list1:
    print(i)
```

```
one
two
three
four
```

4.0.17 all() and any() examples

[22]:
```
# all() and any() examples
lst_bool1 = [True, True, False]
lst_bool2 = [True, True, True]

print(all(lst_bool1))  # False, because one False
print(all(lst_bool2))  # True, all True

print(any(lst_bool1))  # True, because at least one True
print(any([False, False]))  # False, none True
```

```
False
True
True
False
```

[ ]:
```

```