

எங்கள் வாழ்வும் எங்கள் வளமும்
மங்காத தமிழ் என்று சங்கே முழங்கு ... புரட்சிக்கவி

NOTICE

www.DataScienceInTamil.com

Day 4, 6 and 6 - Batch 3 - Python Language-List (Collection Data Type) and its properties

Watch the 'List' video in You Tube : Day 3, 4, 5 and 6

<https://youtu.be/YzllUGi8l4M> - List - day 3 recording

<https://youtu.be/2uNA2stOffE> - List - day 4 recording

<https://youtu.be/QbovDYjKjNM> - List - day 5 recording

<https://youtu.be/fdjLbQrNtTw> - List - day 6 recording

இந்த குழுவில் உங்கள் நண்பர்களை இணைக்க விரும்பினால் அதற்கான லிங்க்

To join DataScienceInTamil Telegram group:

<https://t.me/joinchat/lUZEsR-zidpjZjEx>

To Join the class, please fill the form :

<https://forms.gle/QFpLHwAoinFaX2cE6>

To watch the recorded videos in YouTube :

<https://www.youtube.com/channel/UCTCMjShTpZg96cXloCO9q1w>

Official Website:

<https://DataScienceInTamil.com/>

Join Zoom Meeting (From Sep 26 2022 to Oct 26 2022)

<https://us06web.zoom.us/j/88900302653?pwd=MVBFUlhqTTE1LzFFRUVpTzZ2S1Vsdz09>

Meeting ID: 889 0030 2653

Passcode: 1234

Monday through Friday 8 PM to 10 PM (IST)

மேலும் முக்கிய கேள்விகள் பதில்களுக்கு

<https://www.DatascienceInTamil.com/#faq>

- We support open-source products to spread Technology to the mass.
- This is completely a FREE training course to provide introduction to Python language
- All materials / contents / images/ examples and logo used in this document are owned by the respective companies / websites. We use those contents for FREE teaching purposes only.
- We take utmost care to provide credits whenever we use materials from external source/s. If we missed to acknowledge any content that we had used here, please feel free to inform us at info@DataScienceInTamil.com.
- All the programming examples in this document are for FREE teaching purposes only.

Thanks to all the open-source community and to the below websites from where we take references / content /code example. definitions, please use these websites for further reading:

- Python Notes For Professionals
- <https://docs.python.org>
- <https://www.w3schools.com>
- <https://www.geeksforgeeks.org>
- <https://docs.python.org>
- <https://www.askpython.com>
- <https://docs.python.org>
- <https://www.programiz.com>

What to cover today?

- 1. Python Collections / container**
- 2. Create a List**
- 3. List items by referring index number**
- 4. Range of Negative Indexes**
- 5. Change Item / element Value**
- 6. Loop Through a List**
- 7. Check if Item Exists (**in** operator)**

8. List Length

9. List methods and supported operators

1. **append(value)**
2. **insert(index, value)**
3. **remove(value)**
4. **pop([index])**
5. **del()**
6. **Multiple Element deletion >> ::2, ::3**
7. **index(value, [startIndex])**
8. **clear()**
9. **reverse()**
10. **count(value)**
11. **sort()**
12. **Can't sort a list of int and str**
13. **Join Two Lists (+)**
14. **extend(enumerable)**

15. Extend using OPERATORS

10. The list() Constructor

11. Multiplying an existing list (*)

12. Copying

13. There are 5 ways of copying

1. using assignment operator → `lst2 = lst1`

2. using list() constructor → `list(list1)`

3. using copy() → (import copy package)





















4. using deepcopy() → (import copy package)

5. using slicing → `b = a[:]`

14. Copy vs Deep copy ()

Topic : List

Python Data Structures

	Indexing	Ordered	Mutable	Duplicate
 List				
 Tuple				
 Set				
 Dictionary				

There are four collection data types in the Python programming language:

Collection is heterogeneous (means it accepts any data type, where as array is homogeneous)

- **List** is a collection which is ordered and changeable. Index is possible, Allows duplicate members. not hasable ..no hash-id/ unhashable type
-
- **Tuple** is a collection which is ordered and unchangeable. Index is possible, . Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members, Index is NOT possible
- **Dictionary** is a collection which is ordered, changeable, and indexed (only key can be indexed – after changing the dict to list. No duplicate key members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

The Python List is a general data structure widely used in Python programs. They are found in other languages, often referred to as dynamic arrays. They are both mutable and a sequence data type that allows them to be indexed and sliced. The list can contain different types of objects, including other list objects.

Python Collections

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
 - An ordered collection of n values ($n \geq 0$)
 - Lists are mutable, so you can change the values in a list
 - A list contains items separated by commas and enclosed within square brackets [].
lists are almost similar to arrays in C. One difference is that all the items belonging to a list can be same or of different data type.
 - Not hashable; mutable (immutable/ hashable)
1. The elements of a list can be accessed via an index, or numeric representation of their position. Lists in Python are zero-indexed
 2. In Python, a list is merely an ordered collection of valid Python values. A list can be created by enclosing values, separated by commas, in square brackets
 3. The elements / items of a list are not restricted to a single data type, which makes sense given that Python is a dynamic data language:

Example

Create a List

```
thislist = ["apple", "banana", "cherry"]  
print(thislist)
```

```
thislist = ["orange", "Mango", "Neuter", range, range, range(5)]  
print(thislist)
```

output

```
['orange', 'Mango', 'Neuter', <class 'range'>, <class 'range'>, range(0, 5)]
```

```
mixed_list = [1, 'abc', True, 2.34, None]  
print(mixed_list)
```

```
a = [1, 2, 3]  
print(a)  
b = ['a', 1, 'python', (1,2)]  
b[2] = 'something else'  
print(b)
```

```
# a = [5, 10, "Linda"]  
# print(a)  
# print("Original LIST", id(a))
```

```
b = ['a', 1, 'python', (1,2) , [], {} ]
print (b)
print("id of list b is ", id(b))
print("id of item Python is ", id(b[2]) )
print("=====")
b[2] = 'Java'
print(b)
print("Modified LIST", id(b))
print("id of item Java is ", id(b[2]) )
```

Output

```
['a', 1, 'python', (1, 2), [], {}]
id of list b is 2227036726912
id of item Python is 2227035976048
=====
['a', 1, 'Java', (1, 2), [], {}]
Modified LIST 2227036726912
id of item Java is 2227036048240
```

You access the list items / elements by referring to the index number:

Example

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]  
print(thislist[1])
```

Negative Indexing

Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

Example

Print the last item of the list:

```
thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])
```

Range of Indexes / slicing

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a **new list** with the specified items.

Example

Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:5])
```

Note: The search will start at index **2 (included)** and end at index 5 (excluded)

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to "orange":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[:4])
```

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "cherry" and to the end:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:])
```

```
-----  
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
# print(thislist[0:5])
```

```
# print(thislist[:5])  
# print(thislist[:])  
print(thislist[2:])
```

```
for item in enumerate(thislist):  
    print(item)
```

```
print("Length is - len() ::", len(thislist))  
print("Id is - id() ::", id(thislist))
```

output

```
['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

```
(0, 'apple')
```

```
(1, 'banana')
```

```
(2, 'cherry')
```

```
(3, 'orange')
```

```
(4, 'kiwi')
```

```
(5, 'melon')
```

```
(6, 'mango')
```

```
Length is - len() :: 7
```

```
Id is - id() :: 26694824
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[-4:-1])
```

Access Items

```
silapthikaram = ["சிலப்பதிகாரம்", 21, "Sr.Data Analyst", "Single", "Ph.D", "Canara Bank",  
"sb_54", False]  
print(silapthikaram[2])  
print(silapthikaram[-2])  
print(silapthikaram[1:-1])  
print(silapthikaram[1::2])
```

Sr.Data Analyst

sb_54

[21, 'Sr.Data Analyst', 'Single', 'Ph.D', 'Canara Bank', 'sb_54']

[21, 'Single', 'Canara Bank', False]

Another pgm for negative ranges

```

thislist = ["apple", "banana", "cherry", "Nector", "Halwa"]
print(thislist[4])
print(thislist[0:4])
print(thislist[4:5])
print(thislist[4:10])
print("=====")

print(thislist[4:4])
print(thislist[3:3]) # in range of index, first value and second value are equal
print(thislist[4:2]) # in range of index, first value should be larger than second value
print("=====")

print(thislist[-3:-1])
print(thislist[-5:-2])

```

output

Halwa

['apple', 'banana', 'cherry', 'Nector']

['Halwa']

['Halwa']

=====

[]

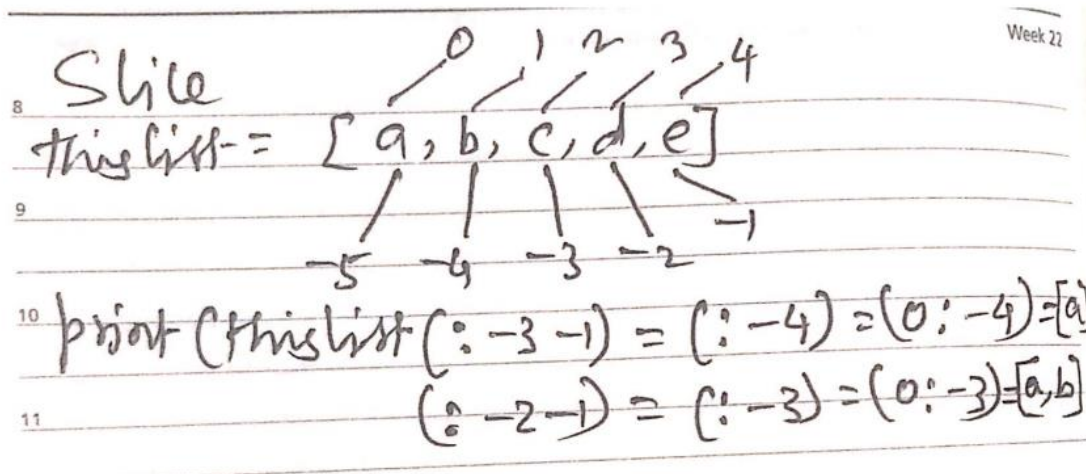
[]

[]

=====

['cherry', 'Nector']

['apple', 'banana', 'cherry



Additional exercise for slicing

```
print(thislist[:-3:-1]) # need more info
print(thislist[:-3 -1:]) # need more info
```

```
print(thislist[-4-1:-1])
print(thislist[-4-1:-1-1]) # (-5 : -2) (Arithmetic based)
-----
```

[-1 : -3 : -1] = [: -3 : -1] - both will give same results
[: -3 :] = [: -3 : 1] = [0 : -3 : 1] - all three will give same results

1. If there is no starting point and step, then list will automatically consider 0 as starting point & 1 as step
 2. If there is no starting point and step is there, then starting point will be decided based on the step, that is if step is 1, then starting point will be 0, if step is -1, then starting point will be -1
-

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Loop Through a List

You can loop through the list items by using a for loop:

Example

Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
    print(x)
```

what happens to the ids, if a list has duplicate items

Ans: the duplicate items share same id

```
thislist = ["apple", "apple", "banana", "banana", "cherry", "Nector", "Halwa", "Apple"]
for item in thislist:
    # print(id(item))
    # print(item, id(item))
    print("item is : ", item, ", item's id is", id(item))
```

output

item is : apple , item's id is 2567657416880

item is : apple , item's id is 2567657416880

item is : banana , item's id is 2567657413808

item is : banana , item's id is 2567657413808

item is : cherry , item's id is 2567657442160

item is : Nector , item's id is 2567657441968

item is : Halwa , item's id is 2567658127024

item is : Apple , item's id is 2567658127216

Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

Example

Check if "apple" is present in the list:

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

```
if "apple" in thislist:
```

```
    print("Yes, 'apple' is in the fruits list")
```

List Length

To determine how many items a list has, use the len() function:

Example

Print the number of items in the list:

```
thislist = ["apple", "banana", "cherry"]  
print(len(thislist))
```

List methods / functions and supported
operators

Python has a set of built-in methods that you can use on lists/arrays.

Method	Description
<u>append()</u>	Adds an element at the end of the list
<u>clear()</u>	Removes all the elements from the list
<u>copy()</u>	Returns a copy of the list
<u>count()</u>	Returns the number of elements with the specified value
<u>extend()</u>	Add the elements of a list (or any iterable), to the end of the current list
<u>index()</u>	Returns the index of the first element with the specified value
<u>insert()</u>	Adds an element at the specified position
<u>pop()</u>	Removes the element at the specified position
<u>remove()</u>	Removes the first item with the specified value
<u>reverse()</u>	Reverses the order of the list
<u>sort()</u>	Sorts the list



append(value)

- appends a new element to the end of the list

Note that the `append()` method only appends one new element to the end of the list. If you append a list to another list, the list that you append becomes a single element at the end of the first list.

```
a = [1, 2, 3, 4, 5]
a.append(6)
a.append(7)
a.append([80, 90])
print(a)
```

output

```
[1, 2, 3, 4, 5, 6, 7, [80, 90]]
```

In `append()`, if we use `range()`, we will get surprise results, so DON'T use it

```
a = [1, 2, 3, 4, 5, 5]
a.append(range(3))
print(a)
```

output

```
[1, 2, 3, 4, 5, 5, range(0, 3)]
```

To add an item to the end of the list, use the append() method:

Example

Using the append() method to append an item:

```
thislist = ["apple", "banana", "cherry"]  
thislist.append("orange")  
print(thislist)
```

Append an element of a different type, as **list elements do not need to have the same data type**

```
a = [1, 2, 3, 4, 5]  
a.append(6)  
a.append("AAA")  
print(a)
```

output

```
[1, 2, 3, 4, 5, 6, 'AAA']
```

a = [10, 2, 30, 40, 5, 5]
b = [10, 20]


```

a.append(b)
print (a)
print ( a[6] )
print ( a[6] [0] )
print ( a[6] [1] )

print("=====")
for item in enumerate(a):
    print(item)

```

output

```

[10, 2, 30, 40, 5, 5, [10, 20]]
[10, 20]
10
20
=====
(0, 10)
(1, 2)
(2, 30)
(3, 40)
(4, 5)
(5, 5)
(6, [10, 20])

```

TAKING THE 6 INDEX OF THE LIST, WHICH IS ANOTHER LIST THAT WE ADDED JUST NOW

output

```
[1, 2, 3, 4, 5, 5, [10, 20]]  
[10, 20]
```

insert(index, value)

– inserts value just **before the specified index**. Thus after the insertion the new element occupies position index.

a.insert(0, 0) # insert 0 at position 0 a.insert(2, 5) # insert 5 at position

To add an item at the specified index, use the insert() method:

Example

Insert an item as the second **position**:

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

```
for item in enumerate(thislist):  
    print(item)  
print("+++++")
```

```
thislist.insert(2, "orange")  
print(thislist)
```

remove(value)

removes the **first occurrence of the specified value**. If the provided value cannot be found, a `ValueError` is raised.

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

```
thislist.remove('banana')  
print(thislist)
```

`ValueError`, because 'banana1' is not in the list - a

pop([index])

removes and returns the item at index. With no argument it removes and returns the last element of the list.

```
pop(self, index=-1, /)
```

Remove and return item at index (default last).

Raises `IndexError` if list is empty or index is out of range.

Raises `IndexError` if list is empty or index is out of range.

```
a = [0, 1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10]  
print(a.pop(2)) # print(a.pop(-1)) negative index also possible  
print(a)
```

output

2

[0, 1, 3, 4, 5, 6, 7, 7, 8, 9, 10]

Note: slicing and range of index is not possible using pop()

PS: pop() follows LIFO

Example

The pop() method removes the specified index, (or the last item if index is not specified):

```
thislist = ["apple", "banana", "cherry"]  
thislist.pop()  
print(thislist)
```

Pop function

```
names = ['Alice', 'Craig', 'Diana', 'Diana', 'Eric', 'Bob']  
print(names)  
print(id(names))  
print(type(names))  
print(len(names))  
  
print("+++++")  
for item in enumerate(names):  
    print(item)
```

```
print("+++++")
```

```
print (names.pop(4))  
print (names)  
print(id(names))
```

output

```
['Alice', 'Craig', 'Diana', 'Diana', 'Eric', 'Bob']
```

```
25121960
```

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

```
6
```

```
+++++
```

```
(0, 'Alice')
```

```
(1, 'Craig')
```

```
(2, 'Diana')
```

```
(3, 'Diana')
```

```
(4, 'Eric')
```

```
(5, 'Bob')
```

```
+++++
```

```
Eric
```

```
['Alice', 'Craig', 'Diana', 'Diana', 'Bob']
```

```
25121960
```

```
-----
```

Remove() returns item at index (defaults to the last item) with names.pop([index]), returns the item

del()

Example

The del keyword removes the **specified** index:

```
thislist = ["apple", "banana", "cherry"]  
print(id(thislist))  
del thislist[0] # observe..there is no dot/ . after the del  
print(thislist)  
print(id(thislist))
```

del thislist
output

```
37901480  
['banana', 'cherry']  
37901480
```

Example

The del keyword can also delete the list completely:

```
thislist = ["apple", "banana", "cherry"]  
del thislist
```

Multiple Element deletion

it is possible to delete multiple elements in the list using the **del** keyword and slice notation:

```
a = list(range(10))
print(a)
del a[::2] #'every second element is deleted'
del a[::3] #'every 3rd element is deleted'
```

```
print(a)
```

```
# a = [1, 3, 5, 7, 9]
del a[-1]
print(a)
# a = [1, 3, 5, 7]
```

```
del a[:]
print(a)
# a = []
```

output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[1, 3, 5, 7, 9]
[1, 3, 5, 7]
[]
```

Del (list) // deletes the complete list's object

::2 (every other element) ::3 (Every third element) and so on will be removed (the items divided by 3 or 2 will be removed)

```
a = list(range(20))  
del a[::5]  
print(a)
```

Output:

[1, 2, 3, 4, 6, 7, 8, 9, 11, 12, 13, 14, 16, 17, 18, 19]

Note: 5,10,15 will be removed as is where divided by 5

```
=====
```

```
a = list(range(20))  
print(a)  
print(a[8::2] )  
del a[8::2] # from 8, every other element will be removed  
print(a)
```

output

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

[8, 10, 12, 14, 16, 18]

[0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 13, 15, 17, 19]

index(value, [startIndex, stopIndex])

`index(self, value, start=0, stop=9223372036854775807, /)`

Return first index of value.

Raises `ValueError` if the value is not present.

– gets the index of the first occurrence of the input value. If the input value is not in the list a `ValueError` exception is raised. **If a second argument is provided, the search is started at that specified index.**

```
a = [10, 20, 30, 40, 50, 501, 502, 50, 503, 504, "AAA"]
for i in enumerate(a):
    print(i)
```

```
print("=====")
# id, en, help, type, len
print(a.index(40))
print(a.index(50, 5, 8))
```

output : run the code to view the result

```
-----
bank = [10, 20, 30, 40, 50, 50, "ArulMary"]
# print(a.index(40))
if "ArulMary" in bank:
    print(bank.index("ArulMary"))
else:
    print("That bank customer is not available")
```

```
silapthikaram = ["சிலப்பதிகாரம்", 21, "Sr.Data Analyst", "Single", "Ph.D", "Canara Bank",  
"sb_54", 21, False]  
for item in enumerate(silapthikaram):  
    # print(item)  
    pass  
print(silapthikaram.index(21))  
print(silapthikaram.index(21, 3, 8)) # it find the index of 21, from 3rd index to 8th index
```

output

1

7

Notes

a.index(7) # Returns: 6

a.index(49) # ValueError, because 49 is not in a.

a.index(7, 7) # Returns: 7

a.index(7, 8) # ValueError, because there is no 7 starting at index

clear()

removes all items from the list and gives empty list

```
list.clear()
```

Example

The clear() method empties the list:

```
thislist = ["apple", "banana", "cherry"]  
thislist.clear()  
print(thislist)
```

```
-----  
thislist = ["apple", "banana", "cherry"]  
print(id(thislist))  
thislist.clear()  
print(thislist)  
print(id(thislist))
```

```
del(thislist) # we are trying to find the id after del the list from the memory
```

```
print(id(thislist))
```

```
-----
```

reverse()

– reverses the list in-place and returns None

```
a = [10, 20, 30, 40, 50, 50, "AAA"]  
print(a.reverse())
```

output

None

TO AVOID IT AND GET THE REVERSE VALUES USE THE BELOW CODE

```
a = [10, 20, 30, 40, 50, 50, "AAA"]  
a.reverse()  
print(a)
```

OUTPUT

```
['AAA', 50, 50, 40, 30, 20, 10]
```

Or use the below code to reverse the list or any iterable

```
a = [10, 20, 30, 40, 50, 50, "AAA"]  
print(a[::-1])
```

```
['AAA', 50, 50, 40, 30, 20, 10]
```

=====

count(value)

counts the number of occurrences of any value in the list

```
list.count(5)
```

```
a = [10, 20, 30, 40, 50, 50, 50, "AAA"]  
print(a.count(50))
```

output

3

sort()

sorts the list in numerical and lexicographical order and returns None

```
lst = [10, 50, 20, 30, 40, 50, 100, 50,]  
lst.sort()  
print (lst)
```

output

```
[10, 20, 30, 40, 50, 50, 50, 100] # sorts the list in numerical
```

Sort and reverse

```
lst = [10, 50, 20, 30, 40, 50, 100, 50,]  
lst.sort()  
print(lst)
```

```
lst.reverse()
print(lst)
output
[10, 20, 30, 40, 50, 50, 50, 100]
[100, 50, 50, 50, 40, 30, 20, 10]
```

```
-----
lst = ['a', 'c', 's', 'a',] # sorts the list in lexicographical order
lst.sort()
print (lst)
```

```
output
['a', 'a', 'c', 's']
```

```
-----
lst = ["Sudha", 'a', 'c', 's', 'a',] # sorts the list in lexicographical order based on the ASCII
values
lst.sort()
print (lst)
l = [83, 97, 99, 115, 97]
l.sort()
print(l)
```

```
output
-----
```

```
lst = ["Sudha", "Vanitha", "Sabi"] # sorts the list in lexicographical order based on the  
ASCII values
```

```
lst.sort()
```

output

```
['Sabi', 'Sudha', 'Vanitha']
```

if list has str and numerical type, we cant sort, it is an error

```
lst = ["Sudha", "Vanitha", "Sabi", [4,10,3]] # sorts the list in lexicographical order based on  
the ASCII values
```

```
lst.sort()
```

```
print (lst)
```

output

```
TypeError: '<' not supported between instances of 'list' and 'str'
```

**we can not sort a list if it has numerical
and string**

```
lst = ['a', 'c', 's', 'a', 20, 1, 5,]  
lst.sort()  
print (lst)
```

output

TypeError: '<' not supported between instances of 'int' and 'str'

=====

Sort and reverse can be combined together

Lists can also be reversed when sorted using the reverse=True flag in the sort() method.

```
lst = [10, 100, 20, 3, 40, 50, 50,]  
lst.sort(reverse=True)  
print (lst)  
output  
[100, 50, 50, 40, 20, 10, 3]
```

Copy using copy() method

Copy()

Copy a List

You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.

There are ways to make a copy, one way is to use the built-in List method `copy()`.

Example

Make a copy of a list with the `list.copy()` method:

```
thislist = ["apple", "banana", "cherry"]
print("THIS LIST ", thislist)
print(id(thislist))
thislist.append("Sudha")
print(thislist)
print(id(thislist))
print("=====")
```

```
mylist = thislist.copy()
print("MY LIST", mylist)
print(id(mylist))
mylist.append("Linda")
print(mylist)
```

```
print("=====")
print(thislist)
```

output

```
THIS LIST ['apple', 'banana', 'cherry']
```

```

1782073546240
['apple', 'banana', 'cherry', 'Sudha']
1782073546240
=====
MY LIST ['apple', 'banana', 'cherry', 'Sudha']
1782074424896
['apple', 'banana', 'cherry', 'Sudha']
-----

```

Copy using assignment (=)operator

```

originalList = ["apple", "banana", "cherry"]
myList = originalList # copying the original list to "Mylist" using assignment operator
print("ORIGINLA LIST\n", originalList)
originalList.append("Aswathy")
print("MODIFIED ORIGINAL LIST\n", originalList)

print(myList)
print(id(originalList))
print(id(myList))
print("=====")

myList.append("Kivi")
print("MY LIST IS", myList)

```

```
print("ORIGINAL LIST ", originalList)
print(id(originalList))
print(id(myList))
#
```

Output

```
-----
ORIGINAL LIST
['apple', 'banana', 'cherry']
MODIFIED ORIGINAL LIST
['apple', 'banana', 'cherry', 'Aswathy']
['apple', 'banana', 'cherry', 'Aswathy']
2828669310464
2828669310464
=====
MY LIST IS ['apple', 'banana', 'cherry', 'Aswathy', 'Kivi']
ORIGINAL LIST ['apple', 'banana', 'cherry', 'Aswathy', 'Kivi']
2828669310464
2828669310464
=====
```

Copy using list()

Another way to make a copy is to use the built-in method list().

-

Example

Make a copy of a list with the list() method:

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print("MY LIST ",mylist)
print("ID OF THIS LIST", id(thislist))
print("ID OF MY LIST", id(mylist))
print("=====")

thislist.append("Linda")
print("MODIFIED ORIGINAL LIST\n", thislist)
print("=====")

print("MY LIST ",mylist)
#
print("ID OF THIS LIST", id(thislist))
print("ID OF MY LIST", id(mylist))
```

output

```
MY LIST ['apple', 'banana', 'cherry']
ID OF THIS LIST 1332393187968
```

ID OF MY LIST 1332393187968

=====

MODIFIED ORIGINAL LIST

['apple', 'banana', 'cherry', 'Linda']

=====

MY LIST ['apple', 'banana', 'cherry']

ID OF THIS LIST 1332393187968

ID OF MY LIST 1332393187968

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Example

Join two list:

list1 = ["a", "b", "c"]

list2 = [1, 2, 3]

list3 = list1 + list2

print(list3)

Another way to join two lists are by appending all the items from list2 into list1, one by one:

(Don't use this method, the easy way is use extend())

Example

Append list2 into list1:

```
list1 = ["a", "b", "c"]  
print("ID OF LIST1 ", id(list1))
```

```
list2 = ["Aswathy", "Lin", "Su"]  
print("ID OF LIST2 ", id(list2))  
list1.extend(list2)  
print(list1)  
print("ID OF LIST1 ", id(list1))
```

output

```
ID OF LIST1 2275999700480  
ID OF LIST2 2276000573056  
['a', 'b', 'c', 'Aswathy', 'Lin', 'Su']  
ID OF LIST1 2275999700480
```

extend(enumerable)

extends the list by appending elements from another enumerable

```
a = [1, 2, 3, 4, 5, 5]
b = [10, 20]
a.extend(b)
print(a)
output
[1, 2, 3, 4, 5, 5, 10, 20]
```

Or you can use the extend() method, which purpose is to add elements from one list to another list:

Example

Use the extend() method to add list2 at the end of list1:

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
```

```
list1.extend(list2)
print(list1)
```

Adding list using + operator and using extend() gives same result

```

silapthikaram = ["சிலப்பதிகாரம்", 21, "Sr.Data Analyst", "Single", "Ph.D", "Canara Bank",
"sb_54", False]
manimkalai = ["மணிமேகலை", 49, "Data Scientist", "single", "masters",]
tamil = silapthikaram + manimkalai
print(tamil)
print("=====")

(silapthikaram.extend(manimkalai))
print(silapthikaram)

```

can't multiply sequence by non-int of type 'list'

```

list1 = ["a", "b", "c"]
silapthikaram = ["சிலப்பதிகாரம்", 21, "Sr.Data Analyst", "Single", "Ph.D", "Canara Bank",
"sb_54", False]
print(list1 * silapthikaram)

```

output

TypeError: can't multiply sequence by non-int of type 'list'

range(stop) -> range object range(start, stop[, step]) -> range object


```
a = [1, 2, 3, 4, 5, 5]
a.extend(range(3))
print(a)
```

output

```
[1, 2, 3, 4, 5, 5, 0, 1, 2] #0,1,2 IS FROM range()
```

```
-----
a = [1, 2, 3, 4, 5, 5]
a.extend(range(2,10,2)) # THIS range(START WITH 2, GOES UP TO 10, STEP 2)
print(a)
```

OUTOUT

```
[1, 2, 3, 4, 5, 5, 2, 4, 6, 8]
```

Another ex

```
for i in range(10):
    print(i)
print("=====")
```

```
print(list(range(10)))
print("=====")
```

```
list1 = [10,20,30]
list1.extend(range(10))
print(list1)
```

output

Extend of the list can be possible using OPERATORS

```
a = [1, 2, 3, 4, 5, 5]
b = [10, 20, "A", "BBB"]
print (a + b) # WE USE + OPERATOR TO CONCATENATE 2 LISTS /
```

OUTPUT

```
[1, 2, 3, 4, 5, 5, 10, 20, 'A', 'BBB']
```

```
t = [1, 2, 3, [1, 2, 3,]]
print (t[3])
```

```
t[3] = t[3] + [4,5]
print (t[3])
```

output

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

The list() Constructor

It is also possible to use the list() constructor to make a new list.

Example

Using the list() constructor to make a List:

```
thislist = list(("apple", "banana", "cherry")) # note the double round-brackets  
print(thislist)
```

List with enumerate

```
silapthikaram = ["சிலப்பதிகாரம்", 21, "Sr.Data Analyst", "Single", "Ph.D", "Canara Bank",  
"sb_54", False]  
print(silapthikaram)  
print(silapthikaram[ : ])  
print(silapthikaram[ 1:5 ])
```

```
for index, item in enumerate(silapthikaram):  
    print(index, item)
```

Multiplying an existing list

Note : multiplying an existing list by an integer will produce a larger list consisting of that many copies of the original. This can be useful for example for list initialization:

```
lst = [450, "Name", True]
print (lst * 3)
lst1 = [1, 2, 3, 4, 5]
print (lst1 * 5)
```

output

```
[450, 'Name', True, 450, 'Name', True, 450, 'Name', True]
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

Copying

The default assignment "=" assigns a **reference** of the **original list to the new name**. That is, the **original name and new name are both pointing to the same list object (same memory location / pointer**. **Changes made through any of it will be reflected in another**. This is often not what you intended.

There are 5 ways of copying

1. using assignment operator // lst2 = lst1
2. using list() // list(list1)
3. using `copy()` // (import copy package)
4. using `deepcopy()` // (import copy package)
5. using slicing // b = a[:]

copy using assignment operator

the change made in the **list** **did** effect in other lists

In Python, **Assignment statements do not copy objects**, they create bindings between a target and an object. When we use = operator user thinks that this creates a new object; but, it doesn't. **It only creates a new variable that shares the reference (id of the list) of the original object.**

```
import copy
originalList = [10, 20, 3, 40, 50, 50,]
print("Original list", originalList)
print("ID of the original list: ", id(originalList))
duplicateList = originalList
print("duplicate list", duplicateList)
print("ID of the duplicate list: ", id(duplicateList))
```

```

print("=====")

duplicateList.append(1000)

print("Original list after copying and adding new element",originalList)
print("duplicate list after copying and adding new element",duplicateList)
print("=====")

print("ID of the original list : ", id (originalList))
print("ID of the original list after copying and adding new element: ", id (originalList))
print("ID of the duplicate list : ", id (duplicateList))
print("ID of the duplicate list after copying and adding new element: ", id (duplicateList))

```

output

```

Original list [10, 20, 3, 40, 50, 50]
ID of the original list : 25030568
duplicate list [10, 20, 3, 40, 50, 50]
ID of the duplicate list : 25030568
=====
Original list after copying and adding new element [10, 20, 3, 40, 50, 50, 1000]
duplicate list after copying and adding new element [10, 20, 3, 40, 50, 50, 1000]
=====
ID of the original list : 25030568
ID of the original list after copying and adding new element: 25030568
ID of the duplicate list : 25030568
ID of the duplicate list after copying and adding new element: 25030568
=====
Original list after adding new element [10, 20, 3, 40, 50, 50, 1000]
duplicate list after adding new element [10, 20, 3, 40, 50, 50, 1000]
=====

```

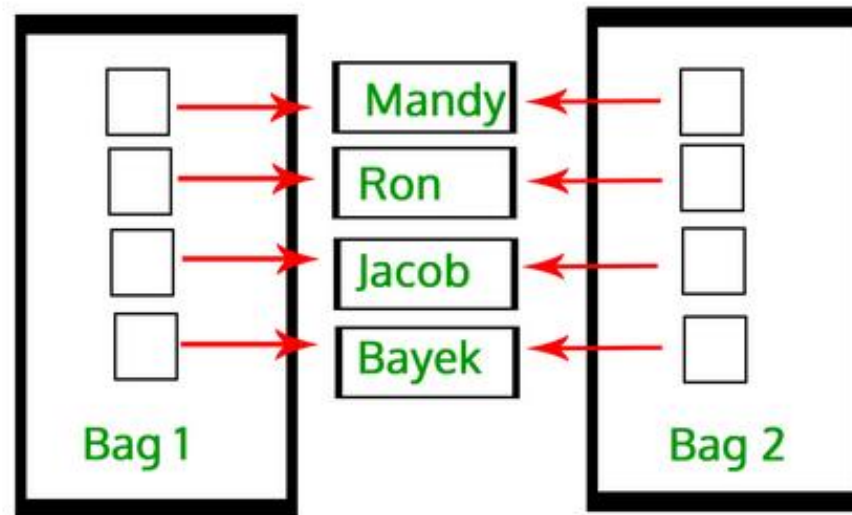
ID of the original list : 35975080
ID of the original list after adding new element: 35975080
ID of the duplicate list : 35975080
ID of the duplicate list after adding new element: 35975080

=====

Shallow copy (using copy.copy)

Shallow copy

Shallow Copy



A shallow copy creates a **new object** which stores the reference of the original elements.

So, a **shallow copy doesn't create a copy of nested objects**, instead it **just copies the reference** of nested objects. This means, a copy process

does not recursive ie, **shallow copy**, ie, **copies** the sub list of list and its reference/memory address (and the content of the address)

This means it will create **new and independent object** with reference + content

the change made in the list did not effect in other lists

Copy using copy() method

```
import copy
originalList = [10, 20, 3, ["Lin", "mel"]]
print("Original list", originalList)
print("ID of the original list : ", id(originalList))

for item in originalList:
    print(id(item))
print("=====")

duplicateList = copy.copy(originalList)
print("duplicate list", duplicateList)
print("ID of the duplicate list : ", id(duplicateList))
for item in duplicateList:
    print(id(item))
print("=====")
```

```

duplicateList.append(1000)

print("Original list after copying and adding new element",originalList)
print("duplicate list after copying and adding new element",duplicateList)
print("=====")

print("ID of the original list : ", id (originalList))
print("ID of the original list after copying and adding new element: ", id (originalList))
print("ID of the duplicate list : ", id (duplicateList))
print("ID of the duplicate list after copying and adding new element: ", id (duplicateList))

```

Output

```

Original list [10, 20, 3, ['Lin', 'mel']]
ID of the original list : 2910226979328
2910221369872
2910221370192
2910221369648
2910226972992 #(Id for ['Lin', 'mel'])
=====
duplicate list [10, 20, 3, ['Lin', 'mel']]

```

ID of the duplicate list : 2910226985280

2910221369872

2910221370192

2910221369648

2910226972992 # (Id for ['Lin', 'mel'])

Note: copy.copy(): if a list has a sub list in it, and if we copy the original list to another list, the id of the sublist (in both lists) will be SAME

=====

Original list after copying and adding new element [10, 20, 3, ['Lin', 'mel']]

duplicate list after copying and adding new element [10, 20, 3, ['Lin', 'mel'], 1000]

=====

ID of the original list : 2910226979328

ID of the original list after copying and adding new element: 2910226979328

ID of the duplicate list : 2910226985280

ID of the duplicate list after copying and adding new element: 2910226985280

=====

How to access the value from the id / memory address (copy.copy())

```
import copy
import ctypes

originalList = [10, 20, 3, ["Lin", "mel"]]
print("Original list", originalList)
print("ID of the original list : ", id(originalList))

for item in originalList:
    print(id(item))
print("=====")

duplicateList = copy.copy(originalList)
print("duplicate list", duplicateList)
print("ID of the duplicate list : ", id(duplicateList))
for item in duplicateList:
    print(id(item))
print("=====")

x = id(originalList[3])
print('ID OF ["Lin", "mel FROM ORIGINAL LIST"]', x)
print("=====")

y = id(duplicateList[3])
print('ID OF ["Lin", "mel FROM DUPLICATE LIST"]', y)
# a = ctypes.cast()
```

```
print("=====")
'''
access values from id
# get the value through memory address
a = ctypes.cast(x, ctypes.py_object).value
'''
a = ctypes.cast(x, ctypes.py_object).value
print(a) # it gives the content(Values or values) of the memory address
```

output

Original list [10, 20, 3, ['Lin', 'mel']]

ID of the original list : 2141546755712

2141537501712

2141537502032

2141537501488

2141546692800 # copy.copy() gives same id

=====

duplicate list [10, 20, 3, ['Lin', 'mel']]

ID of the duplicate list : 2141546584320

2141537501712

2141537502032

2141537501488

2141546692800 # copy.copy() gives same id

=====

ID OF ["Lin", "me1 FROM ORIGINAL LIST"] 2141546692800

=====

ID OF ["Lin", "me1 FROM DUPLICATE LIST"] 2141546692800

=====

['Lin', 'me1']

=====

Deepcopy (using copy.deepcopy)

How to access the value from the id / memory address (copy.deepcopy())

```
import copy
import ctypes

originalList = [10, 20, 3, ["Lin", "mel"]]
print("Original list", originalList)
print("ID of the original list : ", id(originalList))

for item in originalList:
    print(id(item))
print("=====")

duplicateList = copy.deepcopy(originalList)
print("duplicate list", duplicateList)
print("ID of the duplicate list : ", id(duplicateList))
for item in duplicateList:
    print(id(item))
print("=====")

x = id(originalList[3])
print('ID OF ["Lin", "mel FROM ORIGINAL LIST"]', x)
print("=====")

y = id(duplicateList[3])
print('ID OF ["Lin", "mel FROM DUPLICATE LIST"]', y)
# a = ctypes.cast()
```

```
print("=====")
'''
access values from id
# get the value through memory address
a = ctypes.cast(x, ctypes.py_object).value
'''
a = ctypes.cast(x, ctypes.py_object).value
print(a) # it gives the content(Values or values) of the memory address
```

output

Original list [10, 20, 3, ['Lin', 'mel']]

ID of the original list : 2282978490048

2282972709392

2282972709712

2282972709168

2282978427072 ## copy.deepcopy() gives different id

=====

duplicate list [10, 20, 3, ['Lin', 'mel']]

ID of the duplicate list : 2282978318784

2282972709392

2282972709712

2282972709168

2282978323392

=====

ID OF ["Lin", "mel FROM ORIGINAL LIST"] 2282978427072

=====

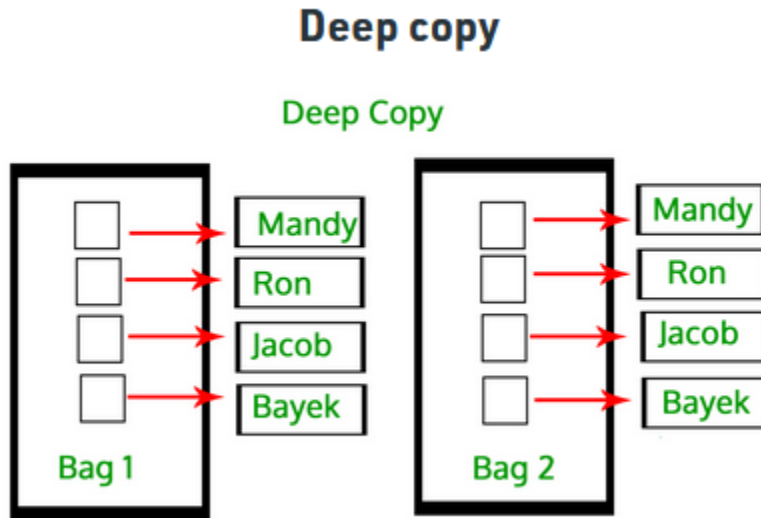
ID OF ["Lin", "mel FROM DUPLICATE LIST"] 2282978323392

=====

['Lin', 'mel'] # copy.deepcopy() gives different id

=====

the change made in the list did not effect in other lists



Copy using deepcopy() method

A deep copy creates a new object and **recursively adds the copies of nested objects** present in the original elements.

The deep copy creates independent copy of original object and all its nested objects. (Table contents + All chapters)

```

import copy
originalList = [10, 20, 3, ["Lin", "mel"]]
print("Original list", originalList)
print("ID of the original list : ", id (originalList))

for item in (originalList):
    print(id(item))
print("=====")

duplicateList = copy.deepcopy(originalList)
print("duplicate list", duplicateList)
print("ID of the duplicate list : ", id (duplicateList))
for item in (duplicateList):
    print(id(item))
print("=====")

duplicateList.append(1000)

print("Original list after copying and adding new element", originalList)
print("duplicate list after copying and adding new element", duplicateList)
print("=====")

print("ID of the original list : ", id (originalList))
print("ID of the original list after copying and adding new element: ", id (originalList))
print("ID of the duplicate list : ", id (duplicateList))
print("ID of the duplicate list after copying and adding new element: ", id (duplicateList))

```

output

C:\Users\Melcose\Programs\Python\Python310\python.exe

C:/Users/Melcose/PycharmProjects/pythonProject/Sudha_Linda.py

Original list [10, 20, 3, ['Lin', 'mel']]

ID of the original list : 1125794290240

1125788680720

1125788681040

1125788680496

1125794283840 # id of the sublists ['Lin', 'mel']]

=====

duplicate list [10, 20, 3, ['Lin', 'mel']]

ID of the duplicate list : 1125794296128

1125788680720

1125788681040

1125788680496

1125794294656 # id of the sublist ['Lin', 'mel']

Note: copy.copy(): if a list has a sub list in it, and if we copy the original list to another list, the id of the sublist (in both lists) will **VARY**

=====

Original list after copying and adding new element [10, 20, 3, ['Lin', 'mel']]

duplicate list after copying and adding new element [10, 20, 3, ['Lin', 'mel'], 1000]

=====

ID of the original list : 1125794290240

ID of the original list after copying and adding new element: 1125794290240

ID of the duplicate list : 1125794296128

ID of the duplicate list after copying and adding new element: 1125794296128

Process finished with exit code 0

b = a[:] // alternative to list.copy()

```
a = [3,4,5, "Kala", "Anna"]
print(id(a))
print("Ori\n", a)
b = a[ : ]
print(b)
print("=====")
```

```
b.append("Kamaraj")
print(b)
print(a)
print(id(b))
print(id(a))
```

```
output
12080296
Ori [3, 4, 5, 'Kala', 'Anna']
[3, 4, 5, 'Kala', 'Anna']
[3, 4, 5, 'Kala', 'Anna', 'Kamaraj']
[3, 4, 5, 'Kala', 'Anna']
12083176
12080296
```
