LIST_VHA

November 1, 2025

1 LIST

2 We will cover the following topics in this chapter

What are Lists?

Lists Vs Arrays

Characterstics of a List

How to create a list

Access items from a List

Editing items in a List

Deleting items from a List

Operations on Lists

Functions on Lists

3 INTRO

The primary data structure of Python is the sequence. Every element of a sequence is assigned a number—its position or index. The first index will be zero, the second index will be one, and so on.

There are six types of sequences in Python, but lists and tuples are the most common among these

4 What are Lists

List is a data type where you can store multiple items under 1 name. More technically, lists act like dynamic arrays which means you can add more items on the fly.

Lists are heterogenous data structures created by elements separated with commas and enclosed within square brackets.

5 Array Vs Lists

Fixed Vs Dynamic Size(int arr[50] in c)

Convenience -> Hetrogeneous (convenience=float)

Speed of Execution(list slow)

Memory(list require more space)

6 How lists are stored in memory

[8]:
```
#referantioal array means store referance address or pointer
L = [1,2,3]

print(id(L))
print(id(L[0]))
print(id(L[1]))
print(id(L[2]))
print(id(1))
print(id(2))
print(id(3))
```

```
2045914107136
2045830654256
2045830654288
2045830654320
2045830654256
2045830654288
2045830654320
```

7 Characterstics of a List

Ordered

Changeble/Mutable

Hetrogeneous

Can have duplicates

are dynamic

can be nested

items can be accessed

can contain any kind of objects in python

[10]:
```
#ordered
l=[1,2,3]
l1=[3,2,1]
l==l1
```

[10]:
```
False
```

[11]:
```
#changable/mutable
l=[1,2,3]
l[0]=5
print(l)
```

```
[5, 2, 3]
```

[13]:
```
#Hetrogeneous
l=[1,1.0,"apple",(9,0),{1,0},{"a":1,"b":3}]
print(l)
```

```
[1, 1.0, 'apple', (9, 0), {0, 1}, {'a': 1, 'b': 3}]
```

[14]:
```
#duplicate
l=[1,1,1,2,2,3,3,4,4,5,5,6]
print(l)
```

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

[15]:
```
#dynamic
l=[1,2,3]
l.append(5)
l.append((1,2))
print(l)
```

```
[1, 2, 3, 5, (1, 2)]
```

[16]:
```
#can be nested
l=[1,2,3,[1,2]]
print(l)
```

```
[1, 2, 3, [1, 2]]
```

[17]:
```
#items can be accessed
l=[1,2,3,[1,2]]
print(l[0])
print(l[3][1])
```

```
1
2
```

8 Creating List

[18]:
```
#empty list
l=[]
print(l)
```

```
[]
```

```
[59]: l=list()
      print(l)
      l1=list((1,2,3,4))
      print(l1)
```

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

```
[21]: #1d list
      l=[1,2,3]
      l1=["a","b","c"]
      l2=[(1,2),(3,4),(5,6)]
      l3=[1,'a',(1,2),{1,2}]
      print(l,l1,l2,l3)
```

```
[1, 2, 3] ['a', 'b', 'c'] [(1, 2), (3, 4), (5, 6)] [1, 'a', (1, 2), {1, 2}]
```

```
[23]: #2D list
      l=[[1,2],[3,4]]
      print(l)
```

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

```
[26]: #3d list
      l=[[[1,2],[3,4]],[[5,6],[7,8]]]
      print(l)
```

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

```
[27]: #using type conversation
      l=list("hello")
      print(l)
```

```
['h', 'e', 'l', 'l', 'o']
```

```
[60]: l1=list((1,2,3,4))
      print(l1)
```

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

```
[61]: l1=list({1,2,3})
      print(l1)
```

```
[1, 2, 3]
```

9 Accessing items from a list

indexing

slicing

```
[29]: #posative indexing
l=[1,2,3,4]
print(l[0])
print(l[2])
print(l[4])
```

```
1
3
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\2737391842.py in <module>
      3 print(l[0])
      4 print(l[2])
----> 5 print(l[4])

IndexError: list index out of range
```

```
[30]: #negative indexing
l=[1,2,3,4]
print(l[-1])
print(l[-2])
print(l[-4])
print(l[-5])
```

```
4
3
1
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\3623638630.py in <module>
      4 print(l[-2])
      5 print(l[-4])
----> 6 print(l[-5])

IndexError: list index out of range
```

```
[33]: l=[1,2,3,[1,2],"abcd"]
print(l[3][-1])
print(l[3][1])
print(l[4][:2])
```

```
2
2
ab
```

```
[38]: #slicing
      l=[1,2,3,4]
      print(l[::2])
      print(l[1:3])
      print(l[-1:-4:-2])
```

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

```
[45]: l=[1,2,3,[1,2],"abcd"]
      print(l[1:4])
      print(l[2:5])
      print(l[2:5][2][1:2])
```

```
[2, 3, [1, 2]]
[3, [1, 2], 'abcd']
b
```

```
[50]: l=[1,2,3,[1,2],"abcd"]
      print(l[-1:-3:-1][-1][-2])
```

```
1
```

```
[51]: l=[1,2,3,4]
      print(l[::-1])
```

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

```
[52]: l=[1,2,3,[1,2],"abcd"]
      print(l[::-1])
```

```
['abcd', [1, 2], 3, 2, 1]
```

10 Adding Items to a List

append

extend

insert

11 List append() method

The append() method appends a passed obj to the existing list.given only one argument take any data type as argument

Syntax:

list.append(obj)

Parameters

The object to be appended to the list.

Return value

There is no return value, but the updated list is printed.

[53]:
```python
# append
L = [1,2,3,4,5]
L.append(True)
print(L)
```

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

[54]:
```python
# append
L = [1,2,3,4,5]
L.append([1,2,3])
print(L)
```

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

[79]:
```python
# append
L = [1,2,3,4,5]
print(L.append("abcdf"))
print(L)
```

```
None
[1, 2, 3, 4, 5, 'abcdf']
```

[56]:
```python
# append
L = [1,2,3,4,5]
L.append((1,2))
print(L)
```

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

[57]:
```python
# append
L = [1,2,3,4,5]
L.append(1,2)
print(L)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\567615818.py in <module>
      1 # append
      2 L = [1,2,3,4,5]
----> 3 L.append(1,2)
      4 print(L)

TypeError: list.append() takes exactly one argument (2 given)
```

12 List extend() method

The extend() method appends the contents of seq to list:

Syntax:

list.extend(sequence)

sequence is the list's element.

Return value

The extend() method does not return any value but adds the content to an existing list.

[63]:
```
# extend
L = [1,2,3,4,5]
L.extend([6,7,8])
print(L)
```

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

[64]:
```
# extend
L = [1,2,3,4,5]
L.extend("hello")
print(L)
```

```
[1, 2, 3, 4, 5, 'h', 'e', 'l', 'l', 'o']
```

[65]:
```
# extend
L = [1,2,3,4,5]
L.extend(range(10))
print(L)
```

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

[66]:
```
# extend
L = [1,2,3,4,5]
L.extend((1,2,3))
print(L)
```

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

[78]:
```
# extend
L = [1,2,3,4,5]
print(L.extend([6,7,[1,8,"a"]]))
print(L)
```

```
None
[1, 2, 3, 4, 5, 6, 7, [1, 8, 'a']]
```

[76]:
```
# extend
L = [1,2,3,4,5]
L.extend(1,2,3)
```

```
print(L)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\3092189676.py in <module>
      1 # extend
      2 L = [1,2,3,4,5]
----> 3 print(L.extend(1,2,3))
      4 print(L)

TypeError: list.extend() takes exactly one argument (3 given)
```

13 List insert() method

The insert() method inserts object into list at offset index.

Syntax:

list.insert(index, obj)

Parameters

index - This is the index; the object to be inserted

obj - This is the object to be inserted into the list

Return value

It does not return a value; rather, it inserts the given element at the given index.

[70]:
```
# insert
L = [1,2,3,4,5]

L.insert(1,100)
print(L)
```

```
[1, 100, 2, 3, 4, 5]
```

[71]:
```
# insert
L = [1,2,3,4,5]

L.insert(1,[1,2])
print(L)
```

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

[72]:
```
# insert
L = [1,2,3,4,5]

L.insert(1,{5:4,7:5})
```

```
print(L)
```

```
[1, {5: 4, 7: 5}, 2, 3, 4, 5]
```

[75]:
```
# insert
L = [1,2,3,4,5]

print(L.insert(2,"helo"))
print(L)
```

```
None
[1, 2, 'helo', 3, 4, 5]
```

14 Editing items in a List

editing with indexing

editing with slicing

[80]:
```
L = [1,2,3,4,5]

# editing with indexing
L[-1] = 500

# editing with slicing
L[1:4] = [200,300,400]

print(L)
```

```
[1, 200, 300, 400, 500]
```

[81]:
```
L = [1,2,3,4,5]

# editing with indexing
L[-1] = 500

# editing with slicing
L[1:4] = [200,300]

print(L)
```

```
[1, 200, 300, 500]
```

15 Deleting items from a List

Deleting list elements When deleting a list element, you can use either of the del statements to know exactly which element(s) you are deleting. You can use the remove() method if you don't know exactly which items to delete. Rundown can be replaced with the collection as the development community is familiar with this term rather than the name rundown.

del

remove

pop

clear

[82]:
```
# del
L = [1,2,3,4,5]

# indexing
del L[-1]

# slicing
del L[1:3]
print(L)
```

```
[1, 4]
```

[86]:
```
# del
L = [1,2,3,4,5]

# indexing
del L[5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\1200053711.py in <module>
      3
      4 # indexing
----> 5 del L[5]
      6

IndexError: list assignment index out of range
```

[83]:
```
# remove

L = [1,2,3,4,5]

L.remove(5)

print(L)
```

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

[87]:
```
# del
L = [1,2,3,4,5]
```

```
# slicing
del L[1:7]
print(L)
```

```
[1]
```

[88]:
```
# remove

L = [1,2,3,4,5,2]

L.remove(2)

print(L)
```

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

[89]:
```
# remove

L = [1,2,3,4,5,2]

L.remove(6)

print(L)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\2374691690.py in <module>
      3 L = [1,2,3,4,5,2]
      4
----> 5 L.remove(6)
      6
      7 print(L)

ValueError: list.remove(x): x not in list
```

[91]:
```
# remove

L = [1,2,3,4,5,(1,2)]

L.remove((1,2))

print(L)
```

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

16 List pop() method

The pop() method eliminates the last item and returns the last item from the list.

Syntax:

list.pop(obj=list[-1])

Parameters

obj – This is the index of the object to be removed from the list, and it is an optional parameter.

Return value

This method returns the removed object from the list.

[90]:
```
# pop
L = [1,2,3,4,5]

L.pop()

print(L)
```

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

[92]:
```
# pop
L = [1,2,3,4,5]

L.pop(2)

print(L)
```

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

[93]:
```
# pop
L = [1,2,3,4,5]

L.pop(6)

print(L)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\2145222892.py in <module>
      2 L = [1,2,3,4,5]
      3 
----> 4 L.pop(6)
      5 
      6 print(L)

IndexError: pop index out of range
```

[94]:
```
# clear
L = [1,2,3,4,5]

L.clear()

print(L)
```

```
[]
```

17 Operations on Lists

Arithmetic

Membership

Loop

[95]:
```
# Arithmetic (+ ,*)

L1 = [1,2,3,4]
L2 = [5,6,7,8]

# Concatenation/Merge
print(L1 + L2)
```

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

[96]:
```
# Arithmetic (+ ,*)
print(L1*3)
```

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

[97]:
```
#Membership
L1 = [1,2,3,4,5]
L2 = [1,2,3,4,[5,6]]

print(5 not in L1)
print([5,6] in L2)
print(5 not in L2)
```

```
False
True
True
```

[98]:
```
L1 = [1,2,3,4,5]
for i in L1:
  print(i)
```

```
1
2
3
```

```
4
5
```

[99]:
```
L3 = [[[1,2],[3,4]],[[5,6],[7,8]]]

for i in L3:
  print(i)
```

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

[100]:
```
L2 = [1,2,3,4,[5,6]]
for i in L2:
  print(i)
```

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

[104]:
```
L1 = [1,2,3,4,5]
for i,j in enumerate(L1):
  print(i,j)
```

```
0 1
1 2
2 3
3 4
4 5
```

18 List function

len

min

max

sum

sort

sorted

count

index

reverse

sort vs sorted

copy

19 List len() method

The len() method returns the total number of components in the rundown.

Syntax:

len(list)

Parameters

list - This is the list in which the elements are to be counted.

Return value

This returns the number of elements in the list.

[106]:
```
L = [2,1,5,7,0]

print(len(L))
```

```
5
```

[107]:
```
L = [2,1,5,7,0,(7,8),[1,2,3],{1,2,3},{1:2,3:4}]

print(len(L))
```

```
9
```

20 List max() method only for same data type list

This max() method returns the components from the rundown with the most noteworthy worth.

Syntax:

max(list)

Parameters

list - The number of elements in this list is to be counted.

Return value

The max() method returns the components in the rundown with the most elevated worth.

[108]:
```
l=[1,2,3,4,5]
print(max(l))
```

```
5
```

[111]:
```
l=[[1,2],[3,4],[5,6]]
print(max(l))
```

```
[5, 6]
```

```
[110]: l=[1,2,3,4,5,[1,2]]
       print(max(l))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\763757885.py in <module>
      1 l=[1,2,3,4,5,[1,2]]
----> 2 print(max(l))

TypeError: '>' not supported between instances of 'list' and 'int'
```

```
[112]: l=[[1,2],[3,4],[5,4]]
       print(max(l))
```

```
[5, 4]
```

21 List min() method

The min() method returns the components from the rundown with the least worth.

Syntax: min(list)

Parameters

list - This is the list in which the number of elements is to be counted.

Return value This min() method returns the elements from the list with minimum value.

```
[113]: l=[1,2,3,4,5]
       print(max(l))
```

```
5
```

```
[114]: l=["abv","k","k"]
       print(max(l))
```

```
k
```

```
[115]: l=["abv","k","k"]
       print(min(l))
```

```
abv
```

22 sum

add the values in the list that has numbers

```
[117]: print(sum[1,2,3])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\3616098951.py in <module>
----> 1 print(sum[1,2,3])

TypeError: 'builtin_function_or_method' object is not subscriptable
```

[118]:
```
l=[1,2,3]
print(sum(l))
```

```
6
```

[119]:
```
l=['a','b']
print(sum(l))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\1404571346.py in <module>
      1 l=['a','b']
----> 2 print(sum(l))

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

23 List sort() method

The Python list sort() capacity can be utilized to sort a list in climbing, plummeting, or client characterized request.

Syntax:

list.sort([func])

Parameters NA

[132]:
```
l=[1,5,3]
print(l.sort())
print(l)
```

```
None
[1, 3, 5]
```

[135]:
```
l=[1,5,3]
print(sorted(l,reverse=True))
```

```
[5, 3, 1]
```

```
[134]: l=[1,5,3]
       print(sorted(l))
```

```
[1, 3, 5]
```

```
[127]: l=[[1,2],[5,4],[3,6]]
       l.sort()
       print(l)
```

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

```
[128]: l=[[1,2],[5,4],[3,6]]
       print(sorted(l,reverse=False))
```

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

```
[129]: l=[1,"a",5]
       l.sort()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\3434189602.py in <module>
      1 l=[1,"a",5]
----> 2 l.sort()

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

```
[130]: l=[1,"m",2]
       print(sorted(l,reverse=False))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\2957039629.py in <module>
      1 l=[1,"m",2]
----> 2 print(sorted(l,reverse=False))

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

```
[131]: # sort (vs sorted)
       L = [2,1,5,7,0]
       print(L)
       print(sorted(L))
       print(L)
       L.sort()
       print(L)
```

```
[2, 1, 5, 7, 0]
```

```
[0, 1, 2, 5, 7]
[2, 1, 5, 7, 0]
[0, 1, 2, 5, 7]
```

24 List count() method

The count() method returns the count of how frequently obj occurs in the list.

Syntax: list.count(obj)

Parameters

obj - Object to be counted in the list.

Return value

The count() method returns the count of how many times obj occurs in the list.

```
[137]: l=[1,2,1,"a",(1,2),(3,4),(1,2),"a"]
       print(l.count(1))
       print(l.count('a'))
       print(l.count((1,2)))
```

```
2
2
2
```

25 List index() method

The index() method returns the lowest index in list that obj appears.

Syntax:

list.index(obj)

Parameters

obj – obj is the object to be determined.

Return value

This method returns the index of the found object or raises an exception indicating that the value was not found.

```
[138]: # index
       L = [1,2,1,3,4,1,5]
       L.index(1)
```

```
[138]: 0
```

```
[145]: # index
       L = [1,2,1,3,4,1,5]
       L.index(1,1,9)
```

[145]: 2

[139]:
```
# index
L = [1,2,1,3,4,1,(1,5)]
L.index((1,5))
```

[139]: 6

[140]:
```
# index
L = [1,2,1,3,4,1,5,[6,4]]
L.index(6)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26276\1487934888.py in <module>
      1 # index
      2 L = [1,2,1,3,4,1,5,[6,4]]
----> 3 L.index(6)

ValueError: 6 is not in list
```

[141]:
```
# index
L = [1,2,1,3,4,1,5,[6,4]]
L.index([6,4])
```

[141]: 7

26 List reverse() method

The reverse() method reverses the objects of the list in place.

Syntax:

list.reverse()

Parameters NA

Return value

The reverse() method does not return any value; rather, it reverses the given object from the list.

[146]:
```
# reverse
L = [2,1,5,7,0]
# permanently reverses the list
L.reverse()
print(L)
```

```
[0, 7, 5, 1, 2]
```

[147]:
```
# reverse
L = [2,1,5,7,0]
# permanently reverses the list
print(L.reverse())
print(L)
```

```
None
[0, 7, 5, 1, 2]
```

[148]:
```
# copy -> shallow
L = [2,1,5,7,0]
print(L)
print(id(L))
L1 = L.copy()
print(L1)
print(id(L1))
```

```
[2, 1, 5, 7, 0]
2045915042688
[2, 1, 5, 7, 0]
2045914975360
```

[149]:
```
# copy -> shallow
L = [2,1,5,7,0]
print(L)
print(id(L))
L1 = L[::]
print(L1)
print(id(L1))
```

```
[2, 1, 5, 7, 0]
2045915043328
[2, 1, 5, 7, 0]
2045915042560
```

[153]:
```
# not copy -> shallow
L = [2,1,5,7,0]
print(L)
print(id(L))
L1 = L
print(L1)
print(id(L1))
L.append(7)
print(L)
print(L1)
```

```
[2, 1, 5, 7, 0]
2045915042240
[2, 1, 5, 7, 0]
```

```
2045915042240
[2, 1, 5, 7, 0, 7]
[2, 1, 5, 7, 0, 7]
```

[152]:
```python
a=5
print(a)
print(id(a))
b=a
print(b)
print(id(b))
a+=5
print(a)
print(b)
```

```
5
2045830654384
5
2045830654384
10
5
```

27 reverse for all datatype

-The reversed() function allows us to process the items in a sequence in reverse order. It accepts a sequence and returns an iterator.

Syntax:

reversed(sequence) -> reverse iterator

[4]:
```python
print( reversed([44, 11, -90, 55, 3]) )

print(list(reversed([44, 11, -90, 55, 3]))) # reversing a list

print( list(reversed((6, 1, 3, 9)))) # reversing a tuple

print(list(reversed("hello"))) # reversing a string
```

```
<list_reverseiterator object at 0x0000024605C00B20>
[3, 55, -90, 11, 44]
[9, 3, 1, 6]
['o', 'l', 'l', 'e', 'h']
```

[5]:
```python
print( reversed([44, 11, -90, 55, 3]) )

print(tuple(reversed([44, 11, -90, 55, 3]))) # reversing a list

print( tuple(reversed((6, 1, 3, 9)))) # reversing a tuple
print(tuple(reversed("hello"))) # reversing a string
```

```
<list_reverseiterator object at 0x0000024605C61430>
(3, 55, -90, 11, 44)
(9, 3, 1, 6)
('o', 'l', 'l', 'e', 'h')
```

28 Python eval(): Evaluate Expressions Dynamically

[10]:
```
inp=(eval(input()))
print(type(inp))
```

```
5
<class 'int'>
```

[11]:
```
inp=(eval(input()))
print(type(inp))
```

```
'vishal10'
<class 'str'>
```

[12]:
```
inp=(eval(input()))
print(type(inp))
```

```
["j","o",9,(5,8)]
<class 'list'>
```

[13]:
```
inp=(eval(input()))
print(type(inp))
```

```
{1,2,3}
<class 'set'>
```

[14]:
```
inp=(eval(input()))
print(type(inp))
```

```
(5,)
<class 'tuple'>
```

[15]:
```
inp=(eval(input()))
print(type(inp))
```

```
{1:7,2:7}
<class 'dict'>
```

29 Python sorted() Method

- The sorted() method returns a sorted list from the specified iterables (string, list, tuple, set).
- Syntax:

sorted(iterable, key, reverse)

Parameters: iterable: The iterable to be arranged in ascending or descending order.

key: (Optional) A function that serves as a key for the sort comparison.

reverse: (Optional) If true, sorts in descending order.

Return Value: ###### Returns a list object with sorted items.

- The following example returns the sorted list of the elements of iterable.

[16]:
```python
nums = [2,1,5,3,4]
asc_nums = sorted(nums)
dsc_nums = sorted(nums, reverse = True)
print("Ascending Numbers: ", asc_nums)
print("Descending Numbers: ", dsc_nums)

nums_tuple = (8,7,6,10,9)
asc_nums = sorted(nums_tuple)
dsc_nums = sorted(nums_tuple, reverse = True)
print("Ascending Numbers: ", asc_nums)
print("Descending Numbers: ", dsc_nums)

mystr = 'gcadbfe'
asc_str = sorted(mystr)
dsc_str = sorted(mystr, reverse = True)
print("Ascending String: ", asc_str)
print("Reversed String: ", dsc_str)
```

```
Ascending Numbers:  [1, 2, 3, 4, 5]
Descending Numbers:  [5, 4, 3, 2, 1]
Ascending Numbers:  [6, 7, 8, 9, 10]
Descending Numbers:  [10, 9, 8, 7, 6]
Ascending String:  ['a', 'b', 'c', 'd', 'e', 'f', 'g']
Reversed String:  ['g', 'f', 'e', 'd', 'c', 'b', 'a']
```

[17]:
```python
numdict = {1:'One',3:'Three', 2:'Two'}
asc_nums = sorted(numdict)
dsc_nums = sorted(numdict, reverse=True)

print("Ascending List: ", asc_nums)
print("Descending List: ", dsc_nums)
```

```
Ascending List:  [1, 2, 3]
Descending List:  [3, 2, 1]
```

30 Sort using Custom Function as Key

- The key parameter can be used to sort the iterable based on different functions.

[18]:
```
numstr = ('One','Two','Three','Four')
asc_nums = sorted(numstr, key=len)
dsc_nums = sorted(numstr, key=len, reverse=True)

print("Ascending List: ", asc_nums)
print("Descending List: ", dsc_nums)
```

```
Ascending List:  ['One', 'Two', 'Four', 'Three']
Descending List:  ['Three', 'Four', 'One', 'Two']
```

31 You can use set user-defined function or lambda function to the key parameter. For example, the following sorts the list of string by the last character of a string.

[19]:
```
def getlastchar(s):
        return s[len(s)-1]

code = ('bb','cd', 'aa', 'zc')
asc_code = sorted(code, key=getlastchar) # using user-defined function
dsc_code = sorted(code, key=getlastchar, reverse=True)

print("Ascending Code: ", asc_code)
print("Descending Code: ", dsc_code)

print('----Using lambda function----')
asc_code = sorted(code, key=lambda s: s[len(s)-1]) # using lambda function
dsc_code = sorted(code, key=lambda s: s[len(s)-1], reverse=True)

print("Ascending Code: ", asc_code)
print("Descending Code: ", dsc_code)
```

```
Ascending Code:  ['aa', 'bb', 'zc', 'cd']
Descending Code:  ['cd', 'zc', 'bb', 'aa']
----Using lambda function----
Ascending Code:  ['aa', 'bb', 'zc', 'cd']
Descending Code:  ['cd', 'zc', 'bb', 'aa']
```

32 Python enumerate() Method

- The enumerate() is a constructor method returns an object of the enumerate class for the given iterable, sequence, iterator, or object that supports iteration. The returned enumerate object contains tuples for each item in the iterable that includes an index and the values obtained from iterating over iterable. #### Syntax:
- enumerate(iterable, start=0)

Return Value:

Returns an enumerate object.

The following example gets an object of the enumerate class for the list and converts enumerate to list.-

[20]:
```
cities = ['Delhi','Chicago','New York']
enum = enumerate(cities)
print(type(enum))

enumlist = list(enum)
print(enumlist)
```

```
<class 'enumerate'>
[(0, 'Delhi'), (1, 'Chicago'), (2, 'New York')]
```

33 In the above example, the default index starts from 0, but we can change the initial counter to any number.

[21]:
```
cities = ['Delhi','Chicago','New York']
enum = enumerate(cities, start=5)
print(list(enum))
```

```
[(5, 'Delhi'), (6, 'Chicago'), (7, 'New York')]
```

34 The enumerate() function can be used with loops as follows.

[22]:
```
for index, city in enumerate(cities):
    print(index,city)
```

```
0 Delhi
1 Chicago
2 New York
```

[24]:
```
x="VISHAL10"
print(list(enumerate(x)))
print(tuple(enumerate(x)))
print(dict(enumerate(x)))
```

```
[(0, 'V'), (1, 'I'), (2, 'S'), (3, 'H'), (4, 'A'), (5, 'L'), (6, '1'), (7, '0')]
((0, 'V'), (1, 'I'), (2, 'S'), (3, 'H'), (4, 'A'), (5, 'L'), (6, '1'), (7, '0'))
{0: 'V', 1: 'I', 2: 'S', 3: 'H', 4: 'A', 5: 'L', 6: '1', 7: '0'}
```

[34]:
```
x=(8,0,9,7)
print(list(enumerate(x)))
print(tuple(enumerate(x)))
print(dict(enumerate(x)))
```

```
[(0, 8), (1, 0), (2, 9), (3, 7)]
((0, 8), (1, 0), (2, 9), (3, 7))
{0: 8, 1: 0, 2: 9, 3: 7}
```

[32]:
```
x={1,2,3}
print(list(enumerate(x)))
print(tuple(enumerate(x)))
print(dict(enumerate(x)))
```

```
[(0, 1), (1, 2), (2, 3)]
((0, 1), (1, 2), (2, 3))
{0: 1, 1: 2, 2: 3}
```

35 What is List Comprehension?

- List comprehension is a short and elegant way to create lists in Python. It allows you to generate a new list by applying an expression to each item in an existing iterable (like a list, tuple, or range

35.0.1 [expression for item in iterable if condition]

expression → The value or operation you want to store in the list.

item → The variable representing each element from the iterable.

iterable → A sequence (like a list, string, range, etc.).

condition (optional) → Filters items before applying the expression

[1]:
```
squares = [x**2 for x in range(1, 6)]
print(squares)
```

```
[1, 4, 9, 16, 25]
```

[2]:
```
evens = [x for x in range(1, 11) if x % 2 == 0]
print(evens)
```

```
[2, 4, 6, 8, 10]
```

[3]:
```
words = ["apple", "banana", "cherry"]
uppercase_words = [word.upper() for word in words]
print(uppercase_words)
```

```
['APPLE', 'BANANA', 'CHERRY']
```

[4]:
```
matrix = [[1, 2], [3, 4], [5, 6]]
flat = [num for row in matrix for num in row]
print(flat)
```

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

[5]:
```
labels = ["Even" if x % 2 == 0 else "Odd" for x in range(1, 6)]
print(labels)
```

```
['Odd', 'Even', 'Odd', 'Even', 'Odd']
```

[ ]:
```

```