Data Structures in python

Data structures are divided in to two types In built & User Defined

- In in-built we have 1) LIST 2) TUPLE 3) SET 4) DICIONARY 5) RANGE

1) LIST -

A list in Python is a collection of items enclosed in square brackets [ ] and separated by commas. Lists are versatile and can store elements of different data types

List Characteristics

- Ordered : Lists maintain the order of elements.
- Mutable : Items can be changed after creation.
- Allow Duplicates : Lists can contain duplicate values

```
In [23]: l = []
```

```
In [24]: l
```

```
Out[24]: []
```

```
In [25]: l = [10,20,30,40,50]
         l
```

```
Out[25]: [10, 20, 30, 40, 50]
```

```
In [26]: len(l)
```

```
Out[26]: 5
```

```
In [27]: type(l)
```

```
Out[27]: list
```

Functions in list

- 1. Append( ) 2) copy( ) 3) count( ) 4) remove( ) 5) clear( ) 6) Extend( ) 7)Index( ) 8)Insert( ) 9) pop( ) 10) reverse( ) 11) sort( )

Append()

The append() method in Python adds a single element to the end of a list. It modifies the original list,we can append various types of elements to a list, including integers, strings, floats, and even other lists. However, when appending a list to another list using append(), the entire list is added as a single element

- Append() will take only one argument

In [28]:
```
l1 = []
l1
```

Out[28]:
```
[]
```

In [29]:
```
type(l1)
```

Out[29]:
```
list
```

In [30]:
```
len(l1)
```

Out[30]:
```
0
```

In [31]:
```
l1.append(5)
```

In [32]:
```
l1
```

Out[32]:
```
[5]
```

In [33]:
```
l1.append(20)
l1
```

Out[33]:
```
[5, 20]
```

In [34]:
```
l1.append(25)
l1
```

Out[34]:
```
[5, 20, 25]
```

In [35]:
```
l1.append('shravani')
l1
```

Out[35]:
```
[5, 20, 25, 'shravani']
```

In [36]:
```
l1.append(2.3)
l1
```

Out[36]:
```
[5, 20, 25, 'shravani', 2.3]
```

In [37]:
```
l1.append(True)
l1.append(10+2j)
```

```
l1
```

Out[37]:
```
[5, 20, 25, 'shravani', 2.3, True, (10+2j)]
```

Copy()

- copy from one list to another list
- The method creates a new list containing the same elements as the original list but maintains its own identity in memory.

In [38]:
```
print(l)
print(l1)
```

```
[10, 20, 30, 40, 50]
[5, 20, 25, 'shravani', 2.3, True, (10+2j)]
```

In [39]:
```
l2 = ' shravani', ' shaanu ' , ' chinnu'
l2
```

Out[39]:
```
(' shravani', ' shaanu ', ' chinnu')
```

In [40]:
```
print(l)
print(l1)
print(l2)
```

```
[10, 20, 30, 40, 50]
[5, 20, 25, 'shravani', 2.3, True, (10+2j)]
(' shravani', ' shaanu ', ' chinnu')
```

In [41]:
```
l2 = l.copy()
```

In [42]:
```
l2
```

Out[42]:
```
[10, 20, 30, 40, 50]
```

In [43]:
```
print(l)
print(l1)
print(l2)
```

```
[10, 20, 30, 40, 50]
[5, 20, 25, 'shravani', 2.3, True, (10+2j)]
[10, 20, 30, 40, 50]
```

Count()

- The count() method in Python is used to count the number of occurrences of a specified element in a list. It returns the count of how many times an element is present in the list

In [44]:
```
l
```

```
Out[44]: [10, 20, 30, 40, 50]
```

```
In [45]: l.count(10)
```

```
Out[45]: 1
```

```
In [46]: l.count ( 20)
```

```
Out[46]: 1
```

```
In [47]: l1
```

```
Out[47]: [5, 20, 25, 'shravani', 2.3, True, (10+2j)]
```

```
In [48]: l1.count('shravani')
```

```
Out[48]: 1
```

Remove()

- The .remove() method in Python is used to delete the first occurrence of a specified value from a list. It modifies the list in place and raises a ValueError if the element is not found
- It will take exactly one argument

```
In [49]: print(l)
         print(l1)
         print(l2)
```

```
[10, 20, 30, 40, 50]
[5, 20, 25, 'shravani', 2.3, True, (10+2j)]
[10, 20, 30, 40, 50]
```

```
In [50]: l.remove(30)
```

```
In [51]: l
```

```
Out[51]: [10, 20, 40, 50]
```

```
In [52]: l.remove (10,20,30)          ## as it takes only one argument, that's why thowr
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[52], line 1
----> 1 l.remove (10,20,30)          ## as it takes only one argument, that's wh
y thowrn an error

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

In [53]:
```
l2.remove(20)
```

In [54]:
```
l2
```

Out[54]:
```
[10, 30, 40, 50]
```

In [55]:
```
l1.remove(True)
```

In [56]:
```
l1
```

Out[56]:
```
[5, 20, 25, 'shravani', 2.3, (10+2j)]
```

Clear()

- The clear() method in Python is a built-in function that removes all items from a list, effectively making it an empty list. This method does not delete the list itself but clears its content

In [57]:
```
print(l)
print(l1)
print(l2)
```

```
[10, 20, 40, 50]
[5, 20, 25, 'shravani', 2.3, (10+2j)]
[10, 30, 40, 50]
```

In [58]:
```
l.clear()
```

In [59]:
```
l
```

Out[59]:
```
[]
```

In [60]:
```
print(l)
print(l1)
print(l2)
```

```
[]
[5, 20, 25, 'shravani', 2.3, (10+2j)]
[10, 30, 40, 50]
```

Extend()

- In Python, extend() method is used to add items from one list to the end of another list. This method modifies the original list by appending all items
- Using extend() method is easy and efficient way to merge two lists or add multiple elements at once
- It will take only one argument

```
In [61]: a = [1,2,3]
         b = [ 4,5,6]
```

```
In [62]: a.extend(b)
```

```
In [63]: a
```

```
Out[63]: [1, 2, 3, 4, 5, 6]
```

```
In [64]: b
```

```
Out[64]: [4, 5, 6]
```

```
In [65]: b.extend(a)
```

```
In [66]: b
```

```
Out[66]: [4, 5, 6, 1, 2, 3, 4, 5, 6]
```

Index()

- The .index() method in Python is used to find the position of the first occurrence of a specified element in a list. If the element is not found, it raises a ValueError

```
In [67]: print(a)
         print(b)
```

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

```
In [68]: print(len(a))
         print(len(b))
```

```
6
9
```

```
In [69]: a.index(2)
```

```
Out[69]: 1
```

```
In [70]: a.index(5)
```

```
Out[70]: 4
```

```
In [71]: b.index(4)
```

```
Out[71]: 0
```

Insert()

- The list.insert() method in Python is used to insert an element at a specific position in a list without replacing existing elements
- It adds the before the index value

In [72]:
```
print(a)
print(b)
```
```
[1, 2, 3, 4, 5, 6]
[4, 5, 6, 1, 2, 3, 4, 5, 6]
```

In [73]:
```
b.insert(3,0)
```

In [74]:
```
b
```
Out[74]:
```
[4, 5, 6, 0, 1, 2, 3, 4, 5, 6]
```

In [75]:
```
a.insert(0,0)
```

In [76]:
```
a
```
Out[76]:
```
[0, 1, 2, 3, 4, 5, 6]
```

pop()

- By default, it removes the last item, but we can specify an index to remove a particular element. It directly modifies the original list
- Index Level Elimination not by value

In [77]:
```
print(a)
print(b)
```
```
[0, 1, 2, 3, 4, 5, 6]
[4, 5, 6, 0, 1, 2, 3, 4, 5, 6]
```

In [78]:
```
a.pop()
```
Out[78]:
```
6
```

In [79]:
```
a.pop()
```
Out[79]:
```
5
```

In [80]:
```
b.pop()
```
Out[80]:
```
6
```

In [81]:
```
b.pop()
```
Out[81]:
```
5
```

In [82]:
```
a.pop()
```

Out[82]:
```
4
```

Reverse()

- The reverse() method in Python is used to reverse the order of elements in a list in-place, meaning it modifies the original list without creating a new one

In [89]:
```
print(a)
print(b)
```

```
[0, 1, 2, 3]
[4, 3, 2, 1, 0, 6, 5, 4]
```

In [90]:
```
a.reverse()
a
```

Out[90]:
```
[3, 2, 1, 0]
```

In [91]:
```
b.reverse()
b
```

Out[91]:
```
[4, 5, 6, 0, 1, 2, 3, 4]
```

Sort()

- The sort() method in Python is used to sort the elements of a list in ascending order by default. It modifies the list in place and does not return a new list

In [92]:
```
print(l)
print(l1)
print(l2)
```

```
[]
[5, 20, 25, 'shravani', 2.3, (10+2j)]
[10, 30, 40, 50]
```

In [93]:
```
l1.sort()                # As it sort doen't take string values, int, complex, and
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[93], line 1
----> 1 l1.sort()

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

In [94]:
```
l2.sort()
l2
```

```
Out[94]: [10, 30, 40, 50]

In [95]: l3 = [5, 25, 15, 2, 1, 16, 18]

In [96]: len(l3)

Out[96]: 7

In [97]: type(l3)

Out[97]: list

In [98]: l3.sort()

In [99]: l3          # we got here in acsendibg order which is parameter tuning

Out[99]: [1, 2, 5, 15, 16, 18, 25]

In [100…  l3.sort(reverse = True)     # we got here from descending order which is hyper para

In [101…  l3

Out[101…  [25, 18, 16, 15, 5, 2, 1]

In [103…  l4 = ['a, z, r, f, s, t, u, m, p']

In [105…  l4.sort()
          l4

Out[105…  ['a, z, r, f, s, t, u, m, p']

In [106…  l4.sort(reverse = True)
          l4

Out[106…  ['a, z, r, f, s, t, u, m, p']
```

Slicing in List

```
In [112…  l1

Out[112…  [5, 20, 25, 'shravani', 2.3, (10+2j)]

In [113…  l1[:]

Out[113…  [5, 20, 25, 'shravani', 2.3, (10+2j)]

In [114…  l1[:5]

Out[114…  [5, 20, 25, 'shravani', 2.3]
```

In [115…
```
l1[1:]
```
Out[115…
```
[20, 25, 'shravani', 2.3, (10+2j)]
```
In [116…
```
l1[:-1]
```
Out[116…
```
[5, 20, 25, 'shravani', 2.3]
```
In [117…
```
l1[:-2]
```
Out[117…
```
[5, 20, 25, 'shravani']
```
In [119…
```
l1[:-3]
```
Out[119…
```
[5, 20, 25]
```
In [120…
```
l1[::4]
```
Out[120…
```
[5, 2.3]
```
In [122…
```
l1[::-3]
```
Out[122…
```
[(10+2j), 25]
```

Membership in List

In [200…
```
l1
```
Out[200…
```
[5, 20, 25, 'shravani', 2.3, (10+2j)]
```
In [202…
```
15 in l1
```
Out[202…
```
False
```
In [203…
```
20 in l1
```
Out[203…
```
True
```

Nested Indexing in list

- index inside the index is called Nested index

In [126…
```
l1[3]
```
Out[126…
```
'shravani'
```
In [127…
```
l1[3][0]
```
Out[127…
```
's'
```

In [128…
```
l1[3][3]
```
Out[128…
```
'a'
```

In [135…
```
print(l1[3][0])
print(l1[3][1])
print(l1[3][2])
print(l1[3][3])
print(l1[3][4])
print(l1[3][5])
print(l1[3][6])
print(l1[3][7])
```
```
s
h
r
a
v
a
n
i
```

All / Any in List

The all ( ) Method returns

- True - If all elements in a list are true
- False - If all elements in a list is false

The any ( ) function returns True if any elements in the list is True, If not any ( ) returns False

In [231…
```
l1
```
Out[231…
```
[5, 20, 25, 'shravani', 2.3, (10+2j)]
```

In [232…
```
all(l1)
```
Out[232…
```
True
```

In [233…
```
any(l1)
```
Out[233…
```
True
```

In [234…
```
l4 = [1,3,5]
l4
```
Out[234…
```
[1, 3, 5]
```

In [235…
```
all(l4)
```
Out[235…
```
True
```

In [236…
```
l4.append(0)
l4
```
Out[236…
```
[1, 3, 5, 0]
```
In [237…
```
all(l4)
```
Out[237…
```
False
```

2) TUPLE

- Tuples are defined using parentheses ().
- A tuple is an ordered, immutable collection of elements in Python.
- Ordered → Elements have a fixed position (index).
- Immutable → Once created, you cannot change, add, or remove elements.
- Can store mixed data types (integers, strings, lists, etc.)

In [164…
```
t = ()
```
In [165…
```
type(t)
```
Out[165…
```
tuple
```
In [166…
```
len(t)
```
Out[166…
```
0
```
In [188…
```
t = 'shravani', 'shaanu', 15, 2.5, 5+ 15j, False, True, 2.5, 'shaanu'
t
```
Out[188…
```
('shravani', 'shaanu', 15, 2.5, (5+15j), False, True, 2.5, 'shaanu')
```

Tuple has only two functions

- Count Function
- Index Function

Count()

In [189…
```
t
```
Out[189…
```
('shravani', 'shaanu', 15, 2.5, (5+15j), False, True, 2.5, 'shaanu')
```

In [190…
```
len(t)
```
Out[190…
```
9
```
In [191…
```
t.count('shaanu')
```
Out[191…
```
2
```
In [192…
```
t.count(2.5)
```
Out[192…
```
2
```
In [193…
```
t.count(15)
```
Out[193…
```
1
```

Index()

In [194…
```
t.index(15)
```
Out[194…
```
2
```
In [195…
```
t.index('shaanu')
```
Out[195…
```
1
```
In [197…
```
t.index('shravani')
```
Out[197…
```
0
```
In [ ]: