

Unit 3: Python complex Data types

Lecture -1

Today's Target

- Creating lists
- Accessing elements in list
- Insert element in list using append(), insert(), extend()
- AKTU PYQs

By PRAGYA RAJVANSHI
B.Tech, M.Tech(C.S.E)

The list is a sequence data type which is used to store the collection of data

```
1 Var = ["hello", "for", "gateway"]  
2 print(Var)
```



input

```
['hello', 'for', 'gateway']
```

Creating a List in Python

- Lists in Python can be created by just placing the sequence inside the square brackets[]. .
- list doesn't need a built-in function for its creation of a list. .
- *the list may contain mutable elements*

```
List = []
print("Blank List: ")
print(List)
List = [10, 20, 14]
print("\nList of numbers: ")
print(List)
List = ["GATEWAY", "CLASSES", "HELLO"]
print("\nList Items: ")
print(List[0])
print(List[2])
```

```
Blank List:
[]
```

```
List of numbers:
[10, 20, 14]
```

```
List Items:
GATEWAY
HELLO
```

Creating a list with multiple distinct or duplicate elements

- A list may contain duplicate values with their distinct positions and hence, multiple distinct or duplicate values can be passed as a sequence at the time of list creation.

```
List = [1, 2, 4, 4, 3, 3, 3, 6, 5]
print("\nList with the use of Numbers: ")
print(List)
List = [1, 2, 'Geeks', 4, 'For', 6, 'Geeks']
print("\nList with the use of Mixed Values: ")
print(List)
```

Creating a list with multiple distinct or duplicate elements

```
1 2  
List with the use of Numbers:
```

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

```
List with the use of Mixed Values:
```

```
[1, 2, 'Geeks', 4, 'For', 6, 'Geeks']
```

Accessing elements from the List

- In order to access the list items refer to the index number. Use the index operator [] to access an item in a list. The index must be an integer. Nested lists are accessed using nested indexing. .

```
List = ["HI", "For", "HELLO"]
print("Accessing a element from the list")
print(List[0])
print(List[2])
```

```
Accessing a element from the list
HI
HELLO
```

Accessing elements from a multi-dimensional list

```

1  # Creating a Multi-Dimensional List
2  # (By Nesting a List inside a List)
3  List = [['HELLO', 'For'], ['BYE']]
4  # accessing an element from the
5  # Multi-Dimensional List using
6  # index number
7  print("Accessing a element from a Multi-Dimensional list")
8  print(List[0][1])
9  print(List[1][0])
10

```

input

```

Accessing a element from a Multi-Dimensional list
For
BYE

```

Indexing means referring to an element of an iterable by its position within the iterable

```
1 List = ["hello", "For", "hellp"]
2 print("\nList Items: ")
3 print(List[0])
4 print(List[2])
```

```
List Items:
hello
hellp
```


In Python, negative sequence indexes represent positions from the end of the array.

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

```
1 List = [1, 2, 'HELLO', 4, 'For', 6, 'BYE']  
2 print("Accessing element using negative indexing")  
3 print(List[-1])  
4 print(List[-3])
```

input
Accessing element using negative indexing
BYE
For

Getting the size of Python list

Python len() is used to get the length of the list..

```
1 List1 = []  
2 print(len(List1))  
3 List2 = [10, 20, 14]  
4 print(len(List2))  
5
```

0
3

Method 1: Using append() method

- Elements can be added to the List by using the built-in append() function. Only one element at a time can be added to the list by using the append() method, for the addition of multiple elements with the append() method, loops are used. Tuples can also be added to the list with the use of the append method because tuples are immutable. Unlike Sets, Lists can also be added to the existing list with the use of the append() method.

Adding Elements to a Python List(append())

Syntax: `list.append(item)`

Parameters:

item: an item to be added at the end of the list, The parameter is mandatory and omitting it can give an error.

Returns: The method doesn't return any value

Adding Elements to a Python List `append()`

```
1  # Python program to demonstrate
2  # Addition of elements in a List
3  # Creating a List
4  List = []
5  print("Initial blank List: ")
6  print(List)
7  # Addition of Elements in the List
8  List.append(1)
9  List.append(2)
10 List.append(4)
11 print("\nList after Addition of Three elements: ")
12 print(List)
13
```

Adding Elements to a Python List `append()`

```
13 # Adding elements to the List using Iterator
14 for i in range(1, 4):
15     List.append(i)
16 print("\nList after Addition of elements from 1-3: ")
17 print(List)
18
19 # Adding Tuples to the List
20 List.append((5, 6))
21 print("\nList after Addition of a Tuple: ")
22 print(List)
23
24 # Addition of List to a List
25 List2 = ['For', 'Geeks']
26 List.append(List2)
27 print("\nList after Addition of a List: ")
28 print(List)
```

Adding Elements to a Python List append()

Initial blank List:

```
[]
```

List after Addition of Three elements:

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

List after Addition of elements from 1-3:

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

List after Addition of a Tuple:

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

List after Addition of a List:

```
[1, 2, 4, 1, 2, 3, (5, 6), ['For', 'Geeks']]
```

Method 2: Using insert() method

append() method only works for the addition of elements at the end of the List, for the addition of elements at the desired position, insert() method is used. Unlike append() which takes only one argument, the insert() method requires two arguments(position, value).

```
# Python program to demonstrate Addition of elements in a List  
# Creating a List  
List = [1,2,3,4]  
print("Initial List: ")  
print(List)  
# Addition of Element at specific Position  
List.insert(3, 12)  
List.insert(0, 'hello')  
print("\nList after performing Insert Operation: ")  
print(List)
```


Method 2: Using insert() method

Initial List:

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

List after performing Insert Operation:

```
['hello', 1, 2, 3, 12, 4]
```

Method 2: Using insert() method

the syntax of the insert() method is

```
list.insert(index, element)
```

Here, element is inserted to the list at the indexth index. All the elements after elem are shifted to the right.).

insert() Parameters

The insert() method takes two parameters:

index - the index where the element needs to be inserted

element - this is the element to be inserted in the list

Method 3: Using extend() method

Other than append() and insert() methods, there's one more method for the Addition of elements, extend(), this method is used to add multiple elements at the same time at the end of the list

```
# Creating a List
List = [1, 2, 3, 4]
print("Initial List: ")
print(List)
# Addition of multiple elements
# to the List at the end
List.extend([8, 'bye', 'Always'])
print("\nList after performing Extend Operation: ")
print(List)
```

Method 3: Using extend() method

Syntax: `list.extend(iterable)`

Parameters:

iterable: Any iterable (list, set, tuple, etc.)

Returns: None

Method 3: Using extend() method

```
Initial List:
```

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

```
List after performing Extend Operation:
```

```
[1, 2, 3, 4, 8, 'bye', 'Always']
```

difference between-

basis for Comparison	Append()	Extend()
Purpose	To add a single entry to the end of a list, use the append() function.	To add additional elements or an iterable to the end of a list, use the extend() function.
Input	accepts only one input element.	accepts as input an iterable (such as a list or tuple).
Operation	The append() function adds the full input to the list as a single item.	extend() adds each item to the list independently after iterating through each one in the input.
Efficiency	Since append() only executes one operation, it is typically quicker and more effective than extend().	When adding elements from numerous iterables or with huge inputs, extend() could take longer.

Unit 3: Python complex Data types

Lecture -2

Today's Target

- Lists
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

Reverse a list

- **Python List reverse()** is an inbuilt method in the Python programming language that reverses objects of the List in place i.e. it doesn't use any extra space but it just modifies the original list.
- **Syntax:** *list_name.reverse()*
- **Parameters:** *There are no parameters.*
- **Returns:** *The reverse() method does not return any value but reverses the given object from the list.*

Reverse a list

```
1 list1 = [1, 2, 3, 4, 1, 2, 6]
2 list1.reverse()
3 print(list1)
4 list2 = ['a', 'b', 'c', 'd', 'a', 'a']
5 list2.reverse()
6 print(list2)
```

input

```
[6, 2, 1, 4, 3, 2, 1]
['a', 'a', 'd', 'c', 'b', 'a']
```

Error in reverse() Method

When anything other than list is used in place of list, then it returns an AttributeError

```
1 # error when string is used in place of list
2 string = "abgedge"
3 string.reverse()
4 print(string)
5
```

input

```
Traceback (most recent call last):
  File "/home/main.py", line 3, in <module>
    string.reverse()
AttributeError: 'str' object has no attribute 'reverse'
```

Error in reverse() Method

When anything other than list is used in place of list, then it returns an AttributeError

```
1 # error when string is used in place of list
2 string = "abgedge"
3 string.reverse()
4 print(string)
5
```

input

```
Traceback (most recent call last):
  File "/home/main.py", line 3, in <module>
    string.reverse()
AttributeError: 'str' object has no attribute 'reverse'
```

Reverse a List using the Slicing Operator

In this example, the `[::-1]` slicing operator creates a new list which is the reverse of the `my_list`

```
1 my_list = [1, 2, 3, 4, 5]
2 reversed_list = my_list[::-1]
3 print(reversed_list)
```

[5, 4, 3, 2, 1]

Reversing a sublist using Slicing

In this example, we are reversing a sublist from index 1 to 3 using `[::-1]` operator.

```
1 my_list = [1, 2, 3, 4, 5]
2 print('Original list:', my_list)
3 my_list[1:4] = my_list[1:4][::-1]
4 print('Reversed sublist:', my_list)
5
6
```

Original list: [1, 2, 3, 4, 5]
Reversed sublist: [1, 4, 3, 2, 5]

Accessing Elements in Reversed Order

```
1 my_list = [1, 2, 3, 4, 5]
2 for element in reversed(my_list):
3     print(element)
```

5
4
3
2
1

Reversing a list of mixed DataTypes

```
1 my_list = [1, 'apple', 2.5, True]
2 print('Original list:', my_list)
3 my_list.reverse()
4 print('Reversed list:', my_list)
5
```

Original list: [1, 'apple', 2.5, True]
Reversed list: [True, 2.5, 'apple', 1]

In Python, list slicing is a common practice and it is the most used technique for programmers to solve efficient problems. Consider a Python list, in order to access a range of elements in a list, you need to slice a list. One way to do this is to use the simple slicing operator i.e. colon(:). With this operator, one can specify where to start the slicing, where to end, and specify the step. List slicing returns a new list from the existing list.

Python List Slicing Syntax

The format for list slicing is of Python List Slicing is as follows:

`Lst[Initial : End : IndexJump]`

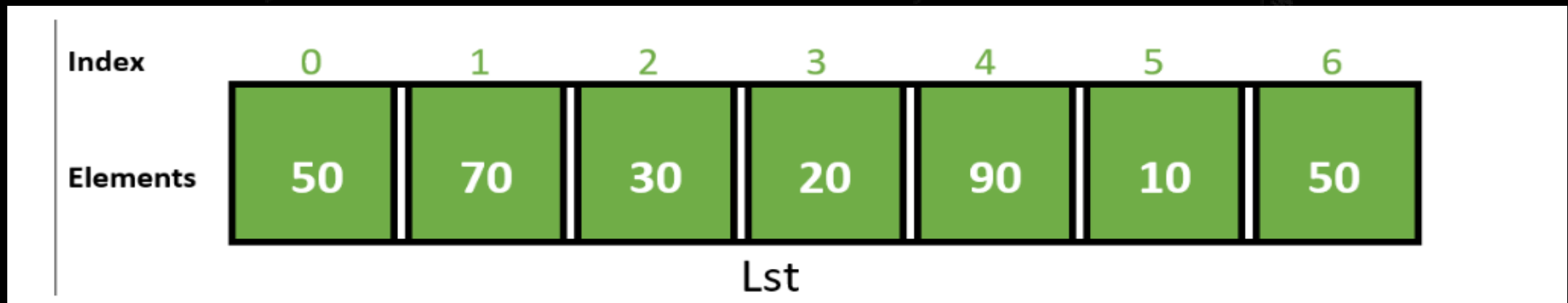
If Lst is a list, then the above expression returns the portion of the list from index Initial to index End, at a step size IndexJump.

Indexing in Python List

Indexing is a technique for accessing the elements of a Python List. There are various ways by which we can access an element of a list.

Positive Indexes

In the case of Positive Indexing, the first element of the list has the index number 0, and the last element of the list has the index number N-1, where N is the total number of elements in the list (size of the list).



Python List Slicing

```
1 Lst = [50, 70, 30, 20, 90, 10, 50]  
2 print(Lst[::])
```



input

```
[50, 70, 30, 20, 90, 10, 50]
```

Negative Indexes

The below diagram illustrates a list along with its negative indexes. Index -1 represents the last element and -N represents the first element of the list, where N is the length of the list.



Lst

Negative Indexing of a Python List

Python List Slicing

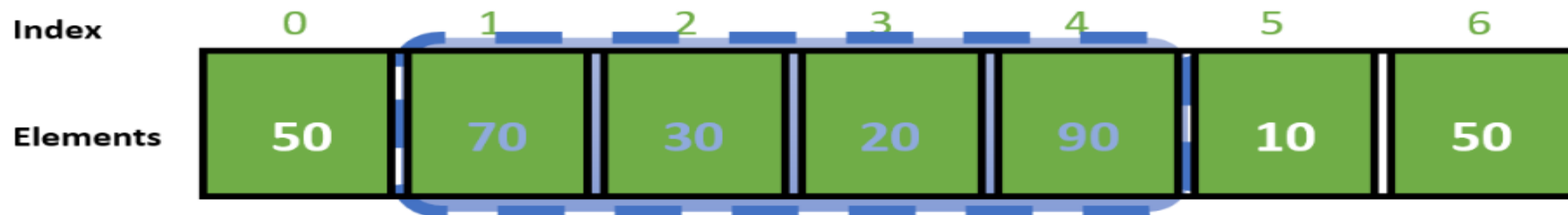
```
1 # Initialize List
2 Lst = [50, 70, 30, 20, 90, 10, 50]
3 # Display List
4 print(Lst[-7::1])
5
```



inp

```
[50, 70, 30, 20, 90, 10, 50]
```

Slicing



`Lst[1:5]`

Python List Slicing

Slicing

```
1 Lst = [50, 70, 30, 20, 90, 10, 50]
2 print(Lst[1:5])
3
```



```
[70, 30, 20, 90]
```

Slicing

```
1 Lst = [50, 70, 30, 20, 90, 10, 50]
2 print(Lst[1:5])
3
```



```
[70, 30, 20, 90]
```

Slicing

```
1  # Initialize list
2  List = [1, 2, 3, 4, 5, 6, 7, 8, 9]
3
4  # Show original list
5  print("Original List:\n", List)
6
7  print("\nSliced Lists: ")
8
9  # Display sliced list
10 print(List[3:9:2])
11
12 # Display sliced list
13 print(List[::2])
14
15 # Display sliced list
16 print(List[::])
17
```



```
Original List:
```

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

```
Sliced Lists:
```

```
[4, 6, 8]
```

```
[1, 3, 5, 7, 9]
```

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

Reversed list can be generated by using a negative integer as the IndexJump argument. Leaving the Initial and End as blank. We need to choose the Initial and End values according to a reversed list if the IndexJump value is negative

```
# Initialize List
List = ['HI', 4, 'BYE !']

# Show original List
print("Original List:\n", List)

print("\nSliced Lists: ")

# Display sliced List
print(List[::-1])

# Display sliced List
print(List[::-3])

# Display sliced List
print(List[:1:-2])
```

Original List:

```
['HI', 4, 'BYE !']
```

Sliced Lists:

```
['BYE !', 4, 'HI']
```

```
['BYE !']
```

```
['BYE !']
```

Slicing

```
# Initialize List
List = [-999, 'G4G', 1706256, '^_^', 3.1496]
# Show original List
print("Original List:\n", List)
print("\nSliced Lists: ")
# Display sliced List
print(List[10::2])
# Display sliced List
print(List[1:1:1])
# Display sliced List
print(List[-1:-1:-1])
# Display sliced List
print(List[:0:])
```

Original List:

```
[-999, 'G4G', 1706256, '^_^', 3.1496]
```

Sliced Lists:

```
[]
```

```
[]
```

```
[]
```

```
[]
```

List slicing can be used to modify lists or even delete elements from a list.

```
# Initialize List
List = [-999, 'G4G', 1706256, 3.1496, '^_^']
# Show original List
print("Original List:\n", List)
print("\nSliced Lists: ")
# Modified List
List[2:4] = ['Geeks', 'for', 'Geeks', '!']
# Display sliced List
print(List)
# Modified List
List[:6] = []
# Display sliced List
print(List)
```

```
Original List:
```

```
[-999, 'G4G', 1706256, 3.1496, '^_^']
```

```
Sliced Lists:
```

```
[-999, 'G4G', 'Geeks', 'for', 'Geeks', '!', '^_^']
```

```
['^_^']
```

By concatenating sliced lists, a new list can be created or even a pre-existing list can be modified

```
# Initialize List
List = [1, 2, 3, 4, 5, 6, 7, 8, 9]
# Show original List
print("Original List:\n", List)
print("\nSliced Lists: ")
# Creating new List
newList = List[:3]+List[7:]
# Display sliced List
print(newList)
# Changing existing List
List = List[::2]+List[1::2]
# Display sliced List
print(List)
```


Original List:

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

Sliced Lists:

```
[1, 2, 3, 8, 9]
```

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

How elements are deleted from list(remove())

Definition and Use of Python list remove() Function

- The list remove() function in [Python](#) removes the first occurrence of a given item from the list. Here, we will see how we can remove elements from the Python list by remove function.
- It only takes one argument that is the element you want to remove and if that element is not present in the list, it gives **ValueError**.
- It is very useful in removing incorrect values from a list, without affecting the rest of the list.

How elements are deleted from list(remove())

```
1 # creating a list
2 number = [1, 2, 3, 4, 5, 6, 23.4, 7, 8]
3 # removing 23.4 from list
4 number.remove(23.4)
5 # printing new list
6 print(number)
```



input

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

Remove an element from the list in Python

```
1  # the first occurrence of 1 is removed from the list
2  list1 = [ 1, 2, 1, 1, 4, 5 ]
3  list1.remove(1)
4  print(list1)
5  list2 = [ 'a', 'b', 'c', 'd' ]
6  list2.remove('a')
7  print(list2)
8
```



input

```
[2, 1, 1, 4, 5]
['b', 'c', 'd']
```

Deleting Element that doesn't Exist

```
1 list2 = [ 'a', 'b', 'c', 'd' ]  
2 list2.remove('e')  
3 print(list2)
```



input

```
Traceback (most recent call last):  
  File "/home/main.py", line 2, in <module>  
    list2.remove('e')  
ValueError: list.remove(x): x not in list
```

Remove all Occurrences of a value from a List

```
1 list1 = [1, 2, 3, 4, 1, 1, 1, 4, 5]
2 # looping till all 1's are removed
3 while (list1.count(1)):
4     list1.remove(1)
5 print(list1)
6
```

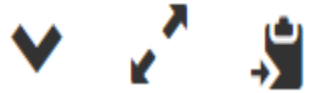


input

[2, 3, 4, 4, 5]

Remove Duplicates from List in Python

```
1 list2 = [ 'a', 'b', 'c', 'd', 'd', 'e', 'd' ]  
2 list2.remove('d')  
3 print(list2)  
4
```



input

```
['a', 'b', 'c', 'd', 'e', 'd']
```

Given a list, remove all the 2's from the list using in keyword

```
1 mylist = [1, 2, 3, 2, 2]
2 # Looping till all 2's are removed
3 while 2 in mylist:
4     mylist.remove(2)
5 print(mylist)
6
```



input

[1, 3]

Removing a nested list element from a list

```
1 data = [[1, 2], [3, 4], [5, 6]]  
2 data.remove([3, 4])  
3 print(data) |  
4
```



input

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

Removing an Element by Value from a List

```
1 my_list = [1, 2, 3, 4, 5]
2 # Remove element '3' from the list
3 if 3 in my_list:
4     my_list.remove(3)
5 print("Updated list:", my_list)
```



input

Updated list: [1, 2, 4, 5]

Python List pop() Method

Python list pop() function removes elements at a specific index

```
1  # create a list
2  fruits = ["apple", "mango", "cherry"]
3  # remove last item from the list
4  fruits.pop()
5  # print the list
6  print(fruits)
```



input

```
['apple', 'mango']
```

Python List pop() Method

Definition of Python List pop method()

pop() function removes and returns the value at a specific index. It is an inbuilt function of Python.

It can be used with and without parameters; without a parameter list pop() returns and removes the last value from the list by default, but when given an index value as a parameter, it only returns and removes the element at that index.

Python List pop() Method

Exception: pop() method raises **IndexError** when the index is out of range.

Return: Returns The last value or the given index value from the list.

Python List pop method Syntax ()

list_name.pop(index)

Parameters:

index (*optional*) – The value at index is popped out and removed. If the index is not given, then the last element is popped out and removed.

1. Remove item at last index from list

```
1 my_list = [1, 2, 3, 4]
2 print("Popped element:", my_list.pop())
3 print("List after pop():", my_list)
4
```

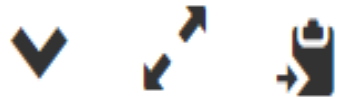


input

```
Popped element: 4
List after pop(): [1, 2, 3]
```

2. Remove Item at specific index from List

```
1 my_list = [1, 2, 3, 4, 5, 6]  
2 print(my_list.pop(3), my_list)
```



input

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

3. Index Error: pop index out of range

```
1 my_list = [ 1, 2, 3, 4, 5, 6 ]  
2 print(my_list.pop(8))
```



input

Traceback (most recent call last):

File "/home/main.py", line 3, in <module>

print(my_list.pop(8))

IndexError: pop index out of range

4.Remove Item at Negative index from Python List

```
1 my_list = [1, 2, 3, 4, 5, 6]
2 popped_item = my_list.pop(-2)
3 print("New list", my_list)
4 print("Popped Item", popped_item)
5
```



input

New list [1, 2, 3, 4, 6]

Popped Item 5

Remove Item from List using Del()

We can Remove Elements from List using Del(). The Python del statement is not a function of List. Items of the list can be deleted using the del statement by specifying the index of the item (element) to be deleted.

```
1 lst = ['Iris', 'Orchids', 'Rose', 'Lavender',
2       'Lily', 'Carnations']
3 print("Original List is :", lst)
4
5 # using del statement
6 # to delete item (Orchids at index 1)
7 # from the list
8 del lst[1]
9 print("After deleting the item :", lst)
10
```

input

```
Original List is : ['Iris', 'Orchids', 'Rose', 'Lavender', 'Lily', 'Carnations']
After deleting the item : ['Iris', 'Rose', 'Lavender', 'Lily', 'Carnations']
```

MODULE 3:conditional and loops

Lecture -3

Today's Target

- List operation
- List comprehension
- Tuples and operations
- AKTU PYQs

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

clear()

- Python List clear() method removes all items from the List.

```
1 lis = [1, 2, 3]
2 lis.clear()
3 print(lis)
4
```

[]

Clearing 2D list using list clear() Method

- we are creating a 2D list, and we are clearing the list at all indexes of the 2d list and printing it.

```
1  # Defining a 2-d List
2  lis = [[0, 0, 1],
3         [0, 1, 0],
4         [0, 1, 1]]
5  # clearing the List
6  lis.clear()
7  print(lis)
```

input

```
[]
```

Python List count() method

- Python List count() method returns the count of how many times a given object occurs in a list using Python.

```
1 fruits = ["Apple", "Mango", "Banana", "Cherry", "Papaya"]  
2 print(fruits.count("Apple"))  
3  
4
```



input

1

List count() method Syntax

syntax: `list_name.count(object)`

Parameters:

object: is the item whose count is to be returned.

Returns: Returns the count of how many times object occurs in the list.

Exception:

TypeError: Raises TypeError If more than 1 parameter is passed in count() method.

Python List count() method

```
1 Rand = [1,3,2,4,1,3,2,4,5,2,3]  
2 print(Rand.count(2))
```

3

Exceptions While Using Python list count() method

List count() in Python raises TypeError when more than 1 parameter is passed

```
1 list1 = [1, 1, 1, 2, 3, 2, 1]
2 print(list1.count(1, 2))
3
```

```
Traceback (most recent call last):
  File "/home/main.py", line 2, in <module>
    print(list1.count(1, 2))
TypeError: list.count() takes exactly one argument (2 given)
```

```
...Program finished with exit code 1
Press ENTER to exit console.□
```

python list sort() method sorts the elements of a list. In this article, we will see how to sort a list in Python using sort() function.

Python List sort() Method Syntax

```
List_name.sort(reverse=True/False, key=myFunc)
```

Parameters:

reverse (Optional): for reverse=True, it will sort the list descending. Default is reverse=False

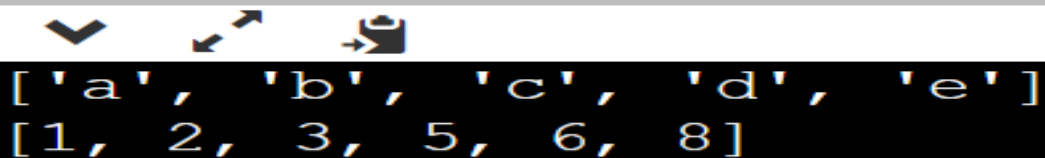
key (Optional) – A function to specify the sorting criteria(s)

sort()

```

1  # using list alphabets
2  alphabets = ['a', 'e', 'd', 'c', 'b']
3  # sorting list
4  alphabets.sort()
5  # printing sorted list
6  print(alphabets)
7  # using a list random numbers
8  random_numbers = [2, 5, 6, 1, 8, 3]
9  # sorting list
10 random_numbers.sort()
11 # printing sorted number list
12 print(random_numbers)
13

```



```

['a', 'b', 'c', 'd', 'e']
[1, 2, 3, 5, 6, 8]

```

Sort a List of Numbers in Ascending Order

```
1 numbers = [1, 3, 4, 2]
2 # Sorting List of Integers in ascending
3 numbers.sort()
4 print(numbers)
```

input

[1, 2, 3, 4]

Sort a List of Numbers in Ascending Order

```
1 strs = ["geeks", "code", "ide", "practice"]  
2 strs.sort()  
3 print(strs)
```



input

```
['code', 'geeks', 'ide', 'practice']
```

Sort a List in Python in Descending Order

Here, we are sorting the list of numbers in Descending order, the same will be for alphabets(Z-A, z-a). To do this we need to pass **reverse=True**, this will sort numbers or the alphabet in descending order

```
1 numbers = [1, 3, 4, 2]
2 numbers.sort(reverse=True)
3 print(numbers)
```

[4, 3, 2, 1]

...Program finished with exit code 0
Press ENTER to exit console. ☐

Python min() Function

Python min() function returns the smallest of the values or the smallest item in an iterable passed as its parameter.

```
2 numbers = [23,25,65,21,98]
3 #printing minimum
4 print(min(numbers))
5
```

Python max() Function

Python max() function returns the largest of the values item in an iterable passed as its parameter.

```
1  #creating a list
2  numbers = [23,25,65,21,98]
3  #printing minimum
4  print(max(numbers))
5
```

98

list of built-in list methods

S.no	Method	Description
1	<u>append()</u>	Used for adding elements to the end of the List.
2	<u>copy()</u>	It returns a shallow copy of a list
3	<u>clear()</u>	This method is used for removing all items from the list.
4	<u>count()</u>	These methods count the elements.
5	<u>extend()</u>	Adds each element of an iterable to the end of the List

list of built-in list methods

S.no	Method	Description
6	<u>index()</u>	Returns the lowest index where the element appears.
7	<u>insert()</u>	Inserts a given element at a given index in a list.
8	<u>pop()</u>	Removes and returns the last value from the List or the given index value.
9	<u>remove()</u>	Removes a given object from the List.
10	<u>reverse()</u>	Reverses objects of the List in place.

list of built-in list methods

S.no	Method	Description
11	<u>sort()</u>	Sort a List in ascending, descending, or user-defined order
12	<u>min()</u>	Calculates the minimum of all the elements of the List
13	<u>max()</u>	Calculates the maximum of all the elements of the List

- List comprehension is used to create a new list from existing sequence.
- It is a tool for transforming a given list into another list.
- Using list comprehension, we can replace the loop with a single expression that produces the same result
- Syntax: `newList = [expression(element) for element in oldList if condition]`
- Parameter:
 - expression: Represents the operation you want to execute on every item within the iterable.
 - element: The term “variable” refers to each value taken from the iterable.
 - iterable: specify the sequence of elements you want to iterate through.(e.g., a list, tuple, or string).
 - condition: (Optional) A filter helps decide whether or not an element should be added to the new list.
- Return:The return value of a list comprehension is a new list containing the modified elements that satisfy the given criteria.

LIST COMPREHENSION

```
1 numbers = [1, 2, 3, 4, 5]
2 squared = [x ** 2 for x in numbers]
3 print(squared)
4
```

[1, 4, 9, 16, 25]

input

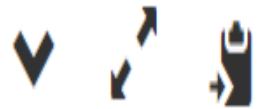
WITHOUT LIST COMPREHENSION

```
1 numbers = [1, 2, 3, 4, 5]
2 for i in range(0, len(numbers)):
3     numbers[i] = numbers[i] ** 2
4 print(numbers)
```

[1, 4, 9, 16, 25]

Even list using List Comprehension

```
1 list = [i for i in range(11) if i % 2 == 0]  
2 print(list)
```



input

```
[0, 2, 4, 6, 8, 10]
```

Python Tuple is a collection of objects separated by commas. In some ways, a tuple is similar to a Python list in terms of indexing, nested objects, and repetition but the main difference between both is Python tuple is immutable, unlike the Python list which is mutable.

Creating Python Tuples

There are various ways by which you can create a tuple in [Python](#). They are as follows:

- Using round brackets
- With one item
- Tuple Constructor

Create Tuples using Round Brackets ()

```
1 var = ("Geeks", "for", "Geeks")
2 print(var)
3 print(type(var))
```

input

```
('Geeks', 'for', 'Geeks')
<class 'tuple'>
```

Create a Tuple With One Item

```
1 mytuple = ("Geeks",)
2 print(type(mytuple))
3 #NOT a tuple
4 mytuple = ("Geeks")
5 print(type(mytuple))
```

```
<class 'tuple'>
<class 'str'>
```

Tuple Constructor in Python

```
1 tuple_constructor = tuple(("dsa", "development", "deep learning"))
2 print(tuple_constructor)
```

input

('dsa', 'development', 'deep learning')

What is Immutable in Tuples?

- Tuples in Python are similar to [Python lists](#) but not entirely. Tuples are immutable and ordered and allow duplicate values. Some Characteristics of Tuples in Python.
- We can find items in a tuple since finding any item does not make changes in the tuple.
- One cannot add items to a tuple once it is created.
- Tuples cannot be appended or extended.
- We cannot remove items from a tuple once it is created.

What is Immutable in Tuples?

```

1 mytuple = (1, 2, 3, 4, 5)
2 # tuples are indexed
3 print(mytuple[1])
4 print(mytuple[4])
5 # tuples contain duplicate elements
6 mytuple = (1, 2, 3, 4, 2, 3)
7 print(mytuple)
8 # adding an element
9 mytuple[1] = 100
10 print(mytuple)
11
12

```

What is Immutable in Tuples?

```
2
5
(1, 2, 3, 4, 2, 3)
Traceback (most recent call last):
  File "/home/main.py", line 12, in <module>
    mytuple[1] = 100
TypeError: 'tuple' object does not support item assignment
```

```
...Program finished with exit code 1
Press ENTER to exit console.□
```

Accessing Values in Python Tuples

- Tuples in Python provide two ways by which we can access the elements of a tuple.
- Using a positive index
- Using a negative index

Python Access Tuple using a Positive Index

```
1 var = ("hi", "for", "bye")
2 print("Value in Var[0] = ", var[0])
3 print("Value in Var[1] = ", var[1])
4 print("Value in Var[2] = ", var[2])
5
```



input

```
Value in Var[0] = hi
Value in Var[1] = for
Value in Var[2] = bye
```


Access Tuple using Negative Index

```
1 var = (1, 2, 3)
2 print("Value in Var[-1] = ", var[-1])
3 print("Value in Var[-2] = ", var[-2])
4 print("Value in Var[-3] = ", var[-3])
5
```



input

```
Value in Var[-1] = 3
Value in Var[-2] = 2
Value in Var[-3] = 1
```

tuples with different datatype and without parenthesis

```

1  #tuples with different types of data types
2  var = (1, 2, 3.5, "hi")
3  print(var)
4  print(type(var))
5  #tuples also created without parenthesis
6  var1=6,8,9.6
7  print(var1)
8  print(type(var1))
9

```

```

(1, 2, 3.5, 'hi')
<class 'tuple'>
(6, 8, 9.6)
<class 'tuple'>

```

input

Different Operations Related to Tuples

- **Different Operations Related to Tuples**
- Below are the different operations related to tuples in Python:
- Concatenation
- Nesting
- Repetition
- Slicing
- Deleting(not allowed)
- Finding the length
- Multiple Data Types with tuples
- Conversion of lists to tuples
- Tuples in a Loop

Concatenation of Python Tuples

```

1  # Code for concatenating 2 tuples
2  tuple1 = (0, 1, 2, 3)
3  tuple2 = ('python', 'hey')
4  # Concatenating above two
5  print(tuple1 + tuple2)
6

```



input

```
(0, 1, 2, 3, 'python', 'hey')
```

Nesting of Python Tuples

A nested tuple in Python means a tuple inside another tuple.

```
2 tuple1 = (0, 1, 2, 3)
3 tuple2 = ('python', 'geek')
4 tuple3 = (tuple1, tuple2)
5 print(tuple3)
6
```

```
((0, 1, 2, 3), ('python', 'geek'))
```

We can create a tuple of multiple same elements from a single element in that tuple..

```
1 tuple3 = ('python',)*3  
2 print(tuple3)  
3
```

(
'python', 'python', 'python')

Slicing Tuples in Python

Slicing a Python tuple means dividing a tuple into small tuples using the indexing method.

```
1 tuple1 = (0, 1, 2, 3)
2 print(tuple1[1:])
3 print(tuple1[::-1])
4 print(tuple1[2:4])
5
```

```
(1, 2, 3)
(3, 2, 1, 0)
(2, 3)
```

Deleting a Tuple in Python

In this example, we are deleting a tuple using [‘del’ keyword](#). The output will be in the form of error because after deleting the tuple, it will give a NameError.

Note: Remove individual tuple elements is not possible, but we can delete the whole Tuple using Del keyword

```
1 tuple3 = ( 0, 1)
2 del tuple3
3 print(tuple3)
```

input

```
Traceback (most recent call last):
  File "/home/main.py", line 4, in <module>
    print(tuple3)
NameError: name 'tuple3' is not defined. Did you mean: 'tuple'?
```


Finding the Length of a Python Tuple

```
1 tuple2 = ('python', 'bye')  
2 print(len(tuple2))  
3
```

2

...Program finished with exit code 0
Press ENTER to exit console.

Multiple Data Types With Tuple

Tuples in Python are heterogeneous in nature. This means tuples support elements with multiple datatypes

```
2 tuple_obj = ("immutable", True, 23)
3 print(tuple_obj)
4
```

input
('immutable', True, 23)

Converting a List to a Tuple

```

1 # Code for converting a list and a string into a tuple
2 list1 = [0, 1, 2]
3 print(tuple(list1))
4 # string 'python'
5 print(tuple('python'))
6

```

input

(0, 1, 2)

('p', 'y', 't', 'h', 'o', 'n')

Tuples in a Loop

```

1 # python code for creating tuples in a loop
2 tup = ('h',)
3 # Number of time loop runs
4 n = 2
5 for i in range(int(n)):
6     tup = (tup,)
7     print(tup)

```

☐
☐
☐

input

```

(('h',),)
(((('h',),),),)

```

AKTU Full Courses (Paid)
Download **Gateway Classes** Application
From Google Play store
All Subjects

Link in Description

Thank You