LIST MANIPULATION

- Creating and accessing list
- List Operations
- Working with list
- List functions and Methods

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR &
SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

What is List?

- Is a container that are used to store a list of values of any type
- List is mutable type i.e. we can change value in place without creating a fresh list
- List is a type of sequence like tuples and string but in differs them in the way that lists are mutable but string and tuples are immutable

Creating and Accessing List

- List are created using SQUARE BRACKETS ([ ])
- Some of the examples of lists are :
  - [ ] # empty list
  - [1,2,3] # list of integers
  - [10,20,13.75,100.5,90[# list of integers and float
  - [“red”,”green”,”blue”] # list of string
  - [“E001”,”Rakesh”,1,90000.5] # list of mixed value
  - [‘A’, ‘B’, ‘C’] # list of characters

Creating and Accessing List

- To create a list the following syntax we need to follow:
  - **ListName = [ ] Or**
  - **ListName = [value1, value2,…..]**
- For example
  - **Family = [“father”,”mother”,”bro”,”sis”]**
  - **Student = [1,”Aman”,”XI”,3150]**
- The above construct is known as **list display construct**
- **Consider more examples**
  - **EMPTY LIST**
    - **L = [ ] Or**
    - **L = list()**

Creating and Accessing List

- **LONG LIST**
  - **L = [1,2,3,44,55,66,77,88,99,4,3,5,6,7,88,100,300 12,13,14,56,78]**
- **NESTED LIST**
  - **L = [1,2,4,[100,200,300], 20]**
  - *The above code will create List L with 5 elements because it will count [100,200,300] as one element. Now L[3] is list of 3 elements*
  - *To print if we write : L[1] it will display 2 and to print 200 we have to write L[3][1] i.e. of 3rd index print 2nd value*

Creating list from existing sequences

- **We can use the following syntax:**
  - ListName = list(sequence)
- **For example (with string)**
  - L1 = list('welcome')
  - >>>L1
  - ['w','e','l','c','o','m','e']
  - **With tuple**
  - T = ('A','B','C','D')
  - L1 = list(T)
  - >>>L2
  - ['A','B','C','D']

Creating list from existing sequences

- We can also create list of single characters or single digits through keyboard input:

For example:

```
>>> list1 = list(input('Enter list elements'))
>>> list1
```

From the above code whatever values we will enter will be of STRING type. To store list of integers through input in python we can use eval() to convert the list items to integers

```
>>> list1 = eval(input("enter list to be entered"))
>>> print("list is ", list1)
```

eval() function

- eval('5+10') 15

- Y = eval("2*5")
- print(Y) 10

- Num = eval(input("enter any value "))
- print(Num, type(Num))
  - Output will be 20 (if input is 20) and <class 'int'>

- Eval wil not only convert the values to int type but also interpret the value as intended type i.e. if you enter float value it will convert it to float or if it is list or tuple it will convert it.

eval() function

```
>>> list1 = eval(input("Enter values :"))
>>> enter values: [10,20,30]
>>> list1
[10,20,30]
```

Accessing List elements

- Similarity with strings:
  - Just like string, every individual elements of lists are accessed from their index position which is from 0 to length-1 in forward indexing and from -1 to –length in backward indexing.
  - For example
    - **Fruits = ["mango","apple","guaua","pomegranate","cherry"]**
    - In above list items from mango to cherry are 0 to 4 and from cherry to mango will be -1 to -5

0	1	2	3	4
Mango	**Apple**	**Guaua**	**Pomegranate**	**cherry**
-5	-4	-3	-2	-1

Accessing List elements

- If list element is large, it will store the reference in the list and the values will be stored somewhere else.

- List elements are accessed just like string like str[2] means character at 2nd index List1[2] means elements at 2nd index and List1[1:3] means all items between index 1 to 2

Accessing List elements

- Another similarities are :

  Length : the same len() function we can use on list to find out number of elements in the list

  Indexing and Slicing

  Membership operators (in and not in)

  Concatenation and replication operators + and *

- Accessing individual elements:
  - >>> a = [1,2,3,4,5]
  - >>> a[1] # 2
  - >>> a[3] # 4
  - >>> a[-2] # 4
  - >>> a[5] # Error

Difference in String and List

- Although there are many similarities in String and List but there are many difference between them also and the main difference is Strings are immutable whereas Lists are mutable

- **>>> student = [1,'Akash','XIA',3150]**
- **>>> student[3]=6300**
- **>>> student**
- **[1,'Akash','XIA',6300]**

Traversing a list

- Just like String , We can use "for" loop to traverse the list.

```
val = [10,20,30,50,100]
for i in val:
        print(i)
```

Program to print list elements along with their index (both +ve,-ve)

```
val = [10,20,30,50,100]
length = len(val)
for i in range(length):
        print ("At Index ", i," and index ",i-length, 'is :', val[i])
```

Comparing List

- Python allows us to use all the relational operators on list like ==, >=, <=,!=,>,<
- Python will compare individual elements of each list.

```
>>> L1=[1,3,5]
>>> L2=[1,3,5]
>>> L3=[1,5,3]
>>> L1==L2
True
>>> L1==L3
False
>>> L1<L2
False
>>> L1<L3
True
>>> L4=[1,3,6]
>>>L1<L4
True
```

Non-equality comparison in List sequences

Comparison	Result	Reason
[1,2,8,9] <[9,1]	True	1<9
[1,2,8,9]<[1,2,9,1]	True	8<9
[1,2,8,9]<[1,2,9,10]	True	8<9
[1,2,8,9]<[1,2,8,4]	False	9<4 is false

VINOD KUMAR VERMA, PGT(CS), KV OEF KANPUR &
SACHIN BHARDWAJ, PGT(CS), KV NO.1 TEZPUR

List Operations

- **Joining List**
  - Jointing the 2 list is very easy, we can use (+) to join 2 or more list
  - Fruits=[“apple”,”mango”,”grapes”]
  - Veg=[“spinach”,”carrot”,”potato”]
  - Fveg = Fruits + Veg
  - Fveg # [“apple”,”mango”,”grapes”, “spinach”,”carrot”,”potato”]

- *Remember: you can only add list with another list not with int, float, complex, or string type.*

List Operations

- **Repeating of Replicating List**
  - Repeating any list is very easy, we can use (*)
  - Fruits=["apple","mango","grapes"]
  - Fruits*2 #["apple","mango","grapes","apple","mango","grapes"]

- **String slicing**
  - Listname[start:End] # from start index to end-1 index

List Slicing

```
val = [10,20,30,40,1,2,3,100,200]
print(val[0:3])
print(val[3:8])
print(val[-4:-1])
val2 = val[2:-2]
print(val2)
print(val2[0:])
print(val[::-1])
val[5]=101
print(val[-8:-2])
```

List Slicing

```
val = [10,20,30,40,1,2,3,100,200]
print(val[3:50])
print(val[-20:4])
print(val[10:20])
```

Step slicing in List

```
val = [10,20,30,40,1,2,3,100,200]
print(val[0:9:2])        # 10,30,1,3,200
```

List Slicing

```
val = [10,20,30,40,1,2,3,100,200]
print(val[2:9:3])
print(val[::3])
print(val[::-2])
```

Extract two list-slices of a given list of numbers. Display and print the sum of elements of first list slice which contain every other elements between index 5 to 15. program should also display the average of second sliced list which contains every fourth elements of list. The list contains 20 elements


```
val = [10,20,30,40,1,2,3,100,200,10,20,30,11,12,15,17,19,90,77,35]
slice1 = val[5:15:2]
slice2 = val[::4]
sum = 0
for x in slice1:
        print(x,"==",end='')
        sum+=x
print("Sum of slice1 elements are ",sum)

sum=0
for  x in slice2:
        print(x,"==",end='')
        sum+=x
avg = sum / len(slice2)
print("Average of slice 2 =",avg)
```

Using Slicing for List modification

```
items=["One","Two","Three","Four"]
items[0:2]=[1,2]
for i in items:
        print(i,end=' ')
items[0:3]="Fantastic"
print()
for i in items:
        print(i,end=' ')
```

Using Slicing for List modification

```
>>> items=[1,2,3,4]
>>> items[3:]="hello"
>>> items                    # [1,2,3,'h','e','l','l','o'], it will work because
                                string is also a sequence
```

But if you try to assign as:

```
items[3:] = 100 # Error
```

If we pass index which is more than highest range, then Python will use next position after the highest index for e.g.

```
>>> items[100:200]=[111,222]
        Python will append these values to the end of list
>>>items                    [1,2,3,'h','e','l','l','o',111,222]
```

Working with List

- So far we have worked with How the Lists are declared, How to access individual elements of List, How to perform Slicing and so on, Now we are going to deal with basic operation on list like : Appending, Updating, Deleting items from List

- Appending elements to a List : appending means adding new items to the end of list. To perform this we use append() method to add single item to the end of the list.

```
>>> Items = [10,20,30]
>>> Items.append(40)
>>> Items                          # [10,20,30,40]
```

Working with List

- **Updating elements of a list**

  ListName[index] = new value

  Items = [10,20,30,40]

  Items[3] = 100

  Items [10,20,30,100]

- Deleting items from a List: to delete items from list we will use del statement. To delete single item give index, and to delete multiple items use slicing

Working with List

- **Deleting single elements of a list**

  Items = [10,20,30,40,50,60,70]

  del items[2]

  Items [10,20,40,50,60,70]

- **Deleting multiple elements of a list**
  - **Items = [10,20,30,40,50,60,70]**
  - **del items[0:3]**
  - **Items [40,50,60,70]**
- *If we use del items then it will delete all the elements and the list too i.e. now no list with the name items will exist in python*

pop() function to delete list item

- We can also use pop() to remove single elements, not list slices. It not only deletes elements but also returns it. Both del and pop() are same except pop() deletes and return the deleted value.

```
>>>Items = [10,20,30,40,50,60,70]
>>>Items.pop()   # if no index is passed last item will be deleted
70
>>>Items.pop(2)
30
```

We can also store the deleted values by pop() as:

```
N1 = items.pop()
N2 = items.pop(3)
```

Making True Copy of a List

- **Sometimes we need to make a copy of a list and we generally do this by assignment as :**

```
a = [10,20,30]
b = a
```

But this will not make b as duplicate list of a ; rather just like python does it will make b to point to where a is pointing i.e. both will refer to same memory address means b is nothing but 'alias' of a

```
>>> a[2]=100
>>>a                    # 10,20,100
>>>b                    # 10,20,100
```

Making true Independent Copy of a List

- But if we don't want copy like this and we want to modify each list separately, then we have to use list() to create a copy of list and both can be modified without affecting other.

```
>>>a = [10,20,30]
>>>b = list(a)
a[2]=100
>>>a
[10,20,100]
>>>b
[10,20,30]
```

LIST FUNCTIONS AND METHODS

- **Python provides many built-in functions and methods for list manipulation.**
- **The syntax of using list functions is :**
  - **ListObjectName.functionName()**
- **There are many list functions like:**
- **index()**
- **append()**
- **extend()**
- **insert()**
- **pop()**
- **remove()**
- **clear()**
- **count()**
- **reverse()**
- **sort()**

index() method

- This function is used to get the index of first matched item from the list. It returns index value of item to search.
- For example

```
>>> L1=[10,20,30,40,50,20]
>>> L1.index(20)
1                    # item first matched at index 1
```

Note: if we pass any element which is not in the list then index function will return an error: ***ValueError: n is not in the list***

```
>>> L1.index(100)              #Error
```

append() method

- This function is used to add items to the end of the list.
- For example

```
>>> family=["father","mother","bro","sis"]
>>> family.append("Tommy")
>>> family
["father","mother","bro","sis","Tommy"]
```

Note: append() funtion will add the new item but not return any value.
Let us understand this:

```
>>> L1=[10,20,30]
>>> L2=L1.append(40)
>>> L2                                  # Empty will not store any item of L1
>>> L1
[10, 20, 30, 40]
```

extend() method

- This function is also used for adding multiple items. With extend we can add only “list” to any list. Single value cannot be added using extend()
- For example

```
>>> subject1=["physics","chemistry","cs"]
>>> subject2=["english","maths"]
>>> subject1.extend(subject2)
>>> subject1
['physics', 'chemistry', 'cs', 'english', 'maths']
```

Note: here subject1 will add the contents of subject2 in it without effecting subject2

Like append(), extend() also does not return any value.

```
>>> subject3 = subject1.extend(subject2)
>>>subject3                                  # Empty
```

Difference between append() and extend()

- append() allows to add only 1 items to a list, extend() can add multiple items to a list.

```
>>> m1=[1,2,3,4]
>>> m2=[100,200]
>>> m1.append(5)
>>> m1
[1, 2, 3, 4, 5]
>>> m1.append(6,7)

Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    m1.append(6,7)
TypeError: append() takes exactly one argument (2 given)
```

Difference between append() and extend()

```
>>> m1.append([6,7])
>>> m1
[1, 2, 3, 4, 5, [6, 7]]
>>> len(m1)
6
```

Now let us see extend() function

```
>>> m2
[100, 200]
>>> m2.extend(300)

Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    m2.extend(300)
TypeError: 'int' object is not iterable
```

Difference between append() and extend()

```
>>> m2.extend([300,400])
>>> m2
[100, 200, 300, 400]
>>> len(m2)
4
```

insert() method

- This function is used to add elements to list like append() and extend(). However both append() and extend() insert the element at the end of the list. But insert() allows us to add new elements anywhere in the list i.e. at position of our choice.

  ListObject.insert(Position,item)

```
>>> L1=[10,20,30,40,50]
>>> L1
[10, 20, 30, 40, 50]
>>> L1.insert(3,35)
>>> L1
[10, 20, 30, 35, 40, 50]
>>>
```

insert() method

```
>>> L1=[10,20,30,40,50]
>>> L1
[10, 20, 30, 40, 50]
>>> L1.insert(0,5)                  #beginning
>>> L1
[5,10, 20, 30, 40, 50]
>>> L1.insert(len(L1),100)          #last
>>> L1
[5,10, 20, 30, 40, 50,100]
>>> L1.insert(-10,2)
>>> L1
[2,5,10, 20, 30, 40, 50,100]
```

pop() method

- We have read about this function earlier, it is used to remove item from list.

  ListObject.pop([index]) #if index is not passed last item will be deleted

```
>>> L1=[10,20,30,40,50]
>>> L1.pop()
50
>>> val = L1.pop(2)
>>> val
30
>>> L1
[10, 20, 40]
>>>
```

pop() method

- The pop() method raises an exception(run time error) if the list is already empty.

```
>>> L1= [ ]
>>> L1.pop()

Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    L1.pop()
IndexError: pop from empty list
```

remove() method

pop() function is used to remove element whose position is given, but what if you know the value to be removed, but you dont know its index or position in the list? Answer is: remove()

It remove the first occurance of given item from the list and return error if there is no such item in the list. It will not return any value.

```
>>> L1.remove(5)
>>> L1
[1, 3, 7, 9, 11, 3, 7]
>>> L1.remove(3)
>>> L1
[1, 7, 9, 11, 3, 7]
>>> L1.remove(10)

Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    L1.remove(10)
ValueError: list.remove(x): x not in list
```

clear() method

This function removes all teh items from the list and the list becomes empty list.

List.clear()

```
>>> L1=[10,20,30,40,50]
>>> L1.clear()
>>> L1
[ ]
```

Note: unlike 'del listname' statement, clear() will removes only the elements and not the list. After clear() the list object still exists as an empty list

count() method

This function returns the count of the item that you pased as an argument. If the given item is not in the list, it returns zero.

```
>>> L1=[10,20,30,40,20,30,100]
>>> L1.count(20)
2
>>> L1.count(40)
1
>>> L1.count(11)
0
```

reverse() method

This function reverses the items in the list. This is done i place i.e. It wil not create a new list . The syntax to use reverse() is:

```
>>> L1=[10,20,30,40,20,30,100]
>>> L1.reverse()
>>> L1
[100, 30, 20, 40, 30, 20, 10]
>>> L2=[11,22,33]
>>> L3=L2.reverse() #it will not return any value
>>> L3                    # empty
[]
```

sort() method

This function sorts the items of the list, by default increasing order. This is done «in place» i.e. It does not create new list.

```
>>> L1=[10,1,7,20,8,9,2]
>>> L1.sort()
>>> L1
[1, 2, 7, 8, 9, 10, 20]
>>> L2=['g','e','a','c','b','d']
>>> L2.sort()
>>> L2
['a', 'b', 'c', 'd', 'e', 'g']
>>> L1.sort(reverse=True)                    # for descending order
>>>L1
[20, 10, 9, 8, 7, 2, 1]
```