Lists in Python

What is List in Python?

- **List** is a versatile data type and an ordered sequence of items. It is one of the most used datatype in Python and is very flexible.
- All the items in a list do not need to be of the same data type.
- Declaring a list is pretty straight forward.
- Items separated by commas are enclosed within brackets [ ].

Syntax:- List_Variable=[val1,val2,val3…….]

```
List_A=[1,2,3,4,5]#same Data Types
List_B=[1,2.5,3.6,'List',(1,2)]#Different Data Types
print(List_A,List_B) #[1, 2, 3, 4, 5]  [1, 2.5, 3.6, 'List', (1, 2)]
```

- Lists are mutable which means that value of its elements can be **changed** or **altered**

How to create a list?

- In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).

```
List_Empty=[] #Empty List []
List_A=[1,2,3,4,5] #Same Data Types - [1, 2, 3, 4, 5]
List_B=[1,2.5,3.6,'List',(1,2)] #Different or Mixed Data Types
# [1, 2.5, 3.6, 'List', (1, 2)]
```

- Nested List: - a list can even have another list as an item. This is called nested list.

```
List_Nest=['Hello',[1,2,3,4],['a','b']] #Nested Lists
print(List_Nest) #['Hello', [1, 2, 3, 4], ['a', 'b']]
```

How to access elements from a list?

- Similar to strings ,lists can also be sliced and concatenated .
- To access values in lists ,**square brackets []** are used to slice along with the index or indices to get value stored at that **index.**
- We can use the **index operator []** to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.
- Trying to access an element other that this will raise an **IndexError**.
- The index must be an integer. We can't use float or other types, this will result into **TypeError**
- Nested list are accessed using nested indexing.

```
my_list = [1,2,3,4,'h',3.5]
print(my_list[0]) # 1
print(my_list[2]) # 3
print(my_list[4]) # 'h'
#print(my_list[4.0]) # Error! Only integer can be used for indexing
n_list = ["Happy", [2,0,1,5]]# Nested List
print(n_list[0][1])# Nested indexing - a
print(n_list[1][3]) # 5
```

Negative indexing

- Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

```
my_list = [1,2,3,4,'h',3.5]
print(my_list[-1]) # 3.5
print(my_list[-2]) # 'h'
print(my_list[-4]) # 3

n_list = ["Happy", [2,0,1,5]]# Nested List
print(n_list[0][-1])# Nested indexing - y
print(n_list[1][-3]) # 0
```

- Negative and Positive Indexing in Figure shown below

List_S=['H','E','L','L','O']

0	1	2	3	4
H	E	L	L	O
-5	-4	-3	-2	-1

Slicing Operations with range/Slicing Operator

- Slicing can be best considering the index to be between the elements along with slicing operator Colon [:]
- We can access a range of items in a list by using the slicing operator (colon)
- **Syntax : -** List[start:end:step]

  start – starting point of the index value

  end or stop – ending point of the index value

  step- increment of the index values

```
my_list = ['r','g','u','k','t','n','u','z',1,2,3,4,5]
print(my_list[2:5]) # elements 3rd to 5th - ['u', 'k', 't']
print(my_list[:-5]) # elements beginning to 4th - ['r', 'g', 'u', 'k', 't', 'n', 'u', 'z']
print(my_list[5:])  # elements 6th to end - ['n', 'u', 'z', 1, 2, 3, 4, 5]
print(my_list[:])   # elements beginning to end
print(my_list[::])  # elements beginning to end
print(my_list[::2]) # elements beginning to end with step2 - ['r', 'u', 't', 'u', 1, 3, 5]
print(my_list[1::3])# elements 1st to end with step 3 - ['g', 't', 'z', 3]
```

How to change or add elements to a list?

- List are mutable, meaning, their elements can be changed unlike **string** or **tuple**.
- We can use assignment operator (=) to change an item or a range of items.

Syntax : - **list[index]=item**

```
odd = [2, 4, 6, 8]  # mistake values
odd[0] = 1          # change the 1st item
print(odd)          # Output: [1, 4, 6, 8]
odd[1:4] = [3,5,7]  # change 2nd to 4th items
print(odd)          # Output: [1, 3, 5, 7]
```

- We can add one **item** in to a list using **append()** method and add Several Items to a list using **extend()** method.

Syntax:-

list.append(item)

list.extend(items)

```
odd = [1, 3, 5]
odd.append(7) # add single Item
print(odd) # Output: [1, 3, 5, 7]
odd.extend([9, 11, 13]) # add multiple items
print(odd)          # Output: [1, 3, 5, 7, 9, 11, 13]
```

To insert ,Concatenate and Repeat

- We can also use + operator to combine two lists. This is also called concatenation.
- The * operator repeats a list for the given number of times.

Syntax :- list3=**list1+list2**

```
add1=[1,3,9] #list1
add2=[5,7] #list 2
add=add1+add2 #concatenation for list1 and list2
print(add) #[1,3,5,7,9]
```

- we can **insert** one item at a desired location by using the method **insert()** or

  insert multiple items by squeezing it into an **empty slice** of a list.

Syntax : -

List.insert(IndexValue,newValue)

List[empty_slice]=[items]

```
odd = [1,3,5,7,9]
odd.insert(9,11)
print(odd)# Output: [1,3,5,7,9,11]
odd[2:2] = [13, 15] # empty slice operation
print(odd) # Output: [1, 3, 13, 15, 5, 7, 9, 11]
```

How to delete or remove elements from a list?

- We can delete one or more items from a list using the keyword **del**. It can even delete the list entirely.

  Syntax : - **del list**
- We can use **remove()** method to remove the given item or **pop()** method to remove an item at the given index.
- The **pop()** method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).
- We can also use the **clear()** method to empty a list.
- Finally, we can also delete items in a list **by assigning an empty list** to a slice of elements.

```
my_list = [1,2,3,4,5,6]
del my_list[2] # delete one item
print(my_list) # Output: [1,2,4,5,6]
del my_list[1:3] # 2,4
print(my_list)# Output: [1,5,6]
del my_list         # delete entire list
print(my_list) # Error: List not defined
```

```
my_list = [1,2,3,4,5,6,7,8]
my_list.remove(3)
print(my_list)#[1, 2, 4, 5, 6, 7, 8] removed item 3
print(my_list.pop(1))
print(my_list)#[1, 4, 5, 6, 7, 8] removed index 1 item
print(my_list.pop())# 8
print(my_list)#[1,4,5,6,7] removed last item
my_list.clear()
print(my_list) #[] empty list
my_list1 = [9,10,11,12,13,14,15]
my_list1[2:]=[] # assigning an empty list
print(my_list1)#[9,10]
```

- **index()** - Returns the index of the first matched item
  - Syntax:- **list.index(**element**)**
- **count()** - Returns the count of number of items contains in a list
  - Syntax: **- list.count(**element**)**
- **sort()** - Sort items in a list in ascending order
  - Syntax: **- list.sort()**
- **reverse()** - Reverse the order of items in the list
  - Syntax: **- list.reverse()**
- **copy()** - Returns a shallow copy of the list
  - Syntax: **- list.copy()**

```
my_list = [3, 8, 1, 6, 0, 8, 4]
print(my_list.index(8)) # Output: 1
print(my_list.count(8))# Output: 2
my_list.sort()
print(my_list)# Output: [0, 1, 3, 4, 6, 8, 8]
my_list.reverse()
print(my_list) # Output: [8, 8, 6, 4, 3, 1, 0]

new_list = my_list #copying with assignment operator
print(new_list)# Output: [8, 8, 6, 4, 3, 1, 0]
new_list.append('a') # add element to list
print( new_list,my_list)

new_list1 = my_list.copy() #copying with function
print(new_list1)
new_list1.append('b') # add element to list
print( new_list1,my_list)
```

Python List Methods

- They are accessed as **list.method()**

append() - Add an element to the end of the list
extend() - Add all elements of a list to the another list
insert() - Insert an item at the defined index
remove() - Removes an item from the list
pop() - Removes and returns an element at the given index
clear() - Removes all items from the list
index() - Returns the index of the first matched item
count() - Returns the count of number of items passed as an argument
sort() - Sort items in a list in ascending order
reverse() - Reverse the order of items in the list
copy() - Returns a shallow copy of the list

List Comprehension: Elegant way to create new List

- List comprehension is an elegant and concise way to create new list from an existing list in Python.
- List comprehension consists of an expression followed by **for statement** inside square brackets.

```
pow2 = [2 ** x for x in range(6)]
print(pow2)# Output: [1, 2, 4, 8, 16, 32]
# equivalent to
pow2 = []
for x in range(6):
   pow2.append(2 ** x)
print(pow2)
```

- A list comprehension can optionally contain more for or if statements. An optional if statement can filter out items for the new list.

```
pow2 = [2 ** x for x in range(10) if x > 5]
print(pow2)
```

Using with Membership Operators and for loop in a list

- We can test if an item exists in a list or not, using the keyword **in** and **not in**

```
list1=[1,3,5,7,11,13,'hi']
print(13 in list1) #True
print('hi' in list1)#True
print('bye' not in list1) #True
print(11 not in list1)#False
```

- Using a **for** loop we can iterate each item in a list

```
iiits=['Nuzvid','RKValley','Srikakulam','Ongole']
for i in iiits:
    print("RGUKT IIIT ",i)
#RGUKT IIIT  Nuzvid
#RGUKT IIIT  RKValley
#RGUKT IIIT  Srikakulam
#RGUKT IIIT  Ongole
```

Built-in Functions with List

- The **all()** method returns True when all elements in the given iterable are true. If not, it returns False.
- Syntax: **all(**iterable**)**

  The all() method takes a single parameter:

  iterable - any iterable (**list**, **tuple**, **dictionary**, etc.) which contains the elements
- In case of dictionaries,

  if all keys (not values) are true or

  the dictionary is empty, all() returns True.

  Else, it returns false for all other cases

```
list1=[1,3,5,7,11,13,'hi']
print(all(list1))#True
list2=[1,0,'','hi']
print(all(list2))#False
list3=[0,False,-1]
print(all(list3))#False
list4=[]
print(all(list4))#True
```

```
str1='RGUKT IIIT'
print(all(str1))#True
str2=' '
print(all(str2))#True
str3='000'
print(all(str3))#True

d1={0:False,1:True}
print(all(d1))#False
d2={1:True,2:'Two'}
print(all(d2))#True
d3={1:'One',False:0}
print(all(d3))#False
d4={}
print(all(d4))#True
```

- The **any()** method returns True if any element of an iterable is true. If not, this method returns False.
- They **any()** method takes an iterable (list, string, dictionary etc.) in Python.
- The any method returns: - **any(**iterable**)**

  True if at least one element of an iterable is true

  False if all elements are false or if an iterable is empty

When	Return Value
All values are true	True
All values are false	False
One value is true (others are false)	True
One value is false (others are true)	True
Empty Iterable	False

- In case of dictionaries, if all keys (not values) are false, any() returns False. If at least one key is true, any() returns True.

```
list1=[1,3,5,7,11,13]
print(any(list1))#True
list2=[1,0,'','hi']
print(any(list2))#True
list3=[0,False,-1]
print(any(list3))#True
list4=[]
print(any(list4))#False

str1='RGUKT IIIT'
print(any(str1))#True
str2=''
print(any(str2))#False
str3='000'
print(any(str3))#True

d1={0:False,1:True}
print(any(d1))#True
d2={1:True,2:'Two'}
print(any(d2))#True
d3={1:'One',False:0}
print(any(d3))#True
d4={}
print(any(d4))#False
```

- The **enumerate()** method adds counter to an iterable and returns it (the enumerate object).

  Syntax : **enumerate(**iterable , start=0**)**

- The **enumerate()** method takes two parameters:

  iterable - a sequence, an iterator, or objects that supports iteration

  start (optional) - enumerate() starts counting from this number. If *start* is omitted, 0 is taken as start.

Return Value from enumerate()

- The enumerate() method adds counter to an iterable and returns it. The returned object is a enumerate object.
- You can convert enumerate objects to list and tuple using **list()** and **tuple()** method respectively.

```
centers= ['NUZ','RKV','SKLM','ONGL']
enum_centers = enumerate(centers)
#<class 'enumerate'>
print(type(enum_centers))

# converting to list
print(list(enum_centers))
#[(0, 'NUZ'), (1, 'RKV'), (2, 'SKLM'), (3, 'ONGL')]
# changing the default counter
enum_centers = enumerate(centers,10)
print(list(enum_centers))
#[(10, 'NUZ'), (11, 'RKV'), (12, 'SKLM'), (13, 'ONGL')]

centers= ['NUZ','RKV','SKLM','ONGL']
for i in  enumerate(centers):
    print(i)
for i,j in enumerate(centers):
    print(i,j)
for i,j in enumerate(centers,100):
    print(i,j)
```

- The **len()** function returns the number of items (length) of an object.
- The **max()** method returns the largest element in an iterable or largest of two or more parameters.
- **Syntax : -**

  max / min (items**)**

  or

Max(list1,list2,key=function**)**

min(list1,list2,key=function**)**

- The **min()** method returns the smallest element in an iterable or smallest of two or more parameters

```
list1=[20,30,70]
print(len(list1))#3
list2=[22,45,79,300]
list3=[1700,1800,1900]
print(max(list1)) #
print(max(list2))

def sumDigit(num):
    sum = 0
    while(num):
        sum += num % 10
        num = int(num / 10)
    return sum
# using max(arg1, arg2, *args, key)
print('Maximum is:', max(100,321,267,59,40,key=sumDigit))
#267
# using max(iterable, key)
num = [15, 300, 2700, 821, 52, 10, 6]
print('Maximum is:', max(num, key=sumDigit))
#821
print(max(list1,list2,list3))
#[1700,1800,1900]
print(max(list1,list2,list3,key=len))
#[22,45,79,300]
```

- The **sorted()** method returns a sorted list from the given iterable.
- The sorted() method sorts the elements of a given iterable in a specific order - Ascending or Descending.
- Syntax : **sorted(**iterable[, key][, reverse]**)**
- Three parameters of sorted()
- **iterable** - sequence (string, tuple, list) or collection (set, dictionary, frozen set) or any iterator
- **reverse (Optional)** - If true, the sorted list is reversed (or sorted in Descending order)
- **key (Optional)** - function that serves as a key for the sort comparison
- A list also has **sort()** method which performs the same way as sorted(). Only difference being, sort() method doesn't return any value and changes the original list itself.

```
# vowels list
pyList = ['e', 'a', 'u', 'o', 'i']
print(sorted(pyList))
#['a', 'e', 'i', 'o', 'u']
# string
pyString = 'Python'
print(sorted(pyString))
#['P', 'h', 'n', 'o', 't', 'y']
# vowels tuple
pyTuple = ('e', 'a', 'u', 'o', 'i')
print(sorted(pyTuple))
#['a', 'e', 'i', 'o', 'u']

# set
pySet = {'e', 'a', 'u', 'o', 'i'}
print(sorted(pySet, reverse=True))
#['u', 'o', 'i', 'e', 'a']
# dictionary
pyDict = {'e': 1, 'a': 2, 'u': 3, 'o': 4, 'i': 5}
print(sorted(pyDict, reverse=True))
#['u', 'o', 'i', 'e', 'a']
list1=[(2, 2), (3, 4), (4, 1), (1, 3)]
def keyE(x):
    return x[1]
print(sorted(list1,key=keyE))
```

- The **sum()** function adds the items of an iterable and returns the sum.

  Syntax:- **sum(**iterable, start**)**

- The sum() function adds *start* and items of the given *iterable* from left to right.

```
list1=[23,34,10,5]
print(sum(list1))
#72
print(sum(list1,20))
#92
```

Practice Problems on Lists

1. Python program to find the sum of natural numbers up to n using recursive function?
2. Matrix Addition using Nested List Comprehension ?
3. Write a program to find the median of a list of numbers ?
4. Write a Python program to sum and multiplies all the items in a list ?
5. Write a Python program to get the largest and smallest number from a list ?
6. Write a Python function that takes two lists and returns True if they have at least one common member ?
7. Write a Python program to print a specified list after removing the prime index elements ?
8. Write a Python program to find the list of words that are longer than n from a given list of words ?
9. Write a Python program to generate and print a list of first and last 5 elements where the values are square of numbers between 1 and 30 (both included).?
10. Write a Python program to convert list to list of dictionaries?
11. Write a program to remove all duplicate values from a list?