Data Structure

Data structure collection of data elements(such as numbers or characters or even other data structure) that is structured in some way, for example, by numbering the elements. The most basic data structure in Python is the " Sequence "

Lists

--> List is one of the sequence data structure.

--> Lists are collection of items(Strings, Intergers or even other lists)

--> List are enclosed in [ ].

--> Each items in the list has assigned index values.

--> List are mutable, which means they can be changed.

List Creation

```
emptylist = []

lst = ['one', 'two', 'three', 'four',"five"] # list of strings

lst2 = [1, 2, 3, 4, 5, 6] # list of integers

lst3 = [[1, 2], [3, 4]] #list of lists

lst4 = [1, 2, 'three', 4.05] # list of different data types

print(emptylist)
print(lst)
print(lst2)
print(lst3)
print(lst4)
```

```
[]
['one', 'two', 'three', 'four', 'five']
[1, 2, 3, 4, 5, 6]
[[1, 2], [3, 4]]
[1, 2, 'three', 4.05]
```

List lenght

```
lst = ['one', 'two', 'three', 'four']
```

```
#find lenght of list
print(len(lst))
```

```
4
```

List Append

```
lst = [1, 2, 3, 4]
print(len(lst))
lst.append('six') # append will add the item at the end
print(lst)
print(len(lst))
```

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

```
lst = [1, 2, 3, 4]
lst.append('six')
print(lst)
```

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

```
lst = [1, 2, 3, 4]
print(lst.append([5,6]))
```

```
None
```

List Insert

```
# systex: lst.insert(x,y) will insert element y at location x

lst = ['one', 'two', 'three','five']
lst.insert(3, 'four')

print(lst)
lst.insert(0, 'zero')
print(lst)
```

```
['one', 'two', 'three', 'four', 'five']
['zero', 'one', 'two', 'three', 'four', 'five']
```

```
lst1 = [10,20,30,40]
print(lst1.insert(1,50))
print(lst1)
```

```
None
[10, 50, 20, 30, 40]
```

```
lst1 = [10,20,30,40]
print(lst1.insert(2,50))
print(lst1)
```

```
None
[10, 20, 50, 30, 40]
```

```
lst1 = [100,200,300,400,500]        # 100,200,300,400,500
lst1.insert(-1,150)                 # 100,200,300,400,150,500
print(lst1)
```

```
[100, 200, 300, 400, 150, 500]
```

List Remove

```
# syntax: lst.remove(x)

lst = ['one', 'two','two', 'three', 'four', 'two']
lst.remove('two') # it will remove first occurence of 'two' in given
list
print(lst)
```

```
['one', 'two', 'three', 'four', 'two']
```

```
lst1 = [10,20,30,40,50]
lst1.remove()
print(lst1)
```

```
----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
<ipython-input-24-d3d87da4b7fa> in <module>
      1 lst1 = [10,20,30,40,50]
----> 2 lst1.remove()
      3 print(lst1)

TypeError: remove() takes exactly one argument (0 given)
```

List Append and Extend

```
lst1 = ['one', 'two', 'three']
lst2 = ['four', 'five', 'six','seven','eight']
# Append
lst1.append(lst2)
print(lst1)
print(lst1[3])
print(lst1[3][2])
```

```
['one', 'two', 'three', ['four', 'five', 'six', 'seven', 'eight']]
['four', 'five', 'six', 'seven', 'eight']
six
```

```
lst1 = ['one', 'two', 'three','four' ]
lst2 =['five', 'six','seven', 'eight']

# expend will join the list with list2
lst1.extend(lst2)
print(lst1)
```

```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

```
lst1 = [1,2,3]
tup1 = (4,5)
print(lst1.extend(tup1))
print(lst1)
```

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

List delete

```
# del to remove item based on index position
lst = ['one', 'two', 'three', 'four','five']

#del lst[1]
#print(lst)
# or we can use pop() method

a = lst.pop(1)
print(a)

print(lst)
```

```
two
['one', 'three', 'four', 'five']
```

```
lst = ['one', 'two', 'three', 'four', 'three']
 # remove an item  form list
lst.remove('three')
print(lst)
```

```
['one', 'two', 'four', 'three']
```

List related Keywords in Pyhton

```
# keyword 'in' is used to test if an item is in a list
lst = ['one', 'two', 'three', 'four']
if 'two' in lst:
        print('AI')
# keyword 'not ' can combined with 'in'
if 'six' not in lst:
    print('ML')
```

```
AI
ML
```

List Reverse

```
lst = ['one', 'two', 'three', 'four', 'five']

lst.reverse()
print(lst)
```

```
['five', 'four', 'three', 'two', 'one']
```

```
lst = ['one', 'two', 'three', 'four', 'five']
lst1 = lst[::-1]
print(lst1)
```

```
['five', 'four', 'three', 'two', 'one']
```

List Sorting

The easiest way to sort a list is with the sorted(list) function.

that takes a list as input and returns a new list with those elements in sorted order.

the original list in not changed.

the sorted() optional agrument reverse= True, e.g sorted(list, reverse=True) makes it sort backwords.

```
lst = [ 25, 50, 35, 10, 51, 70]

sorted_lst = sorted(lst,reverse= True)
print("Sorted list:", sorted_lst)
print("Original list:", lst)
```

```
Sorted list: [70, 51, 50, 35, 25, 10]
Original list: [25, 50, 35, 10, 51, 70]
```

```
# print a list in reverse sorted order
print(" Reverse sorted list:", sorted(lst,reverse=True))
```

```
 Reverse sorted list: [70, 51, 50, 35, 29, 25, 10]
```

```
lst = [ 25, 50, 35.7, 10, 51.79]
lst.sort()
print("sorted list:",lst)
```

```
sorted list: [10, 25, 35.7, 50, 51.79]
```

```
lst = [ 25, 50, 'a', 10, 'b', 70, 29]
print(lst.sort()) # sort list with element of different datatype
```

```
----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
<ipython-input-39-0c43e05a494a> in <module>
      1 lst = [ 25, 50, 'a', 10, 'b', 70, 29]
----> 2 print(lst.sort()) # sort list with element of different
datatype

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

```
lst = [ 25, 50, 'a', 10, 'b', 70, 29]
a = sorted(lst)
print(a)
```

```
----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
<ipython-input-40-070e4b25f1d2> in <module>
      1 lst = [ 25, 50, 'a', 10, 'b', 70, 29]
----> 2 a = sorted(lst)
      3 print(a)

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

List Having a Multiple References

```
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9]
abc = lst
hr = abc
hr.append(10)
print(hr)
print(abc)
# print original list
print("Original list:", lst)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Original list: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

String Split to Create a list

```
# let's take a string

s = "one, two, three, four, five six"
slst = s.split(',')
print(slst)
```

```
['one', ' two', ' three', ' four', ' five six']
```

```
c = "a+b+c+d+e"
a = c.split("+")
print(a)
```

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

```
h = " My Name Is Harsh Raj "
split_h = h.split() # default split is white character: space or tab
print(split_h)
```

```
['My', 'Name', 'Is', 'Harsh', 'Raj']
```

List Indexing

Each item in the list has an assigned index value starting from 0.

Accessing elements in a list is called indexing.

```
lst = [1, 2, 3, 4, 5]
print(lst[2]) #print second element
print(lst[-4]) #print last element using negative index
```

```
3
2
```

List Slicing

Accessing parts of segements is called slicing.

The key point to remenber is that the: end value represents the first value that is not in the selected slice.

```
numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]

#print all the numbers
print(numbers[:])

#print form index 3 to 5
print(numbers[3:5])
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
[40, 50]
```

```
print(numbers)

# print alternate elements in a list
print(numbers[::3])

# print elements start form 2 through rest of the list
print(numbers[2::2])
```

```
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110]
[10, 40, 70, 100]
[30, 50, 70, 90, 110]
```

List Extend Using " + "

```
lst1 = [1, 2, 3, 4, 5,6]
lst2 = ["Harsh", "Raj", "Singh", '26']
lst3 = ['a','b','c']

new_lst = lst2 + lst1 +lst3
print(new_lst)
```

```
['Harsh', 'Raj', 'Singh', '26', 1, 2, 3, 4, 5, 6, 'a', 'b', 'c']
```

List Count

```
lst3 = [1, 2, 3, 1, 2, 2, 1, 2, 3]

# frequency of 1 in a list
print(lst3.count(2))
```

```
4
```

List Looping

```
# loop through a list
lst = [1, 2, 3, 4, 5, 6]
for ele in lst:
    print(ele)
```

```
1
2
3
4
5
6
```

```
lst1 = [ele for ele in lst]
print(lst1)
```

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

List Comprehensions

List Comprehensions provide a concise way to create lists.

common application are to make new lists where element is the result so some operation applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

```
#without list comprehension
Squares = []
for i in range(20):     # [0,1,2,3,---,19]
    Squares.append(i**2)  # [0,1,4,9,16,25,]
print(Squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225,
256, 289, 324, 361]
```

```
# using list comprehension
Squares = [i**2 for i in range(10)]
print(Squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
# example

lst = [-10, -20, -30, 10, 20, 30]

# create a new list with values doubled
newlst = [i*2 for i in lst]
print(newlst)

# filter the list to exclude negative numbers
newlst = [i for i in lst if i >= 0]
print(newlst)

# create a list of touple like (number, square_of_number)
newlst = [(i, i**2) for i in range(10)]
print(newlst)
```

```
[-20, -40, -60, 20, 40, 60]
[10, 20, 30]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49),
(8, 64), (9, 81)]
```

Nested List Comprehensions

```
# let's suppose we have a matrix

matrix = [[1, 2, 3, 4],        #  matrix = [ [ 1, 2, 3, 4],[5, 6, 7, 
8],[9, 10, 11, 12]]
          [5, 6, 7, 8],
          [9, 10, 11, 12]
         ]

# transpose of a matrix without list comprehensions
# transpose matrix =
transpose = []
for i in range(4):       #  0,1,2,3
    lst = []             # [ ]
    for row in matrix:   # [0,1,2]
        lst.append(row[i])  # [2,6,10]
    transpose.append(lst)    # transpose  = [[1,5,9],[2,6,10],
[3,7,11],[4,8,12]]

print(transpose)
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

```
# with comprehension method
matrix = [[1, 2, 3, 4],
          [5, 6, 7, 8],
          [9, 10, 11, 12],
         ]

transpose = [[ row[i] for row in matrix] for i in range(4)]
print(transpose)
```

```
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

```
matrix = []
n = int(input("enter the number is raw"))
m = int(input("enter the number is column"))
for i in range (0,n):                   # i =0,1,2,3,4,5--,n-1
    lst = []                            # lst = []
    for j in range(0,m):               # j = 0,1,2,3--,m-1
        ele = int(input("Enter the element:"))
        lst. append(ele)               # [4,5,6]
    matrix.append(lst)                 #  matrix = [[1,2,3],[4,5,6]]
print(matrix)
```

```
enter the number is raw2
enter the number is column3
Enter the element:1
Enter the element:2
Enter the element:3
Enter the element:4
Enter the element:5
Enter the element:6
[[1, 2, 3], [4, 5, 6]]
```

```
n = int(input("enter the number is raw"))
m = int(input("enter the number is column"))
matrix = [[int(input()) for j in range(0,m)] for i in range(0,n)]
print(matrix)
```

```
enter the number is raw2
enter the number is column3
1
2
3
4
5
6
[[1, 2, 3], [4, 5, 6]]
```

```
list2 = input("Enter interger value seperated by. (comma) for list2:
")
list1 = input("Enter interger value seperated by. (comma) for list1:
")
list2 = list2.split(",")
list1 = list1.split(",")
x = list(map(int,list2))
y = list(map(int,list1))
list3 = []
if len(list1)== len(list2):
    for i in range(len(list2)):
        list3.append(x[i]-y[i])
    print(list3)
```

```
list1 = [1,2,3]
list2 = [3,2,1]
list2.sort()
if list1 == list2:
    print("true")
else:
    print("false")
```

```
lst = [(1,2,3),(4,5,6)]
lst[0]= (10,20,30)
print(lst)
lst[0][1] = 5
```

```
[(10, 20, 30), (4, 5, 6)]

----------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
<ipython-input-12-ee88f9c6ebc1> in <module>
      2 lst[0]= (10,20,30)
      3 print(lst)
----> 4 lst[0][1] = 5

TypeError: 'tuple' object does not support item assignment
```