Data Structures in Python

A data structure is a collection of data elements (such as numbers or characters—or even other data structures) that is structured in some way, for example, by numbering the elements. The most basic data structure in Python is the "sequence". https://docs.python.org/3/tutorial/datastructures.html (https://docs.python.org/3/tutorial/datastructures.html)

List

lst = [1,2,3,"ML"]

Tuple

tp= (1,2,3,4)

Sets

S = {1,2,3}

Dictonary

D = {'a': 1, 'b': 2}

Strings

S = "Hi Welcome"

List

0	1	2	3	4	| indexes
20	"ML"	2.8	True	3	| elements

-> List is one of the Sequence Data structure

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

-> Lists are enclosed in [ ]

-> Each item in the list has an assigned index value.

-> Each item in a list is separated by a comma

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

Append , Extend , Insert ,Remove, Pop, Clear, Index, Sort ,Reverse,Copy

List Creation

In [57]:

```
emptyList = []

myfirstlist = ['one', 'two', 'three', 'four']  # list of strings

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

lst3 = [["How are you?", 2], [3, 4],5]          # list of lists

lst4 = [1, 2, myfirstlist]                      # list of lists

lst5 = [1, 'ramu', 24, 1.24, True]              # list of different datatypes

print(emptyList)

print(type(myfirstlist))

print(lst3)

print(lst4)

print(lst5)
```

```
[]
<class 'list'>
[['How are you?', 2], [3, 4], 5]
[1, 2, ['one', 'two', 'three', 'four']]
[1, 'ramu', 24, 1.24, True]
```

min max sum function on list of integers

In [71]:

```
print(min(lst2))   # This requires a list of numbers only
print(max(lst2))
print(sum(lst2))
#import statistics
#mean(lst2)
```

```
1
4
10
```

Basic List operations

In [15]:

```
b=['Hi!',"Milan"] * 4          # Repatation
print(b)
print(list('Milan'))
3 in [1, 2, 3]
```

```
['Hi!', 'Milan', 'Hi!', 'Milan', 'Hi!', 'Milan', 'Hi!', 'Milan']
['M', 'i', 'l', 'a', 'n']
```

Out[15]:

```
True
```

Replacement in List

In [36]:

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

In [42]:

```
myfirstlist[1]= 2
```

In [43]:

```
print(myfirstlist)
```

```
['one', 2, 'three', 'four']
```

Range Function : iterator or Generator----->

range(n) gives all numbers from 0 upto n-1

genral syntax is range(start,end,stepsize)

In [17]:

```
range(15)   # same as xrange in Python2
```

Out[17]:

```
range(0, 15)
```

In [29]:

```
#print(range(15))
```

In [29]:

```
z=list(range(15))          # Goes up to 14 starts with 0 its also Python 2 output for xrang
z
```

Out[29]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

In [30]:

```
x=list(range(1,15))
x
```

Out[30]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

In [35]:

```
y=list(range(2,9,3))     # step size is 3 by default step is 1
y
```

Out[35]:

```
[2, 5, 8]
```

In [33]:

```
a = list(range(100,120))
a
```

In [138]:

```
#for x in range(50):                    # For Loop
# print(x)
```

Length Function

In [12]:

```
lst = ['one', 'two', 'three', 'four',6]

#find length of a list

print(len(lst))
```

```
5
```

List Append

In [34]:

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

lst.append('five') # append will add the item at the end

print(lst)
```

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

List Insert

In [5]:

```
#syntax: lst.insert(x, y)

lst = ['one', 'two', 'four']     # No "three"

lst.insert(2, "three") # will add element y at location x Location : --->  0 1 2

print(lst)
```

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

List Remove

In [8]:

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

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

lst.remove('two') #it will remove first occurence of 'two' in a given list

print(lst)
```

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

List Append & Extend

In [39]:

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

lst2 = ['five', 'six']

#append

lst.append(lst2)    # append lst 2 at the end of lst

print(lst)
```

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

In [11]:

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

lst2 = ['five', 'six']

#extend will join the list with list1

lst.extend(lst2)

print(lst)
```

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

List Delete

In [45]:

```
#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)
```

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

In [14]:

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

#remove an item from list
lst.remove('three')

print(lst)
```

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

List realted keywords in Python

In [15]:

```
#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

In [16]:

```
#reverse is reverses the entire list

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

lst.reverse()

print(lst)
```

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

List Sorting

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

That takes a list and returns a new list with those elements in sorted order.

The original list is not changed.

The sorted() optional argument reverse=True, e.g. sorted(list, reverse=True), makes it sort backwards.

In [35]:

```
#create a list with numbers
numbers = [3, 1, 6, 2, 8]

sorted_lst = sorted(numbers)


print("Sorted list :", sorted_lst)

#original list remain unchanged
print("Original list: ", numbers)
```

```
Sorted list : [1, 2, 3, 6, 8]
Original list:  [3, 1, 6, 2, 8]
```

In [36]:

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

#orginal list remain unchanged
print("Original list :",  numbers)
```

```
Reverse sorted list : [8, 6, 3, 2, 1]
Original list : [3, 1, 6, 2, 8]
```

In [46]:

```
lst = [1, 20, 5, 5, 4.2]

#sort the list and stored in itself

lst.sort()


print("Sorted list: ", lst)
```

```
Sorted list:  [1, 4.2, 5, 5, 20]
```

In [18]:

```
lst = [1, 20, 'b', 5, 'a']
print(lst.sort()) # sort list with element of different datatypes.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-18-98d08ff0e3ba> in <module>()
      1 lst = [1, 20, 'b', 5, 'a']
----> 2 print(lst.sort())

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

List Having Multiple References

In [19]:

```
lst = [1, 2, 3, 4, 5]
abc = lst              # abc points in same memory location (pointers--abc or lst c c++)
abc.append(6)

#print original list
print("Original list: ", lst)
```

```
Original list:  [1, 2, 3, 4, 5, 6]
```

String Split to create a list

In [51]:

```
s = "This is applied AI Course"
split_lst = s.split() # default split is white-character: space or tab
print(split_lst)
```

```
['This', 'is', 'applied', 'AI', 'Course']
```

List Indexing

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

Accessing elements in a list is called indexing.

In [26]:

```
lst = [1, 2, 3, 4]
print(lst[1]) #print second element

#print last element using negative index
print(lst[-2])              #Indexing could be  from right 0 , 1 ,2 ,3 ...or from left -1
```

```
2
3
```

List Slicing [a:b:c] = [start:end:stepsize]

Accessing parts of segments is called slicing.

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

In [10]:

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

#print all numbers
print(numbers[:])    #  : means start to end    a : b   ---> a = start and b = end but end-

#print from index 0 to index 3
print(numbers[0:4])
```

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

In [27]:

```
print (numbers)
#print alternate elements in a list
print(numbers[::2])                # a:b:c a = start b = end and c = step size
# Print every third number
print(numbers[::3])
#print elemnts start from 2 with step 2
print(numbers[2::2])
print(numbers[2:5:2])
```

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

List extend using "+"

In [52]:

```
lst1 = [1, 2, 3, 4]
lst2 = ['Joshi', 'Udavant', 'Nemade', 'Patil']
new_lst = lst1 + lst2

print(new_lst)
```

```
[1, 2, 3, 4, 'Joshi', 'Udavant', 'Nemade', 'Patil']
```

List Count

In [82]:

```
numbers = [1, 2, 3, 1, 3, 4, 2,3,3, 5]

#frequency of 1 in a list
print(numbers.count(1))

# Question frequency of 3 in a list ?
```

```
2
```

List Looping

In [86]:

```
#loop through a list

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

for ele in lst:
    print(ele)

# What happen if print(2*ele)
```

```
one
two
three
four
```

List Comprehensions

List comprehensions provide a concise way to create lists.

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

In [87]:

```
# without list comprehension
squares = []   # Creating empty list
for i in range(10):
    squares.append(i**2)   #list append
#print(squares)

# Above three lines of code can be wiriiten in one line
```

In [91]:

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

#mul2 = [2*x for x in range(12)]
#print(mul2)
```

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

In [36]:

```
#example

lst = [-10, -20, 10, 20, 50]

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

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


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

```
[-20, -40, 20, 40, 100]
[10, 20, 50]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 6
4), (9, 81)]
```

Nested List Comprehensions

In [129]:

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

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

#transpose of a matrix without list comprehension
transposed = []
for i in range(4):
    lst = []
    for row in matrix:
        lst.append(row[i])
    transposed.append(lst)

print(transposed)
```

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

In [94]:

```
#with list comprehension
transposed = [[row[i] for row in matrix] for i in range(4)]
print(transposed)
```

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

In [101]:

```
[*zip(*matrix)]
```

Out[101]:

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

In [105]:

```
#print(emoji.emojize('Python is :thumbs_up_sign:'))
```

```
Python is :thumbs_up_sign:
```

In [130]:

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

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

In [131]:

```
print(lst[1][2])
```

```
6
```

In [132]:

```
lst = [ 1, (2,3,4),6]
```

In [133]:

```
lst
```

Out[133]:

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

In [134]:

```
lst[1]
```

Out[134]:

```
(2, 3, 4)
```

In [135]:

```
lst[2]
```

Out[135]:

```
6
```

In [137]:

```
lst[1][1]
```

Out[137]:

```
3
```

In [139]:

```
lst=[(1,2),(4,5)]
```

In [140]:

```
lst
```

Out[140]:

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

In [141]:

```
lst[1]
```

Out[141]:

```
(4, 5)
```

In [142]:

```
lst[1][1]
```

Out[142]:

```
5
```

In [ ]:

```

```