Python Lists

-> 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.

In [1]:
```
#List length

lst = ['a', 'b', 'c', 'd']

#find length of a list
print(len(lst))
```

```
4
```

In [2]:
```
#List append

lst.append('e') # append will add the item at the end
print(lst)
```

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

In [3]:
```
#List insert

lst.insert(2, "f") # will add element y at location x

print(lst)
```

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

In [4]:
```
#List remove

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

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

In [6]:
```
#List append and extend

lst = ['a', 'b', 'c', 'd']
lst1 = ['1', '2', '3', '4']

#append
lst.append(lst1)
print(lst)
```

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

In [8]:
```
#extend will join the list with list1
```

```
lst = ['a', 'b', 'c', 'd']
lst1 = ['1', '2', '3', '4']

lst.extend(lst1)
print(lst)
```

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

In [11]:

```
#List delete
#del to remove item based on index position

lst = ['a', 'b', 'c', 'd']

del lst[1]
print(lst)

lst = ['a', 'b', 'c', 'd']
# we can use pop method
a=lst.pop(1)
print(a)

lst = ['a', 'b', 'c', 'd']
#remove an item from list
lst.remove('d')
print(lst)
```

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

In [12]:

```
# "in" keyword in list

lst = ['a', 'b', 'c', 'd']

if 'a' in lst:
    print("a is present in list")

if 'f' not in lst:
    print("f not in list")
```

```
a is present in list
f not in list
```

In [13]:

```
# reverse a list

lst = ['a', 'b', 'c', 'd']
lst.reverse()
print(lst)
```

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

sort a list

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 [16]:

```
lst=[3,6,5,7,9]

sort_lst=sorted(lst)
```

```
print("sorted list :",sort_lst)

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

```
sorted list : [3, 5, 6, 7, 9]
Original list:  [3, 6, 5, 7, 9]
```

In [17]:

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

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

```
Reverse sorted list : [9, 7, 6, 5, 3]
Original list : [3, 6, 5, 7, 9]
```

In [18]:

```
#sort the list and store it in itself

lst=[1.1,0.1,6.7,7,9]
lst.sort()

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

```
sorted list:  [0.1, 1.1, 6.7, 7, 9]
```

In [19]:

```
#list with multiple references

lst=[1,2,3,4]
lst1=lst

lst1.append(5)
#print original list
print("original list: ", lst)
```

```
original list:  [1, 2, 3, 4, 5]
```

In [20]:

```
# string split to list

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

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

In [21]:

```
# 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]
print(lst[1]) #print second element

#print last element using negative index
print(lst[-2])
```

```
2
3
```

In [22]:

```
# list slicing

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

```
#print all numbers
print(lst[:])

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

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

In [24]:

```
print (lst)
#print alternate elements in a list
print(lst[::2])


#print alternate elements start from 2 through rest of the list
print(lst[2::2])
```

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

In [25]:

```
# list extend using "+"

lst1=[1,2,3]
lst2=[4,5,6]

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

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

In [26]:

```
# list count to find the frequency of a number in a list

lst=[1,2,1,5,1,5,6,9,9]

#frequence of 1

print(lst.count(1))
```

```
3
```

In [27]:

```
# list looping

lst=[1,2,3,4]

for ele in lst:
    print(ele)
```

```
1
2
3
4
```

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 [28]:
```
#using list comprehension
squares = [i**2 for i in range(10)]
print(squares)
```

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

In [29]:
```
# nested list comprehensions

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

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]]
```

Tuples

-> A tuple is similar to list

-> The diffence between the two is that we can't change the elements of tuple once it is assigned whereas in the list, elements can be changed

In [30]:
```
# tuple creation

#empty tuple
t = ()

#tuple having integers
t = (1, 2, 3)
print(t)

#tuple with mixed datatypes
t = (1, 'a', 28, 'abc')
print(t)

#nested tuple
t = (1, (2, 3, 4), [1, 'def', 28, 'abc'])
print(t)


#only parenthesis is not enough
t = ('def')
type(t)


#need a comma at the end
t = ('def',)
type(t)


#parenthesis is optional
t = "def",
print(type(t))

print(t)
```

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

```
(1, 'a', 28, 'abc')
(1, (2, 3, 4), [1, 'def', 28, 'abc'])
<class 'tuple'>
('def',)
```

In [31]:
```
# accessing elements in a tuple

t=(1,2,'a',3,4)

print(t[2])
```

```
a
```

In [32]:
```
# negative index
print(t[-1])   # print last element in the tuple
```

```
4
```

In [33]:
```
# nested tuple

t=('a',(1,2,3,4))

print(t[1])
```

```
(1, 2, 3, 4)
```

In [35]:
```
print(t[1][2])
```

```
3
```

In [36]:
```
# slicing
t = (1, 2, 3, 4, 5, 6)

print(t[1:4])

#print elements from starting to 2nd last elements
print(t[:-2])

#print elements from starting to end
print(t[:])
```

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

Changing a Tuple

unlike lists, tuples are immutable This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed.

In [37]:
```
#creating tuple
t = (1, 2, 3, 4, [5, 6, 7])

t[2] = 'x' #will get TypeError
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-37-1fb87c1270c4> in <module>
```

```
      2 t = (1, 2, 3, 4, [5, 6, 7])
      3
----> 4 t[2] = 'x' #will get TypeError

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

In [38]:
```
t[4][1] = 'a'
print(t)
```

```
(1, 2, 3, 4, [5, 'a', 7])
```

In [39]:
```
#repeat the elements in a tuple for a given number of times using the * operator.
t = (('a', ) * 4)
print(t)
```

```
('a', 'a', 'a', 'a')
```

In [41]:
```
# tuple count

t=(1,1,1,1,3,4,5,6,6)

#get the frequency of particular element appears in a tuple
print(t.count(1))

print(t.index(1)) #return index of the first element is equal to 1

#print index of the 1
```

```
4
0
```

In [42]:
```
# tuple membership

#test if an item exists in a tuple or not, using the keyword in.
t = (1, 2, 3, 4, 5, 6)

print(1 in t)
```

```
True
```

In [43]:
```
print(7 in t)
```

```
False
```

In [44]:
```
# built in functions

# tuple length
t = (1, 2, 3, 4, 5, 6)
print(len(t))
```

```
6
```

In [45]:
```
# tuple sort
t = (4, 5, 1, 2, 3)

new_t = sorted(t)
print(new_t) #Take elements in the tuple and return a new sorted list
             #(does not sort the tuple itself).
```

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

In [46]:
```
#get the largest element in a tuple
t = (2, 5, 1, 6, 9)

print(max(t))
```

```
9
```

In [47]:
```
#get the smallest element in a tuple
print(min(t))
```

```
1
```

In [48]:
```
#get sum of elments in the tuple
print(sum(t))
```

```
23
```

In [ ]:
```

```