Lists and Tuples

What we will learn?

- Lists

> Creating list

> Accessing list elements

> Updating elements of list

> Operations on lists

> Methods to process list

> Nested lists

> List comprehensions

- Tuples

> Creating tuple

> Accessing tuple elements

> Operations on tuples

> Functions to process tuples

> Nested tuples

> Modification of elements in a tuple

Lists

It consists of a group of elements.

A single list may contain DataTypes like Integers, Strings, as well as Objects.

They are dynamic, mutable & ordered.

It is represented by square brackets.

Creating empty list

```
l1=[]
```

Initializing list

```
student1=[1,'ABC','Comp',20]
```

Creatng list using range function

```
l2=list(range(2,9,2)) #list() converts sequence to a list
print(l2)
```

```
     [2, 4, 6, 8]
```

Finding length of a list

```
print(len(l2))
```

```
     4
```

Indexing & slicing of a list

Indexing & Slicing in list is similar to array indexing & slicing.

```
print(student1[1])
print(student1[1:3])
print(student1[::-1])
print(student1[-2])
```

```
     ABC
     ['ABC', 'Comp']
     [20, 'Comp', 'ABC', 1]
     Comp
```

Accessing list elements

There are various ways in which we can access the elements of a list.

```
list1=[1,2,3,4,5]
i=0
```

```
while i<len(list1):
  print(list1[i])
  i+=1

     1
     2
     3
     4
     5
```

```
for i in list1:
  print(i)

     1
     2
     3
     4
     5
```

Updating the elements of a list

```
lst=[11,22,33,44]
#update using index
lst[0]=1
print(lst)

#update using append method
lst.append(55)
print(lst)

#update using slicing
lst[2:4]=[2,3]
print(lst)

#deleting an element using del keyword
del lst[1]
print(lst)

#deleting an element using remove method
lst.remove(55)
print(lst)

#deleting last element using pop method
lst.pop()
print(lst)

#deleting an element using pop method
lst.pop(1)
print(lst)
```

```
[1, 22, 33, 44]
[1, 22, 33, 44, 55]
[1, 22, 2, 3, 55]
[1, 2, 3, 55]
[1, 2, 3]
[1, 2]
[1]
```

Operations on list

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

#reversal of list using slicing operator
print(lst[::-1])
#reversal of list using reverse method
lst.reverse()
print(lst)
```

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

```
#concatenation of two lists using + operator
l1=[1,2,3]
l2=['a','b']
l3=l1+l2
print(l3)
```

```
[1, 2, 3, 'a', 'b']
```

```
#repetition of list using * operator
l1=[1,2,3]
print(l1*2)
```

```
[1, 2, 3, 1, 2, 3]
```

```
#searching elements using membership operator
print(4 in l1)
print(4 not in l1)
```

```
False
True
```

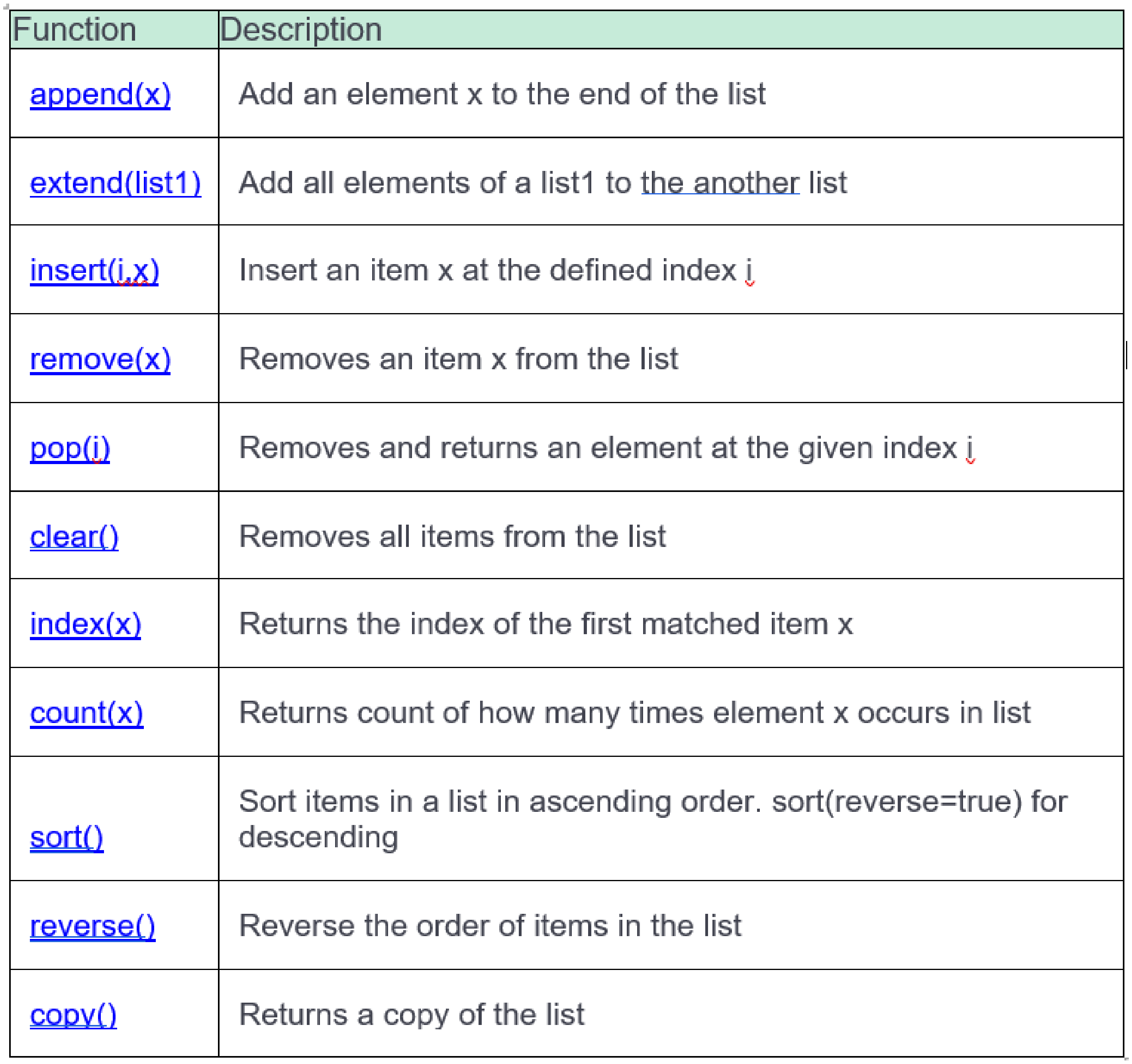
```
#aliasing a list
l2=l1
print(l1,id(l1))
print(l2,id(l2))
```

```
[1, 2, 3] 140711308724736
[1, 2, 3] 140711308724736
```

```
#cloning using slice operator
l2=l1[:]
print(id(l1))
print(id(l2))

#cloning using copy method
l3=l1.copy()
print(id(l1))
print(id(l3))
```

```
140711308724736
140711345714912
140711308724736
140711308724976
```

Methods to process list



Function	Description
append(x)	Add an element x to the end of the list
extend(list1)	Add all elements of a list1 to the another list
insert(i,x)	Insert an item x at the defined index i
remove(x)	Removes an item x from the list
pop(i)	Removes and returns an element at the given index i
clear()	Removes all items from the list
index(x)	Returns the index of the first matched item x
count(x)	Returns count of how many times element x occurs in list
sort()	Sort items in a list in ascending order. sort(reverse=true) for descending
reverse()	Reverse the order of items in the list
copy()	Returns a copy of the list

```
#Finding biggest and smallest elements in list
lst=[22,33,11,77,88,99,0]
print(max(lst))
print(min(lst))
```

```
99
0
```

Program to find common elements in two lists

```
l1=[1,2,3,4,5]
l2=[2,4]
```

```
s1=set(l1)
s2=set(l2)
s3=s1.intersection(s2)

l3=list(s3)
print(l3)
```

```
[2, 4]
```

Nested lists

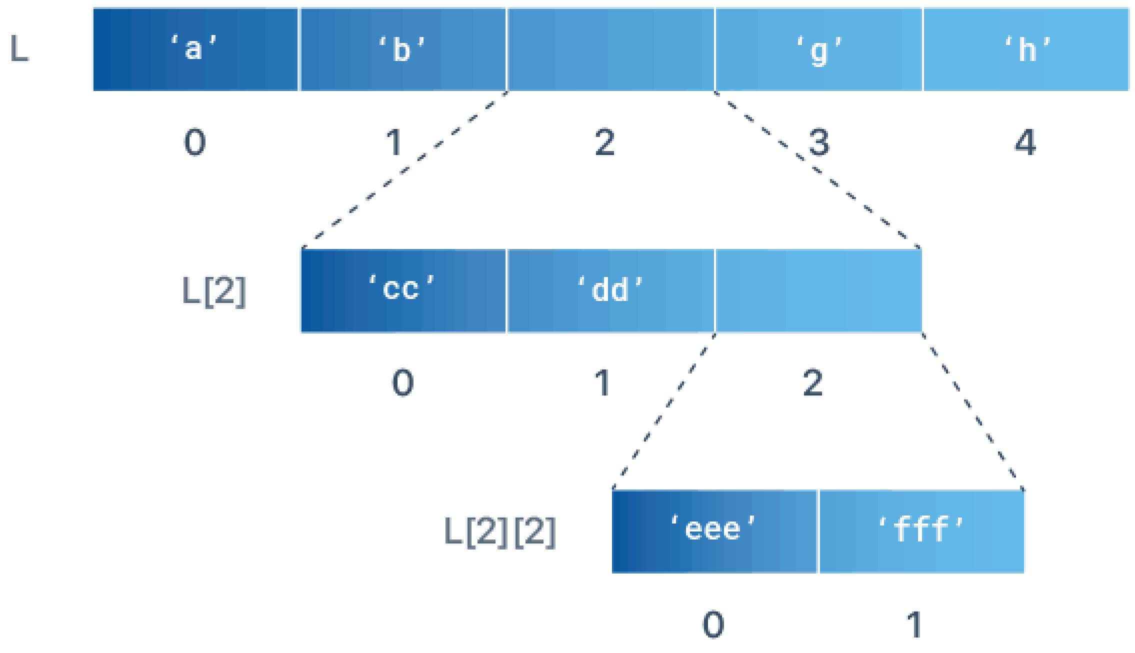
Let's understand concept of nested list by following program.

```
l1=[10,20,30,[80,90]]
#Nested list can be accessed using following ways
print(l1[3])

for i in l1[3]:
  print(i)

#changing first element of nested list
l1[3][0]=70
print(l1[3])
```

```
[80, 90]
80
90
[70, 90]
```

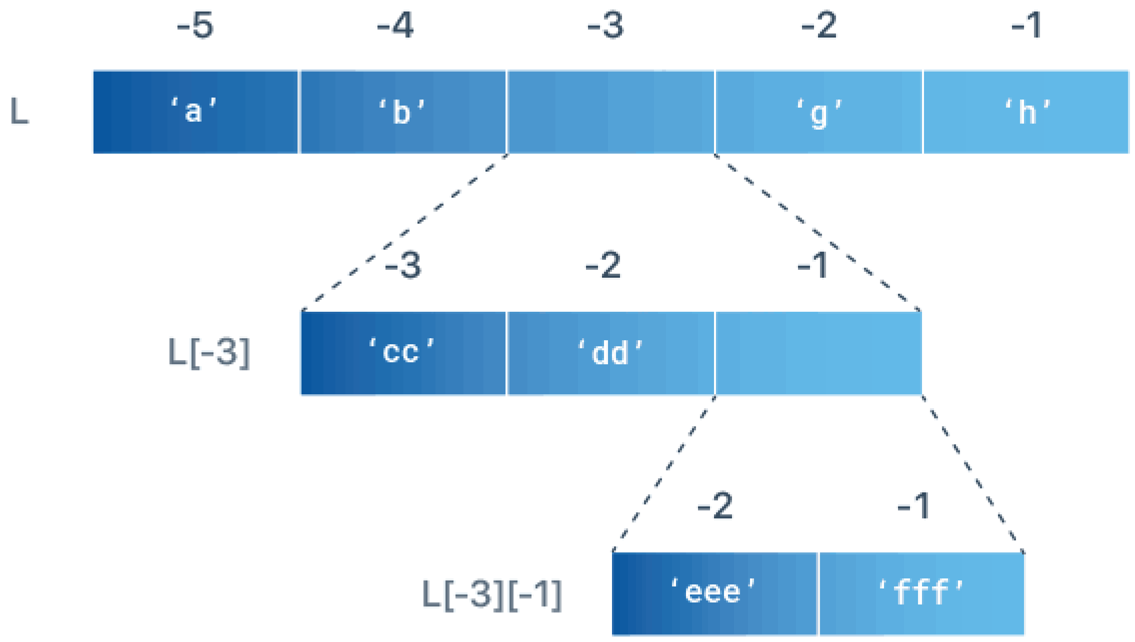
```
L = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']

print(L[2])
print(L[2][2])
print(L[2][2][0])
```

```
['cc', 'dd', ['eee', 'fff']]
['eee', 'fff']
eee
```



```
L = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']

print(L[-3])
print(L[-3][-1])
print(L[-3][-1][-2])
```

```
    ['cc', 'dd', ['eee', 'fff']]
    ['eee', 'fff']
    eee
```



Operations on nested list

```
L = ['a', ['bb', 'cc'], 'd']
L[1].append('xx')
print(L)

L = ['a', ['bb', 'cc'], 'd']
L[1].insert(0,'xx')
print(L)

L = ['a', ['bb', 'cc'], 'd']
L[1].extend([1,2,3])
print(L)

L = ['a', ['bb', 'cc', 'dd'], 'e']
x = L[1].pop(2)
print(L)

L = ['a', ['bb', 'cc', 'dd'], 'e']
L[1].remove('cc')
print(L)

print(len(L[1]))
```

```
['a', ['bb', 'cc', 'xx'], 'd']
['a', ['xx', 'bb', 'cc'], 'd']
['a', ['bb', 'cc', 1, 2, 3], 'd']
['a', ['bb', 'cc'], 'e']
['a', ['bb', 'dd'], 'e']
2
```

Nested lists as matrices

```
mat=[[1,2,3],[4,5,6],[8,9,11]]
#printing rows
for r in mat:
  print(r)

#printing matrix
for r in mat:
  for c in r:
    print(c,end=" ")
  print()

#printing matrix using index
for i in range(len(mat)):
  for j in range(len(mat[i])):
    print(mat[i][j],end=" ")
  print()

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

List comprehensions

List comprehensions in Python provide us with a short and concise way to construct list using sequences like list, set, tuple, dictionary or range which have been already defined.

Syntax:

output_list = [output_exp for var in input_list if (var satisfies this condition)]

Note:

List comprehension may or may not contain an if condition.

create list of even numbers without using list comprehensions

```
even_nums = []
for x in range(21):
    if x%2 == 0:
        even_nums.append(x)
print(even_nums)

     [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

create list of even numbers with using list comprehensions

```
even_nums = [x for x in range(21) if x%2 == 0]
print(even_nums)

     [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Some examples of list comprehension

```
squares=[x**2 for x in range(1,10)]
print(squares)
```

```
even_squares=[x**2 for x in range(1,10) if x%2==0]
print(even_squares)
```

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

```
#adding each elements of two lists with one another

list1=[1,2,3]
list2=[4,5,6,7]

list3=[x+y for x in list1 for y in list2]
print(list3)
```

```
[5, 6, 7, 8, 6, 7, 8, 9, 7, 8, 9, 10]
```

```
list4=[x+y for x in 'ABC' for y in 'DE']
print(list4)
```

```
['AD', 'AE', 'BD', 'BE', 'CD', 'CE']
```

Tuples

A tuple in Python is similar to a list.

The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list.

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

Creating tuple

```
#empty tuple
tup1=()

#tuple with one element
tup2=(1,)   #without comma it will consider an integer

#tuple with multiple elements
tup3=(1,2,3,'python',2.5)
        #or
tup4=1,2,3,'python',2.5

#creating tuple using list
```

```
list1=[1,2,3]
tup5=tuple(list1)

#creating tuple using range function
tup6=tuple(range(1,10))
```

Accessing tuple elements

```
tup1=(7,8,9,10,11)

print(tup1[3])
print(tup1[2:4])
t1,t2=tup1[1:3]
print(t1)
print(t2)
```

```
10
(9, 10)
8
9
```

Operations on tuples

```
#finding length
print(len(tup1))
```

```
5
```

```
#concatenation of two tuples
tup2=('a','b','c')
tup3=tup1+tup2
print(tup3)
```

```
(7, 8, 9, 10, 11, 'a', 'b', 'c')
```

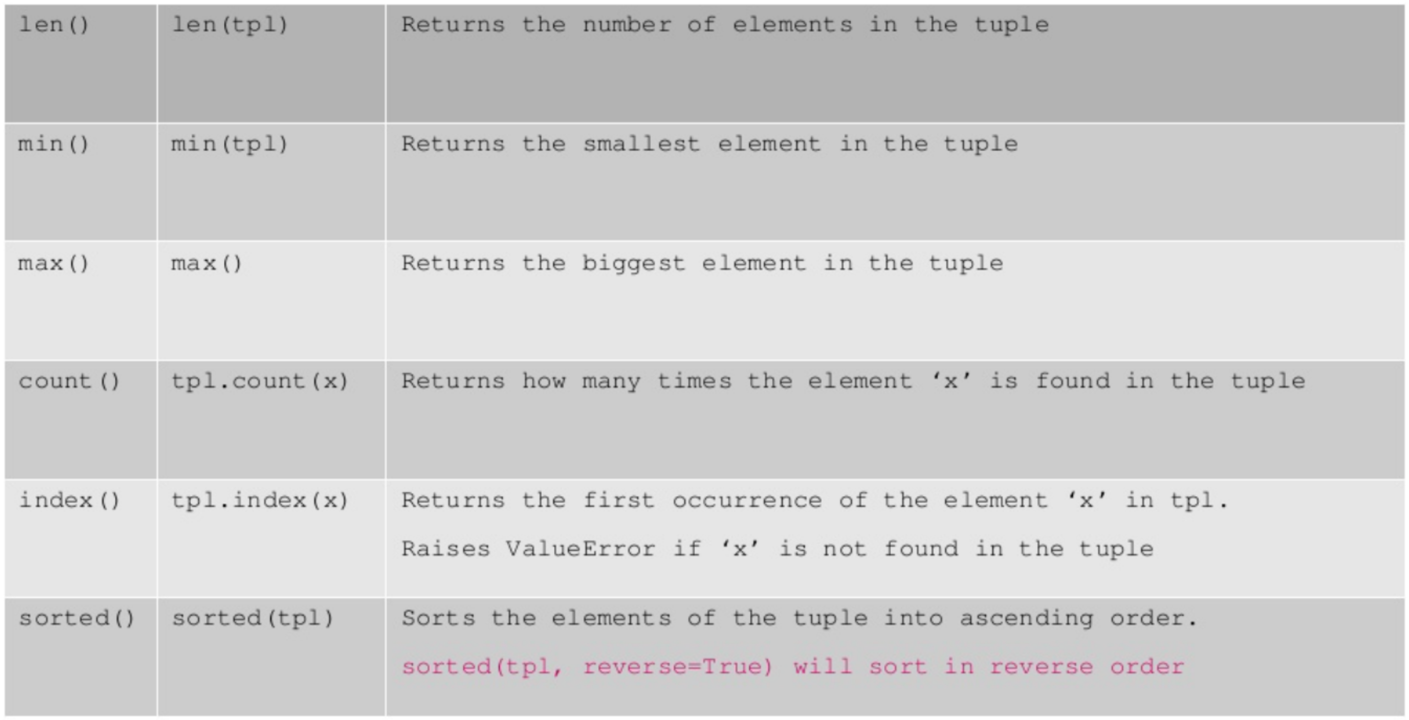
```
#repetition
tup4=(20,)*4
print(tup4)
```

```
(20, 20, 20, 20)
```

```
#use of membership operator
print('a' in tup2)
```

```
True
```

Functions to process tuples



len()	len(tpl)	Returns the number of elements in the tuple
min()	min(tpl)	Returns the smallest element in the tuple
max()	max()	Returns the biggest element in the tuple
count()	tpl.count(x)	Returns how many times the element ‘x’ is found in the tuple
index()	tpl.index(x)	Returns the first occurrence of the element ‘x’ in tpl. Raises ValueError if ‘x’ is not found in the tuple
sorted()	sorted(tpl)	Sorts the elements of the tuple into ascending order. sorted(tpl, reverse=True) will sort in reverse order

Note: No functions available to insert, remove, modify elements in tuple as tuples are immutable in python.

Taking tuple as input

```
#eval() use to evaluate the sequence type

tup1=eval(input("Enter elements in tuple"))
print(type(tup1))
```

```
Enter elements in tuple1,2,3,4
<class 'tuple'>
```

```
str=input("Enter tuple seperated by commas").split(',')
lst=[int(x) for x in str]
tup1=tuple(lst)
print(tup1)
```

```
Enter tuple seperated by commas1,2,3,4
(1, 2, 3, 4)
```

Nested tuples

```
tup1=(11,22,33,(44,55))
print(tup1[3])
print(tup1[3][1])

     (44, 55)
     55
```

```
#storing record of employees in form of (id,name,salary)
emp=((4,'abc',1000),(2,'def',500),(3,'xyz',1800))
print(sorted(emp))    #by default it sorts with respect to 0th element of nested tuple
print(emp)

     [(2, 'def', 500), (3, 'xyz', 1800), (4, 'abc', 1000)]
     ((4, 'abc', 1000), (2, 'def', 500), (3, 'xyz', 1800))
```

```
#sorting with respect to salary
print(sorted(emp,key=lambda x:x[2]))

     [(2, 'def', 500), (4, 'abc', 1000), (3, 'xyz', 1800)]
```

Modifications of elements in tuple

Appending elements in tupple

```
tupleObj = (12 , 34, 45, 22, 33 )
print(id(tupleObj))
tupleObj = tupleObj + (19 ,) #we need to create a copy of existing tuple and then add new ele
print(tupleObj)
print(id(tupleObj))

     140711309455728
     (12, 34, 45, 22, 33, 19)
     140711324630384
```

```
tupleObj = (12 , 34, 45, 22, 33 )
l1=list(tupleObj)
l1.append(19)
tupleObj=tuple(l1)
print(tupleObj)

     (12, 34, 45, 22, 33, 19)
```

Insert an element at specific index in tuple

```
tupleObj = (12,34,45,22,33)
n = 2
# Insert 19 in tuple at index 2
tupleObj = tupleObj[ : n ] + (19 ,) + tupleObj[n : ]
print(tupleObj)
```

```
     (12, 34, 19, 45, 22, 33)
```

Modify / Replace the element at specific index in tuple

```
tupleObj =  (12, 34, 19, 45, 22, 33, 19)
n = 2
# Replace the element at index 2 to 'Test'
tupleObj = tupleObj[ : n] + ('test' ,) + tupleObj[n + 1 : ]
print(tupleObj)
```

```
     (12, 34, 'test', 45, 22, 33, 19)
```

Delete an element at specific index in tuple

```
tupleObj =(12, 34, 'test', 45, 22, 33, 19)
n = 2
# Delete the element at index 2
tupleObj = tupleObj[ : n ] + tupleObj[n+1 : ]
print(tupleObj)
```

```
     (12, 34, 45, 22, 33, 19)
```