More on Lists

In [2]:

```
# List declaration
todoList =["Hangout","Coding","Teaching Python","Teaching Java"]
```

In [3]:

```
# Add an item to the end of the list.
todoList.append("Sleeping")
print(todoList)
```

```
['Hangout', 'Coding', 'Teaching Python', 'Teaching Java', 'Sleeping']
```

In [4]:

```
# extend() method adds the argument to the caller list
schedule =["Thug Life", "Party","Family Reunion"]
# The parameter must be iterable.
todoList.extend(schedule)
print(todoList)
```

```
['Hangout', 'Coding', 'Teaching Python', 'Teaching Java', 'Sleeping', 'Thug Life', 'Party', 'Famil
y Reunion']
```

In [5]:

```
# insert() method helps us to insert the element at any given position.
todoList.insert(0,"Wake up!")
print(todoList)
```

```
['Wake up!', 'Hangout', 'Coding', 'Teaching Python', 'Teaching Java', 'Sleeping', 'Thug Life', 'Pa
rty', 'Family Reunion']
```

In [6]:

```
# remove() removes the argument from the list. If no such element is present it will throw a value
error!
todoList.remove("Thug Life")
print(todoList)
```

```
['Wake up!', 'Hangout', 'Coding', 'Teaching Python', 'Teaching Java', 'Sleeping', 'Party', 'Family
Reunion']
```

In [7]:

```
# pop() removes the item from the given argument index position of the list.
# If no argument is provided, it removes the last element from list.
todoList.pop()
print(todoList)
todoList.pop(6)
print(todoList)
```

```
['Wake up!', 'Hangout', 'Coding', 'Teaching Python', 'Teaching Java', 'Sleeping', 'Party']
['Wake up!', 'Hangout', 'Coding', 'Teaching Python', 'Teaching Java', 'Sleeping']
```

In [8]:

```
# Clears the items in the list
todoList.clear()
todoList
```

Out[8]:

```
[]
```

In [11]:

```python
todoList=['Eat','Sleep','Code','Repeat']
todoList
```

Out[11]:

```
['Eat', 'Sleep', 'Code', 'Repeat']
```

In [12]:

```python
todoList.index('Sleep') # Returns the index position of the item. Error if not element is not pres
ent.
```

Out[12]:

```
1
```

In [13]:

```python
todoList.append('Eat')
todoList
```

Out[13]:

```
['Eat', 'Sleep', 'Code', 'Repeat', 'Eat']
```

In [14]:

```python
# count() count the number of times an element appears in the list.
todoList.count('Eat')
```

Out[14]:

```
2
```

In [15]:

```python
todoList.sort() # Sorts the list ascending order.
todoList
```

Out[15]:

```
['Code', 'Eat', 'Eat', 'Repeat', 'Sleep']
```

In [16]:

```python
todoList.sort(reverse=True) # Sorts the list in descending order.
todoList
```

Out[16]:

```
['Sleep', 'Repeat', 'Eat', 'Eat', 'Code']
```

In [17]:

```python
todoList.reverse() # Reverses the previous list.
todoList
```

Out[17]:

```
['Code', 'Eat', 'Eat', 'Repeat', 'Sleep']
```

In [18]:

```
# copy() returns a shallow copy of the list.
senCopy = todoList.copy()
senCopy
```

Out[18]:

```
['Code', 'Eat', 'Eat', 'Repeat', 'Sleep']
```

Using Lists as Stacks

In [19]:

```
stack=[]
# Pushing item in the stack
stack.append(2)
stack
```

Out[19]:

```
[2]
```

In [20]:

```
stack.append(5)
stack
```

Out[20]:

```
[2, 5]
```

In [21]:

```
#  Popping item from the stack
stack.pop()
stack
```

Out[21]:

```
[2]
```

Using Stack as Queues

In [22]:

```
from collections import deque
# Using List as Queue.
queue = deque(["Devjeet","Navin","Rajat"])
# Queue
queue.append("Rupak")
queue
```

Out[22]:

```
deque(['Devjeet', 'Navin', 'Rajat', 'Rupak'])
```

In [23]:

```
queue.popleft()
queue
```

Out[23]:

```
deque(['Navin', 'Rajat', 'Rupak'])
```

List Comprehensions

In [25]:

```
# 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 elem
ents
# that satisfy a certain condition.

# Normal way
square =[]
for i in range(1,10):
    square.append(i**2)
print(square)
```

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

In [26]:

```
# Alternative Method via List Comprehension
square2 = [i**2 for i in range(1,10)]
print(square2)
```

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

In [28]:

```
matrix=[[1,2,3],[4,5,6],[7,8,9]]
transposed = []
transposed = [[row[i] for row in matrix]for i in range(0,3)]
print(transposed)
```

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

Deleting the Data Structure using del

In [29]:

```
transposed
```

Out[29]:

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

In [30]:

```
del transposed[0] # Deletes a list within list, deletes the first element
transposed
```

Out[30]:

```
[[2, 5, 8], [3, 6, 9]]
```

In [31]:

```
del transposed[0][2] # Deletes an element within a nested list.
transposed
```

Out[31]:

```
[[2, 5], [3, 6, 9]]
```

In [32]:

```
del transposed
transposed # Deletes the whole data structure, unline clear
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-32-95d187d844a1> in <module>
      1 del transposed
----> 2 transposed

NameError: name 'transposed' is not defined
```

Tuples & Sequences

In [33]:

```
# Tuples are much like list but are immutable.
# All read only methods of lists works on tuple!
my_tuple ='Devjeet',21,'devjeetroy.dr@gmail.com'
my_tuple
```

Out[33]:

```
('Devjeet', 21, 'devjeetroy.dr@gmail.com')
```

In [34]:

```
# Nested Tuples
u = my_tuple,(8420169493,'CSE','NSEC') # we used parenthesis here to separate the nested tuples!
u
```

Out[34]:

```
(('Devjeet', 21, 'devjeetroy.dr@gmail.com'), (8420169493, 'CSE', 'NSEC'))
```

In [35]:

```
u[0]
```

Out[35]:

```
('Devjeet', 21, 'devjeetroy.dr@gmail.com')
```

In [36]:

```
u[1]
```

Out[36]:

```
(8420169493, 'CSE', 'NSEC')
```

In [37]:

```
u[0][2]
```

Out[37]:

```
'devjeetroy.dr@gmail.com'
```

In [39]:

```
u[1][2] = "MIT"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-f4669c23619c> in <module>
----> 1 u[1][2] = "MIT"
```

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

In [40]:

```
u.clear()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-40-ab1267e08043> in <module>
----> 1 u.clear()

AttributeError: 'tuple' object has no attribute 'clear'
```

In [41]:

```
ucopy = u.copy()
ucopy
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-41-bd5b2d735c8e> in <module>
----> 1 ucopy = u.copy()
      2 ucopy

AttributeError: 'tuple' object has no attribute 'copy'
```

Tuples are mainly used in places where data manipulation is not needed.

Eg: Historical Data, Cost Index model etc.

In [42]:

```
# A special problem is the construction of tuples containing 0 or 1 items.
# Empty tuples are constructed by an empty pair of parentheses;
# a tuple with one item is constructed by following a value with a comma
empty_tuple=()
print(empty_tuple)
```

```
()
```

In [43]:

```
oneItem='Devjeet'
type(oneItem)
```

Out[43]:

```
str
```

In [44]:

```
# But oneItem has to be a tuple so we have 1 option!

# Method of declaring singleton tuple!
oneItem_1 = 'Devjeet',
type(oneItem_1)
```

Out[44]:

```
tuple
```

In [45]:

```
oneItem_2 = ('Navin')
```

```
type(oneItem_2) # ERROR
```

Out[45]:

```
str
```

In [49]:

```
# Sequence Unpacking

#Tuple
my_test = 'Devjeet',21,'devjeetroy.dr@gmail.com'
name, roll, email = my_test
print(name)
print(roll)
print(email)
```

```
Devjeet
21
devjeetroy.dr@gmail.com
```

In [50]:

```
# List
my_testList= ['Devjeet',21,'devjeetroy.dr@gmail.com']
name, roll, email = my_testList
print(name)
print(roll)
print(email)
```

```
Devjeet
21
devjeetroy.dr@gmail.com
```

In [4]:

```
# Finding max, min and sum of the elements in tuple!
my_c = (1,2,3,4,5,6,7,8,9)
print(max(my_c))
print(min(my_c))
print(sum(my_c))
```

```
9
1
45
```

In [7]:

```
# Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object.
for i in enumerate(my_c):
    print(i)
```

```
(0, 1)
(1, 2)
(2, 3)
(3, 4)
(4, 5)
(5, 6)
(6, 7)
(7, 8)
(8, 9)
```

In [8]:

```
# Enumerate() method adds a counter to an iterable and returns it in a form of enumerate object.
for i,v in enumerate(my_c):
    print(i,"=",v)
```

```
0 = 1
```

```
1 = 2
2 = 3
3 = 4
4 = 5
5 = 6
6 = 7
7 = 8
8 = 9
```

Sets

Python also includes a data type for sets. A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

In [1]:

```
fruit_basket = {'apple','banana','mango','apple','grapes','mango'}
print(fruit_basket)
# All duplicates have been removed.
```

```
{'banana', 'mango', 'apple', 'grapes'}
```

In [3]:

```
# Testing if the elements are present in our set
'mango' in fruit_basket # True means present and fals means absent
```

Out[3]:

```
True
```

In [4]:

```
'guava' in fruit_basket
```

Out[4]:

```
False
```

In [6]:

```
# Different types of set operation
# Playing with names
a = set('devjeet')
b = set ('navin')
print(a,b)
```

```
{'j', 'v', 'e', 'd', 't'} {'i', 'a', 'v', 'n'}
```

In [7]:

```
# Unique letters in a
print(a)
```

```
{'j', 'v', 'e', 'd', 't'}
```

In [10]:

```
sorted(a)
a
```

Out[10]:

```
{'d', 'e', 'j', 't', 'v'}
```

In [11]:

```
sorted(b)
b
```

Out[11]:

```
{'a', 'i', 'n', 'v'}
```

In [12]:

```
# Set Functions and Operations

# Letters in a but not in b
print(a-b)
```

```
{'j', 'e', 'd', 't'}
```

In [13]:

```
# Letters in b but not in a
print(b-a)
```

```
{'i', 'a', 'n'}
```

In [14]:

```
# Letters in a or b or both
print(a|b)
```

```
{'j', 'i', 'v', 'e', 'd', 'a', 't', 'n'}
```

In [15]:

```
# Letters in both a and b
print(a & b)
```

```
{'v'}
```

In [16]:

```
# Letters in a or b but not both
print(a^b)
```

```
{'i', 'j', 'e', 'd', 'a', 't', 'n'}
```

In [18]:

```
# Like set comprehensions, set comprehensions are also present
my_set ={x for x in 'devjeet' if x not in 'navin'}
print(my_set)
```

```
{'j', 'e', 'd', 't'}
```

In [19]:

```
# Adding elements to sets
# Only one element at a time can be added to the set by using add() method,
# loops are used to add multiple elements at a time with the use of add() method.
my_set.add(8)
my_set
```

Out[19]:

```
{8, 'd', 'e', 'j', 't'}
```

In [20]:

```
my_set.add((10,9))
my_set
```

Out[20]:

```
{(10, 9), 8, 'd', 'e', 'j', 't'}
```

In [21]:

```
# Lists cannot be added in sets as they are mutable and hence not hashable.
my_set.add([3,5,6,87])
my_set
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-b64930a0db7d> in <module>
      1 # Lists cannot be added in sets as they are mutable and hence not hashable.
----> 2 my_set.add([3,5,6,87])
      3 my_set

TypeError: unhashable type: 'list'
```

In [22]:

```
# But list can be added to set using update() method.
# The update() method accepts lists, strings, tuples as well as other sets as its arguments.
my_set.update([3,5,6,87])
my_set
```

Out[22]:

```
{(10, 9), 3, 5, 6, 8, 87, 'd', 'e', 'j', 't'}
```

In [12]:

```
# Accessing a set via a loop
my_f = {'a','b','c','d','e','f','g','h'}
for i in my_f:
    print(i)
```

```
a
d
e
f
c
b
h
g
```

In [14]:

```
# Removing element from set using remove() & discard() method.
# If element which is supposed to be deleted is present in set, both work the same!
# But if an element is not present, remove() throws a KeyError but that doesn't happen with discar
d()
my_f.remove('a')
my_f
```

Out[14]:

```
{'b', 'c', 'd', 'e', 'f', 'g', 'h'}
```

In [15]:

```
my_f.remove('a')
```

```
my_f # An ERROR will be throw as a is absent
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-15-893c02566c72> in <module>
----> 1 my_f.remove('a')
      2 my_f # An ERROR will be throw as a is absent

KeyError: 'a'
```

In [16]:

```
my_f.remove('b')
my_f
```

Out[16]:

```
{'c', 'd', 'e', 'f', 'g', 'h'}
```

In [17]:

```
my_f.discard('b') # Though b is not present an ERROR doesn't arise.
my_f
```

Out[17]:

```
{'c', 'd', 'e', 'f', 'g', 'h'}
```

In [20]:

```
# pop() method to delete an element from a set
myTodo = {1,2,3,4,5,6,7,8,9}
myTodo.pop()
myTodo
```

Out[20]:

```
{2, 3, 4, 5, 6, 7, 8, 9}
```

In [21]:

```
# pop() takes no argument.
myTodo.pop(2)
myTodo
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-416ba9db3dbd> in <module>
----> 1 myTodo.pop(2)
      2 myTodo

TypeError: pop() takes no arguments (1 given)
```

In [24]:

```
myTodo.clear()
myTodo # Deletes all the elements from the set.
```

Out[24]:

```
set()
```

FrozenSet

In [25]:

```
myPC = { 'Monitor','CPU','Keyboard','Mouse','Headset','WebCam'}
myPC_frozen = frozenset(myPC)
myPC_frozen = frozenset(myPC)
myPC_frozen
```

Out[25]:

```
frozenset({'CPU', 'Headset', 'Keyboard', 'Monitor', 'Mouse', 'WebCam'})
```

Different & important set methods.

In [26]:

```
myMet = {'a','b','c','d'}
# add() adds an element to the set
myMet.add('e')
myMet
```

Out[26]:

```
{'a', 'b', 'c', 'd', 'e'}
```

In [27]:

```
# remove() removes an element from the set.
myMet.remove('e')
myMet
# Puts up a KeyError if element is not present.
```

Out[27]:

```
{'a', 'b', 'c', 'd'}
```

In [28]:

```
# clear() method empties the set
myMet.clear()
myMet
```

Out[28]:

```
set()
```

In [29]:

```
# copy() returns a shallow copy of the set to another.
myMet = {'a','b','c','d'}
myCopy = myMet.copy()
myCopy
```

Out[29]:

```
{'a', 'b', 'c', 'd'}
```

In [30]:

```
# pop() removes arbitary element from the set.
myMet.pop()
myMet
```

Out[30]:

```
{'a', 'b', 'c'}
```

In [31]:

```
# update() updates the set with union to itself or some another set.
myMet.update('d')
myMet
```

Out[31]:

```
{'a', 'b', 'c', 'd'}
```

In [32]:

```
# union() returns the union of 2 sets in a new set.
myNum={1, 2, 3, 4}
myTot = myMet.union(myNum)
myTot
```

Out[32]:

```
{1, 2, 3, 4, 'a', 'b', 'c', 'd'}
```

In [35]:

```
# different() returns the difference of 2 sets in a new set.
MyNew = myTot.difference(myMet)
MyNew
```

Out[35]:

```
{1, 2, 3, 4}
```

In [38]:

```
# difference_update() removes all elements of another set from this set
myTot.difference_update(MyNew)
myTot
```

Out[38]:

```
{'a', 'b', 'c', 'd'}
```

In [ ]:

```
# Other examples are
# intersection() Returns the intersection of two sets as a new set
# intersection_update() Updates the set with the intersection of itself and another
# isdisjoint() Returns True if two sets have a null intersection
# issubset() Returns True if another set contains this set
# issuperset() Returns True if this set contains another set
# symmetric_difference() Returns the symmetric difference of two sets as a new set
# symmetric_difference_update() Updates a set with the symmetric difference of itself and another
```

Dictionary

Dictionary in Python holds data as Key-Value Pair.

In [39]:

```
# Creating a Dictionary with Integer Keys
Dict = {1: 'Devjeet', 2: 'Navin', 3: 'Vikas'}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)

# Creating a Dictionary with Mixed keys
Dict = {'Name': 'Devjeet', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict)
```

```

Dictionary with the use of Integer Keys:
{1: 'Devjeet', 2: 'Navin', 3: 'Vikas'}

Dictionary with the use of Mixed Keys:
{'Name': 'Devjeet', 1: [1, 2, 3, 4]}
```

In [40]:

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Creating a Dictionary with dict() method
Dict = dict({1: 'C', 2: 'C++', 3:'Java'})
print("\nDictionary with the use of dict(): ")
print(Dict)

# Creating a Dictionary with each item as a Pair
Dict = dict([(1, 'Python'), (2, 'JS')])
print("\nDictionary with each item as a pair: ")
print(Dict)
```

```
Empty Dictionary:
{}

Dictionary with the use of dict():
{1: 'C', 2: 'C++', 3: 'Java'}

Dictionary with each item as a pair:
{1: 'Python', 2: 'JS'}
```

In [41]:

```
# Creating a Nested Dictionary
# as shown in the below image
Dict = {1: 'Devjeet', 2: 'Roy',
  3:{'Coding' : 'Python', 'Web' : 'Node.js', 'Database' : 'MongoDB'}}

print(Dict)
```

```
{1: 'Devjeet', 2: 'Roy', 3: {'Coding': 'Python', 'Web': 'Node.js', 'Database': 'MongoDB'}}
```

In [42]:

```
# Accessing an element in dictionary.
print(Dict[3])
```

```
{'Coding': 'Python', 'Web': 'Node.js', 'Database': 'MongoDB'}
```

In [43]:

```
print(Dict[3]['Coding'])
```

```
Python
```

In [44]:

```
# Removing an element from dictionary using del keyword.
del Dict[3]['Database']
Dict
```

Out[44]:

```
{1: 'Devjeet', 2: 'Roy', 3: {'Coding': 'Python', 'Web': 'Node.js'}}
```

In [48]:

```
# Removing an element from the dictionary using pop().
Dict.pop(3,'Web')
```

Out[48]:

```
{'Coding': 'Python', 'Web': 'Node.js'}
```

In [49]:

```
Dict
```

Out[49]:

```
{1: 'Devjeet', 2: 'Roy'}
```

In [50]:

```
# Empty the whole dictionary.
Dict.clear()
Dict
```

Out[50]:

```
{}
```

Array Fundamentals

In [2]:

```
# An array is a linear sequence of similar data.
# Arrays can be handles in Python 3.x by array module
'''
'b' signed char int 1
'B' unsigned char int 1
'u' Py_UNICODE unicode character 2
'h' signed short int 2
'H' unsigned short int 2
'i' signed int int 2
'I' unsigned int int 2
'l' signed long int 4
'L' unsigned long int 4
'q' signed long long int 8
'Q' unsigned long long int 8
'f' float float 4
'd' double float 8
'''
import array
myArr = array.array('i',[1,2,3,3,4,5,65,67,7,8])
print(myArr)
```

```
array('i', [1, 2, 3, 3, 4, 5, 65, 67, 7, 8])
```

In [4]:

```
for i in myArr:
    print(i, end=" ")
```

```
1 2 3 3 4 5 65 67 7 8
```

In [5]:

```
# using append() to insert new value at end
myArr.append(202)
myArr
```

Out[5]:

```
array('i', [1, 2, 3, 3, 4, 5, 65, 67, 7, 8, 202])
```

In [6]:

```
# using insert() to insert value at specific position.
myArr.insert(2,99)
myArr
```

Out[6]:

```
array('i', [1, 2, 99, 3, 3, 4, 5, 65, 67, 7, 8, 202])
```

In [7]:

```
# Printing after insertion
for i in myArr:
    print(i, end =" ")
```

```
1 2 99 3 3 4 5 65 67 7 8 202
```

In [8]:

```
# The pop() function removes the element at the position mentioned in its argument, and returns it
.
myArr.pop(2)
```

Out[8]:

```
99
```

In [9]:

```
# remove() function is used to remove the first occurrence of the value mentioned in its arguments
.
myArr.remove(4)
myArr
```

Out[9]:

```
array('i', [1, 2, 3, 3, 5, 65, 67, 7, 8, 202])
```

In [10]:

```
# index() function returns the index of the first occurrence of value mentioned in arguments.
myArr.index(202)
```

Out[10]:

```
9
```

In [11]:

```
myArr.index(502) # Returns ValueError if element is not present.
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-9e186332d1db> in <module>
----> 1 myArr.index(502)

ValueError: array.index(x): x not in array
```

In [12]:

```
# reverse() function reverses the array.
myArr.reverse()
```

In [13]:

```
myArr
```

Out[13]:

```
array('i', [202, 8, 7, 67, 65, 5, 3, 3, 2, 1])
```

Some important functions of Array in Python

In [14]:

```
# typecode :- This function returns the data type by which array is initialised.
myArr.typecode
```

Out[14]:

```
'i'
```

In [15]:

```
# itemsize :- This function returns size in bytes of a single array element.
myArr.itemsize
```

Out[15]:

```
4
```

In [16]:

```
# buffer_info() :- Returns a tuple representing the address in which array is stored and number of
elements in it.
myArr.buffer_info()
```

Out[16]:

```
(322198518080, 10)
```

In [19]:

```
# count() :- This function counts the number of occurrences of argument mentioned in array.
myArr.count(3)
```

Out[19]:

```
2
```

In [21]:

```
# extend(arr) :- This function appends a whole array mentioned in its arguments to the specified a
rray.
arr2 = array.array('i',[22,33,44,55])
myArr.extend(arr2)
myArr
```

Out[21]:

```
array('i', [202, 8, 7, 67, 65, 5, 3, 3, 2, 1, 22, 33, 44, 55])
```

In [22]:

```
# Convert an array to a list
l1 = myArr.tolist()
l1
```

Out[22]:

```
[202, 8, 7, 67, 65, 5, 3, 3, 2, 1, 22, 33, 44, 55]
```

In [ ]:

```

```