Unit-5 Mutable Data Structures

Mutable Data structure:

Mutable data structures are those whose state can bemodified as per our need.

Example: list, dictionary, Sets.

Definition of List:

- As the part of programming requirement, we have to store our data permanently forfuture purpose. For this requirement we should go for files.
- If we want to represent a group of individual objects as a single entity where insertionorder is preserved and duplicates and heterogeneous objects are allowed, then we should go for List.
- List is dynamic because based on our requirement we can increase the size anddecrease the size.
- In List the elements will be placed within square brackets and with comma separator.
- We can differentiate duplicate elements by using index and we can preserve insertionorder by using index. Hence index will play very important role.
- Python supports both positive and negative indexes. +ve index means from left to rightwhere as negative index means right to left.
- List objects are mutable. i.e we can change the content.

Creation of List:

(1) We can create empty list object as follows:

- **Example:**
- list=[]
- print(list)
- print(type(list))
- **Output:**
- []
- <class 'list'>

(2) If we know elements already then we can create list as follows:

- **Example:**
- list = [1, 2, 3, 4]
- print(list)

- print(type(list))
- **Output:**
- [1, 2, 3, 4]
- <class 'list’>

(3) Creation of List with Dynamic Input:

- **Example:**
- list=eval(input("Enter List:"))
- print(list)
- print(type(list))
- **Output:**
- Enter List: [5,6,7,8]
- [5, 6, 7, 8]
- <class 'list’>

[**Note**: eval is a built-in- function used in python, **eval function parses the expression argument and evaluates it as a python expression**. In simple words, the eval function evaluates the “String” like a python expression and returns the result as an integer.]

(4) Creation Of List With list() Function:

- **Example 1:**
- l=list(range(0,12,2))
- print(l)
- print(type(l))
- **Output:**
- [0, 2, 4, 6, 8,10]
- <class 'list'>

- **Example 2:**
- s=“arman"
- l=list(s)
- print(l)
- print(type(l))
- **Output:**
- [‘a’, ‘r’, ‘m’, ‘a’, ‘n’]
- <class 'list'>

(5) Creation Of List With split() Function:

- **Example:**
- s="Learning Python is very easy !!!"
- l=s.split()
- print(l)
- print(type(l))
- **Output:**
- ['Learning', 'Python', 'is', 'very', 'easy', '!!!’]
- <class 'list'>

List Vs Mutability:

- Once we create a List object, we can modify its content. Hence List objects are mutable.
- **Example:**
- n=[10,20,30,40]
- print(n)
- n[1]=77
- print(n)

- **Output:**
- [10, 20, 30, 40]
- [10, 77, 30, 40]

Accessing Elements of List:

(1) By Using Index:

- List follows zero based index. ie index of first element is zero.
- List supports both +ve and -ve indexes.
- +ve index meant for Left to Right.
- -ve index meant for Right to Left.
- **Example:**
- list = [10, 20, 30, 40]
- print(list[0])
- print(list[-1])
- print(list[10])
- **Output:**
- 10
- 40
- IndexError: list index out of range

(2) By Using Slice Operator:

- Syntax: list2 = list1[start:stop:step]
- Start: It indicates the Index where slice has to Start and Default Value is 0.
- Stop: It indicates the Index where slice has to End and Default Value is max allowed Index of List ie Length of the List.
- Step: Increment value and Default Value is 1.
- **Example:**
- n=[1,2,3,4,5,6,7,8,9,10]
- print(n[2:7:2])
- print(n[4::2])
- print(n[3:7])

- print(n[8:2:-2])
- print(n[4:100])
- **Output:**
- [3, 5, 7]
- [5, 7, 9]
- [4, 5, 6, 7]
- [9, 7, 5]
- [5, 6, 7, 8, 9, 10]

Loop List:

(1) Loop through the list items by using while loop:

- **Example:**
- n = [0,1,2,3,4,5,6,7,8,9]
- i = 0
- while i < len(n):
- print(n[i])
- i=i+1
- **Output:**
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

(2) Loop through the list items by using for loop:

- **Example:**
- n=[0,1,2,3,4,5,6,7,8,9]
- for n1 in n:
- print(n1)

- **Output:**
- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

(3) Loop through the list items by Index numbers:

- **Example:**
- l = ["A","B","C"]
- x = len(l)
- for i in range(x):
- print(l[i])
- **Output:**
- A
- B
- C

Important Functions of List:

(1) len(): Returns the number of elements present in the list

- **Example:**
- n = [1, 2, 3, 4]
- print(len(n))
- **Output:**
- 4

(2) count(): It returns the number of occurrences of specified item in the list.

- **Example:**
- n=[1,2,2,2,2,3,3]
- print(n.count(1))

- print(n.count(2))
- print(n.count(3))
- print(n.count(4))
- **Output:**
- 1
- 4
- 2
- 0

(3) index(): Returns the index of first occurrence of the specified item.

- **Example:**
- n = [1, 2, 2, 2, 2, 3, 3]
- print(n.index(1))
- print(n.index(2))
- print(n.index(3))
- print(n.index(4))
- **Output:**
- 0
- 1
- 5
- ValueError: 4 is not in list
- If the specified element is not present in the list then we will get ValueError.
- Hence before index() method we have to check whether item present in the list or notby using in operator.

- **Example:**
- print(4 in n) # False

(4) append() Function: We can use append() function to add item at the end of the list.

- **Example:**
- list=[“M”,“N”]
- list.append("A")
- list.append("B")
- list.append("C")
- print(list)
- **Output:**
- [‘M’,‘N’,A', 'B', 'C']

(5) insert() Function: To insert item at specified index position.

- **Example:**
- n=[1,2,3,4,5]
- n.insert(1,28)
- print(n)
- **Output:**
- [1, 28, 2, 3, 4, 5]
- If the specified index is greater than max index then element will be inserted at lastposition.
- If the specified index is smaller than min index then element will be inserted at firstposition.
- **Example:**
- n=[1,2,3,4,5]
- n.insert(10,7)

- n.insert(-10,9)
- print(n)
- **Output:**
- [9, 1, 2, 3, 4, 5, 7]

(6) extend() Function: To add all items of one list to another list.

- l1.extend(l2) all items present in l2 will be added to l1.
- **Example 1:**
- l1=[“Apple",“Mango"]
- l2=[“Orange",“Water Melon "]
- l1.extend(l2)
- print(l1)
- **Output:**
- [‘Apple’, ‘Mango’, ‘Orange’, ‘Water Melon’]
- **Example 2:**
- l1 = [“Amit",“Sumit"]
- l1.extend(“kumar")
- print(l1)
- **Output:**
- [‘Amit', ‘Sumit', ‘K', 'u', ‘m', ‘a', 'r']

(7) remove() Function: We can use this function to remove specified item from the list.

- If the item present multiple times then only first occurrence will be removed.
- **Example:**
- n=[1,2,1,3]
- n.remove(1)

- print(n)
- **Output:**
- [2, 1, 3]
- If the specified item is not present in list then we will get ValueError.
- **Example:**
- y=[10,20,10,30]
- y.remove(40)
- print(n)
- **Output:**
- ValueError: list.remove(x): x not in list
- Hence before using remove() method first we have to check specified elementpresent in the list or not by using in operator.

(8) **pop() Function:** It removes and returns the last element of the list.

- This is only function which manipulates list and returns some element.
- **Example:**
- n=[10,20,30,40]
- print(n.pop())
- print(n.pop())
- print(n)
- **Output:**
- 40
- 30
- [10, 20]
- If the list is empty then pop() function raises IndexError.

- **Example:**
- y = []
- print(y.pop())
- **Output:**
- IndexError: pop from empty list
- In general we can use pop() function to remove last element of the list. But we can useto remove elements based on index.
- n.pop(index): To remove and return element present at specified index.
- n.pop(): To remove and return last element of the list.
- **Example:**
- n = [10,20,30,40,50,60]
- print(n.pop())
- print(n.pop(1))
- print(n.pop(10))
- **Output:**
- 60
- 20
- IndexError: pop index out of range.

(9) **clear() Function:** We can use clear() function to remove all elements of List.

- **Example:**
- n=[1,2,3,4]
- print(n)
- n.clear()
- print(n)
- **Output:**
- [1, 2, 3, 4]
- []

(10) **reverse():** It is used to reverse the order of elements of list.

- **Example:**
- n=[1,2,3,4]
- n.reverse()
- print(n)
- **Output:**
- [4, 3, 2, 1]

(11) **sort():** In list by default insertion order is preserved.

- If want to sort the elements of list according to default natural sorting order then weshould go for sort() method.
- For numbers: Default Natural sorting Order is Ascending Order.
- For Strings : Default Natural sorting order is Alphabetical Order.
- **Example 1:**
- n = [2,5,15,1,0]
- n.sort()
- print(n)
- **Output:**
- [0,1,2,5,15]
- **Example 2:**
- s = ["D","B","C","A"]
- s.sort()
- print(s)
- **Output:**
- [“A","B","C",“D"]

- To use sort() function, compulsory list should contain only homogeneous elements.Otherwise we will get TypeError.
- **Example:**
- n=[20,10,"A","B"]
- n.sort()
- print(n)
- **Output:**
- TypeError: '<' not supported between instances of 'str' and 'int'
- **To Sort in Reverse of Default Natural Sorting Order:** We can sort according to reverse of default natural sorting order by using reverse=True argument.
- **Example 1:**
- n = [4,1,3,2]
- n.sort(reverse = True)
- print(n)
- **Output:**
- [4,3,2,1]
- **Example 2:**
- n = [4,1,3,2]
- n.sort(reverse = False)
- print(n)
- **Output:**
- [1,2,3,4]

Mathematical Operators For List:

- We can use + and * operators for List objects.

(1) Concatenation Operator (+): We can use + to concatenate 2 lists into a single list.

- **Example:**
- a = [1, 2, 3]
- b = [4, 5, 6]
- c = a+b
- print(c)
- **Output:**
- [1, 2, 3, 4, 5, 6]
- To use + operator compulsory both arguments should be list objects, otherwise we willget TypeError.
- **Example:**
- a = [1, 2, 3]
- b = a+[4]
- print(b)
- c = a+4
- print(c)
- **Output:**
- [1, 2, 3,4]
- TypeError: can only concatenate list (not "int") to list.

(2) Repetition Operator (*): We can use repetition operator * to repeat elements of listspecified number of times.

- **Example:**
- x = [1, 2, 3]
- y = x*3
- print(y)
- **Output:**
- [1, 2, 3, 1, 2, 3, 1, 2, 3]

Comparison Operators For List:

- We can use comparison operators for List objects.
- Whenever we are using comparison operators (==, !=) for List objects then thefollowing should be considered
- (1) The Number of Elements
- (2) The Order of Elements
- (3) The Content of Elements (Case Sensitive)
- **Example:**
- x = ["Dog", "Cat", "Rat"]
- y = ["Dog", "Cat", "Rat"]
- z = ["DOG", "CAT", "RAT"]
- print(x == y)
- print(x == z)
- print(x != z)
- **Output:**
- True
- False
- True
- Whenever we are using relaxational Operators (<, <=, >, >=) between List Objects, only 1st Element comparison will be performed.
- **Example 1:**
- x = [50, 20, 30]
- y = [40, 50, 60, 100, 200]

- print(x>y)
- print(x>=y)
- print(x<y)
- print(x<=y)
- **Output:**
- True
- True
- False
- False
- **Example 2:**
- x = ["Dog", "Cat", "Rat"]
- y=["Rat", "Cat", "Dog"]
- print(x>y)
- print(x>=y)
- rint(x<y)
- print(x<=y)
- **Output:**
- False
- False
- True
- True

Membership Operators For List:

- We can check whether element is a member of the list or not by using membershipoperators.

- **(1) in Operator**
- **(2) not in Operator**
- **Example:**
- n=[10,20,30,40]
- print (10 in n)
- print (10 not in n)
- print (50 in n)
- print (50 not in n)
- **Output:**
- True
- False
- False
- True

Aliasing and Cloning of List:

- The process of giving another reference variable to the existing list is called aliasing.
- **Example:**
- x=[10,20,30,40]
- y=x
- print(x)
- print(y)
- **Output:**
- [10,20,30,40]
- [10,20,30,40]

- The problem in this approach is by using one reference variable if we are changingcontent, then those changes will be reflected to the other reference variable.
- **Example:**
- x = [10,20,30,40]
- y = x
- y[1] = 777
- print(x)
- **Output:**
- [10,777,30,40]
- To overcome this problem, we should go for cloning. The process of creating exactly duplicate independent object is called cloning.
- We can implement cloning by using following methods:
- **(1) By using slice operator**
- **(2) By using copy() function**.

(1) By using Slice Operator:

- x = [10,20,30,40]
- y = x[:]
- y[1] = 777
- print(x)
- print(y)
- **Output:**
- [10,20,30,40]
- [10,777,30,40]

(2) By using copy() Function:

- x = [10,20,30,40]
- y = x.copy()
- y[1] = 777
- print(x)
- print(y)
- **Output:**
- [10,20,30,40]
- [10,777,30,40]

Nested Lists:

- Sometimes we can take one list inside another list. Such type of lists are called nestedlists.
- **Example:**
- n=[10,20,[30,40]]
- print(n)
- print(n[0])
- print(n[2])
- print(n[2][0])
- print(n[2][1])
- **Output:**
- [10, 20, [30, 40]]
- 10
- [30, 40]
- 30
- 40

Nested List As Matrix:

- We can represent matrix by using nested lists.
- **Example:**
- n=[[10,20,30],[40,50,60],[70,80,90]]
- print(n)
- print("Elements by Row wise:")
- for r in n:
- print(r)
- print("Elements by Matrix style:")
- for i in range(len(n)):
- for j in range(len(n[i])):
- print(n[i][j],end=' ')
- print()
- **Output:**
- [[10, 20, 30], [40, 50, 60], [70, 80, 90]]
- Elements by Row wise:
- [10, 20, 30]
- [40, 50, 60]
- [70, 80, 90]
- Elements by Matrix style:
- 10 20 30
- 40 50 60
- 70 80 90

List Comprehensions:

- It is very easy and compact way of creating list objects from any inerrable objects(Like List, Tuple, Dictionary, Range etc) based on some condition.
- **Example:**
- Syntax: list = [expression for item in list if condition]
- s = [x*x for x in range (1,11)]
- print(s)
- v = [2**x for x in range (1,6)]
- print(v)
- m = [x for x in s if x%2==0]
- print(m)
- **Output:**
- [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
- [2, 4, 8, 16, 32]
- [4, 16, 36, 64, 100]

Definition of Dictionary:

- If we want to represent a group of objects as key-value pairs then we should go forDictionary.
- **Example:**
- rollno----- name
- phone number – address
- Duplicate keys are not allowed but values can be duplicated.
- Heterogeneous objects are allowed for both key and values.
- Insertion order is not preserved.
- Dictionaries are mutable.
- indexing and slicing concepts are not applicable.

Creation of Dictionary:

- We can create empty dictionary in the following way.
- d = {} OR d = dict()
- We can add entries as follows.
- **Example:**
- d = {}
- d[10]="Lucky"
- d[20]="Arman"
- d[30]="Dhairya"
- print(d)
- **Output:**
- 10: 'Lucky', 20: 'Arman', 30: 'Dhairya'}
- If we know data in advance then we can create dictionary as follows.
- d = {10:'Lucky' ,20:'Arman', 30:'Dhairya'}
- d = {key:value, key:value, key:value}

Accessing Elements of Dictionary:

- We can access data by using keys.
- **Example:**
- d = {10:'Lucky',20:'Arman', 30:'Dhairya'}
- print(d[10])
- print(d[20])
- print(d[40])
- **Output:**
- Lucky
- Arman
- KeyError: 40

- If the specified key is not available then we will get KeyError.
- We can prevent this by checking whether key is already available or not by using inoperator.
- **Example:**
- d = {10:’Lucky',20:’Arman', 30:’Dhairya’}
- if 40 in d:
- print(d[40])

Updating Elements of Dictionary:

- d[key] = value
- If the key is not available then a new entry will be added to the dictionary with thespecified key-value pair.
- If the key is already available then old value will be replaced with new value.
- **Example:**
- d = {10:’Lucky',20:’Arman', 30:’Dhairya’}
- print(d)
- d[40]=“Sumit"

- print(d)
- d[10]=“Sunny"
- print(d)
- **Output:**
- {10:’Lucky',20:’Arman', 30:’Dhairya’}
- {10:’Lucky',20:’Arman', 30:’Dhairya’, 40: ‘Sumit’}
- {10:’Sunny',20:’Arman', 30:’Dhairya’, 40: ‘Sumit’}

Deleting Elements of Dictionary:

(1) del d[key]: It deletes entry associated with the specified key.

- If the key is not available then we will get KeyError.
- **Example:**
- d= {10:’Lucky',20:’Arman', 30:’Dhairya’}
- print(d)
- del d[10]
- print(d)
- del d[40]
- **Output:**
- {10:’Lucky',20:’Arman', 30:’Dhairya’}
- {20: ‘Arman', 30: ‘Dhairya’}
- KeyError: 40

(2) d.clear(): To remove all entries from the dictionary.

- **Example:**
- d= {10:’Lucky',20:’Arman', 30:’Dhairya’}

- print(d)
- d.clear()
- print(d)
- **Output:**
- {10:’Lucky',20:’Arman', 30:’Dhairya’}
- {}

(3) del d: To delete total dictionary. Now we cannot access d.

- **Example:**
- d= {10:’Lucky',20:’Arman', 30:’Dhairya’}
- print(d)
- del d
- print(d)
- **Output:**
- {10:’Lucky',20:’Arman', 30:’Dhairya’}
- NameError: name 'd' is not defined

Dictionary Comprehension:

- It is very easy and compact way of creating objects from any inerrable objects (Like List, Tuple, Dictionary, Range etc) based on some condition.
- Syntax: list = {expression for item in list if condition}
- **Example:**
- squares={x:x*x for x in range(1,6)}
- print(squares)
- doubles={x:2*x for x in range(1,6)}
- print(doubles)

- **Output:**
- {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
- {1: 2, 2: 4, 3: 6, 4: 8, 5: 10}

Important Functions of Dictionary:

(1) dict(): To create a dictionary

- d = dict(): It creates empty dictionary.
- d = dict({10:“Arman",20:“Dhairya"}): It creates dictionary with specified elements.
- d = dict([(10,“Lucky"),(20,“Arman"),(30,“Dhairya")]):
- It creates dictionary with the given list of tuple elements

(2) len(): Returns the number of items in the dictionary.

(3) clear(): To remove all elements from the dictionary.

(4) copy(): To create exactly duplicate dictionary (cloned copy)

- d1 = d.copy()

(5) update():

- d.update(x): All items present in the dictionary x will be added to dictionary d.

(6) get(): To get the value associated with the key.

- d.get(key): If the key is available then returns the corresponding value otherwisereturns None. It won’t raise any error.
- d.get(key,defaultvalue) If the key is available then returns the corresponding valueotherwise returns default value.
- **Example:**
- d={10:“Lucky",20:“Arman",30:“Dhairya"}
- print(d[10])
- print(d.get(10))

- print(d.get(40))
- print(d.get(10,"Guest"))
- print(d.get(40,"Guest"))
- **Output:**
- Lucky
- Lucky
- None
- Lucky
- Guest

(7) popitem(): It removes an arbitrary item(key-value) from the dictionary and returns it.

- If the dictionary is empty then we will get KeyError
- **Example:**
- d={10:“Lucky",20:“Arman",30:“Dhairya"}
- print(d)
- print(d.popitem())
- print(d)
- d={}
- print(d.popitem())
- **Output:**
- {10:“Lucky",20:“Arman",30:“Dhairya"}
- (30, ‘Dhairya’)
- {10: ‘Lucky', 20: ‘Arman'}
- KeyError: 'popitem(): dictionary is empty'

(8) keys(): It returns all keys associated with dictionary.

- **Example:**
- d={10:“Lucky",20:“Arman"}
- print(d.keys())
- for k in d.keys():
- print(k)
- **Output:**
- dict_keys([10, 20])
- 10
- 20

(9) values(): It returns all values associated with the dictionary.

- **Example:**
- d= {10:“Lucky",20:“Arman",30:“Dhairya"}
- print(d.values())
- for v in d.values():
- print(v)
- **Output:**
- dict_values([‘Lucky', ‘Arman', ‘Dhairya'])
- Lucky
- Arman
- Dhairya

(10) items(): It returns list of tuples representing key-value pairs. [(k,v),(k,v),(k,v)]

- **Example:**
- d= {10:“Lucky",20:“Arman",30:“Dhairya"}
- for k,v in d.items():

- print(k,"--",v)
- **Output:**
- 10 -- Lucky
- 20 -- Arman
- 30 -- Dhairya

(11) setdefault():

- d.setdefault(k,v): If the key is already available then this function returns thecorresponding value.
- If the key is not available then the specified key-value will be added as new item to thedictionary.
- **Example:**
- d= {10:“Lucky",20:“Arman",30:“Dhairya"}
- print(d.setdefault(40,"pavan"))
- print(d)
- print(d.setdefault(10,"sachin"))
- **Output:**
- pavan
- {10: ‘Lucky', 20: ‘Arman', 30: ‘Dhairya', 40: 'pavan’}
- Lucky

Definition of Set:

- If we want to represent a group of unique values as a single entity then we should gofor set.
- Duplicates are not allowed.
- Insertion order is not preserved. But we can sort the elements.

- Indexing and slicing not allowed for the set.
- Heterogeneous elements are allowed.
- Set objects are mutable i.e once we creates set object we can perform any changes inthat object based on our requirement.
- We can represent set elements within curly braces and with comma separation.
- We can apply mathematical operations like union, intersection, difference etc on setobjects.

Creation of Set:

- We can create set as follows:
- **Example:**
- s={10,20,30,40}
- print(s)
- print(type(s))
- **Output:**
- {40, 10, 20, 30}
- <class 'set'>
- We can create set objects by using set() Function.
- s = set(any sequence)
- **Example 1:**
- l = [10,20,30,40,10,20,10]
- s=set(l)
- print(s)
- **Output:**
- {40, 10, 20, 30}

- **Example 2:**
- s=set(range(5))
- print(s)
- **Output:**
- {0, 1, 2, 3, 4}
- While creating empty set we have to take special care.
- Compulsory we should use set() function.
- s = {}: It is treated as dictionary but not empty set.
- **Example 1:**
- s={}
- print(s)
- print(type(s))
- **Output:**
- {}
- <class 'dict'>
- **Example 2:**
- s=set()
- print(s)
- print(type(s))
- **Output:**
- set()
- <class 'set'>

Mathematical Operations On The Set:

(1) union():

- x.union(y): We can use this function to return all elements present in both sets.
- x.union(y) OR x|y.
- **Example:**
- x = {10, 20, 30, 40}
- y = {30, 40, 50, 60}
- print (x.union(y))
- print (x|y)
- **Output:**
- {10, 20, 30, 40, 50, 60}
- {10, 20, 30, 40, 50, 60}

(2) intersection():

- x.intersection(y) OR x&y: Returns common elements present in both x and y.
- **Example:**
- x = {10, 20, 30, 40}
- y = {30, 40, 50, 60}
- print (x.intersection(y))
- print(x&y)
- **Output:**
- {40, 30}
- {40, 30}

(3) difference():

- x.difference(y) OR x-y: Returns the elements present in x but not in y.
- **Example:**

- x = {10, 20, 30, 40}
- y = {30, 40, 50, 60}
- print (x.difference(y))
- print (x-y)
- print (y-x)
- **Output:**
- {10, 20}
- {10, 20}
- {50, 60}

(4) symmetric_difference():

- x.symmetric_difference(y) OR x^y: Returns elements present in either x OR y but not in both.
- **Example:**
- x = {10, 20, 30, 40}
- y = {30, 40, 50, 60}
- print (x.symmetric_difference(y))
- print(x^y)
- **Output:**
- x = {10, 50, 20, 60}
- x = {10, 50, 20, 60}

(5) issubset():

- Return True if all items in set x are present in set y
- **Example:**

- x = {"a", "b", "c"}
- y = {"f", "e", "d", "c", "b", "a"}
- z = x.issubset(y)
- print(z)
- **Output:**
- True

(6) issuperset():

- Return True if all items set y are present in set x
- **Example:**
- y = {"a", "b", "c"}
- x = {"f", "e", "d", "c", "b", "a"}
- z = x.issuperset(y)
- print(z)
- **Output:**
- True

Membership Operators In Set :

- We can check whether element is a member of the list or not by using memebership operators.
- (1) in Operator
- (2) not in Operator
- **Example:**
- s=set("durga")
- print(s)
- print('d' in s)
- print('z' in s)

- **Output:**
- {'d','u', 'r' 'g', 'a'}
- True
- False

Important Functions of Set:

(1) add(x): Adds item x to the set.

- **Example:**
- s={10,20,30}
- s.add(40)
- print(s)
- **Output:**
- {40, 10, 20, 30}

2) update(x,y,z): To add multiple items to the set.

- Arguments are not individual elements and these are Inerrable objects like List, Range etc.
- All elements present in the given Iterable objects will be added to the set.
- **Example:**
- s={10,20,30}
- l=[40,50,60,10]
- s.update(l,range(5))
- print(s)
- **Output:**
- {0, 1, 2, 3, 4, 40, 10, 50, 20, 60, 30}

(3) copy(): Returns copy of the set.

- It is cloned object.
- **Example:**
- s = {10,20,30}
- s1 = s.copy()
- print(s)
- print(s1)
- **Output:**
- {10,20,30}
- {10,20,30}

(4) pop(): It removes and returns some random element from the set.

- **Example:**
- s={40,10,30,20}
- print(s)
- print(s.pop())
- print(s)
- **Output:**
- {40, 10, 20, 30}
- 40
- {10, 20, 30}

(5) remove(x): It removes specified element from the set.

- If the specified element is not present in the Set then we will get KeyError.
- **Example:**
- s={40, 10, 30, 20}
- s.remove(30)
- print(s)
- s.remove(50)
- **Output:**
- { 40, 10, 20}
- KeyError: 50

(6) discard(x):

- It removes the specified element from the set.
- If the specified element not present in the set then we won't get any error.
- **Example:**
- s = {10, 20, 30}
- s.discard(10)
- print(s)
- s.discard(50)
- print(s)
- **Output:**
- {20, 30}

- {20, 30}

(7) clear(): To remove all elements from the Set.

- **Example:**
- s={10,20,30}
- print(s)
- s.clear()
- print(s)
- **Output:**
- {10, 20, 30}
- set()

Set Comprehension:

- It is very easy and compact way of creating objects from any inerrable objects (Like List, Tuple, Dictionary, Range etc) based on some condition.
- Syntax: list = {expression for item in list if condition}
- s={x*x for x in range(5)}
- print (s)
- s = {2**x for x in range(2,10,2)}
- print (s)
- **Output:**
- {0, 1, 4, 9, 16}
- {16, 256, 64, 4}

Lambda Function:

- Sometimes we can declare a function without any name, such type of nameless functions are called anonymous functions or lambda functions.

Normal Function: We can define by using def keyword.

- def squareIt(n):
- return n*n

Lambda Function: We can define Lambda Function by using lambda keyword.

- lambda n:n*n
- Syntax of lambda Function: lambda argument_list : expression
- By using Lambda Functions we can write very concise code so that readability of the program will be improved.
- **Lambda Function to find square of given number:**
- s=lambda n:n*n
- print("The Square of 4 is :",s(4))
- print("The Square of 5 is :",s(5))
- **Output:**
- The Square of 4 is : 16
- The Square of 5 is : 25

Lambda Function to find sum of given two numbers:

- s=lambda a,b:a+b
- print("The Sum of 10,20 is:",s(10,20))
- print("The Sum of 100,200 is:",s(100,200))
- **Output:**
- The Sum of 10,20 is: 30

- The Sum of 100,200 is: 300

- Lambda Function internally returns expression value and we are not required to writereturn statement explicitly.
- Sometimes we can pass function as argument to another function. In such caseslambda functions are best choice.
- We can use lambda functions very commonly with filter(), map() and reduce() functions, because these functions expect function as argument.

map() Function:

- For every element present in the given sequence, apply some functionality and generate new element with the required modification.
- For this requirement we should go for map() function.

- Eg: For every element present in the list perform double and generate new list of doubles.
- Syntax: map(function, sequence)

- The function can be applied on each element of sequence and generates new sequence.

Program to double the Numbers from the List by using map() Function (Without Lambda Function):

- l=[1,2,3,4,5]

- def doubleIt(x):

- return 2*x

- l1=list(map(doubleIt,l))

- print(l1)

- **Output:**

- [2, 4, 6, 8, 10]

Program to double the Numbers from the List by using map() Function(With Lambda Function):

- l=[1,2,3,4,5]

- l1=list(map(lambda x:2*x,l))
- print(l1)
- **Output:**
- [2, 4, 6, 8, 10]
- We can apply map() function on multiple lists also. But make sure all list should have same length.
- Syntax: map(lambda x,y:x*y,l1,l2))
- x is from l1 and y is from l2.
- **Example:**
- l1=[1,2,3,4]
- l2=[2,3,4,5]
- l3=list(map(lambda x,y:x*y,l1,l2))
- print(l3)
- **Output:**
- [2, 6, 12, 20]

reduce() Function:

- reduce() function reduces sequence of elements into a single element by applying thespecified function.
- Syntax: reduce(function,sequence)
- reduce() function present in functools module and hence we should write importstatement.

Program to add and multiply all the Numbers of the List by using reduce() Function(With Lambda Function):

- from functools import *
- l=[10,20,30,40,50]

- sum=reduce(lambda x,y:x+y,l)
- print(sum)
- mul=reduce(lambda x,y:x*y,l)
- print(mul)
- **Output:**
- 150
- 12000000

filter() Function:

- **filter() Function:** We can use filter() function to filter values from the given sequence based on some condition.
- filter(function,sequence): Where Function Argument is responsible to perform conditional check and sequence can be List OR Tuple OR String.

Program to filter only Even Numbers from the List by using filter() Function(Without Lambda Function):

```
def isEven(x):
    if x%2==0:
        return True
    else:
        return False
l=[0,5,10,15,20,25,30]
l1=list(filter(isEven,l))
print(l1)
```

- **Output:**
- [0,10,20,30]

Program to filter only Even Numbers from the List by using filter() Function(WithLambda Function):

- l=[0,5,10,15,20,25,30]
- l1=list(filter(lambda x:x%2==0,l))
- print(l1)
- **Output:**
- [0,10,20,30]

frozenset()

Python frozenset() Method creates an immutable Set object from an iterable. It is a built-in Python function. As it is a set object therefore we cannot have duplicate values in the frozenset.

frozenset() in Python

Syntax: frozenset(iterable_object_name)
Parameter: iterable_object_name

- This function accepts iterable object as input parameter.

Return: Returns an equivalent frozenset object.

Example:

```
In [15]: # creating a list
favourite_subject = ["OS", "DBMS", "Algo"]

# creating a frozenset
f_subject = frozenset(favourite_subject)

# below line will generate error
f_subject[1] = "Networking"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-7569b44828c1> in <module>
      6
      7 # below line will generate error
----> 8 f_subject[1] = "Networking"

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

In [16]:
```
mylist = ['apple', 'banana', 'cherry']
x = frozenset(mylist)
x.add('strawberry')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-16-d82416ec4831> in <module>
      1 mylist = ['apple', 'banana', 'cherry']
      2 x = frozenset(mylist)
----> 3 x.add('strawberry')

AttributeError: 'frozenset' object has no attribute 'add'
```