**Deque:**

It is a optimized list for removing and inserting the data.

**import:**

from collections import deque

**Syntax:**

collections.deque([iterable[, maxlen]])

Return:

it will return a newly created deque object which is from left to right with help of data from iterable,if iterable not mentioned ,then the new deque is empty

If maxlen is not mentioned or if it is none ,then it will considered the variable length

deque is supporting following methods

1.append(x):It will add the data end of the deque

2.appendleft(x):It will add the data start of the deque

3.insert(i,x):It will insert the data at particular index here i is index,x is data

4.pop():It will remove the last element in list

5.popleft():It will remove the first element in list

6.remove(x):Remove the value which is x

7.reverse:reverse the deque

3.clear:remove the all elements

4.copy:copy of the deque

5.count(x):count the no of time the x is occured in qeque

6.extend(iterable):Extend the deque at the end

**Example:**

from collections import deque

u\_deque = deque('Hello 123',)

for elem in u\_deque:

print(elem.upper())

u\_deque.append('x')

u\_deque.appendleft('y')

print(u\_deque)

u\_deque.pop()

u\_deque.popleft()

print(list(u\_deque))

print(u\_deque[0])

print(u\_deque[-1])

print(list(reversed(u\_deque)))

u\_deque.extend('456')

print(u\_deque)

u\_deque.rotate(1) # right rotation

print(u\_deque)

u\_deque.rotate(-1) # left rotation

print(u\_deque)

print(deque(reversed(u\_deque)))

print(u\_deque.pop())

u\_deque.extendleft('xyz')

print(u\_deque)

u\_deque.clear()

print(u\_deque)

**output:**

H

E

L

L

O

1

2

3

deque(['y', 'H', 'e', 'l', 'l', 'o', ' ', '1', '2', '3', 'x'])

['H', 'e', 'l', 'l', 'o', ' ', '1', '2', '3']

H

3

['3', '2', '1', ' ', 'o', 'l', 'l', 'e', 'H']

deque(['H', 'e', 'l', 'l', 'o', ' ', '1', '2', '3', '4', '5', '6'])

deque(['6', 'H', 'e', 'l', 'l', 'o', ' ', '1', '2', '3', '4', '5'])

deque(['H', 'e', 'l', 'l', 'o', ' ', '1', '2', '3', '4', '5', '6'])

deque(['6', '5', '4', '3', '2', '1', ' ', 'o', 'l', 'l', 'e', 'H'])

6

deque(['z', 'y', 'x', 'H', 'e', 'l', 'l', 'o', ' ', '1', '2', '3', '4', '5'])

deque([])

**ChainMap**

It is used to club the several mappings. or dictionaries , It will returns a list of dictionaries.

**Import:**

from collections import ChainMap

**Syntax:**

class collections.ChainMap(\*maps)

It is a groups of multiple of other mappings or dicts by together it will create single, updateable or modfies view. If maps are not specified, it will provide single empty dictionary so that a new chain always has at least one mapping

**How to create a chainmap:**

To create a chainmap we are going to use the chainmap constructor,we are send the dictionary as an argument

**Example:**

from collections import ChainMap

dict1 = { 'a' : 1, 'b' : 2 }

dict2 = { 'c' : 3, 'b' : 4 }

chain\_map = ChainMap(dict1, dict2)

print(chain\_map.maps)

print(chain\_map['a'])

dict2['c'] = 5

print(chain\_map.maps)

print (list(chain\_map.keys()))

print (list(chain\_map.values()))

dict3 = {'e' : 5, 'f' : 6}

chain\_map=chain\_map.new\_child(dict3) #Adding a New Dictionary to ChainMap

print(chain\_map)

**Output:**

[{'b': 2, 'a': 1}, {'b': 4, 'c': 3}]

1

[{'b': 2, 'a': 1}, {'b': 4, 'c': 5}]

['b', 'c', 'a']

[2, 5, 1]

ChainMap({'f': 6, 'e': 5}, {'b': 2, 'a': 1}, {'b': 4, 'c': 5})

**Counter**

It is a subclass of dictionary .It is colleting the module takes ampping or iterable as the argumnet and it will returns the dictionary.In dictionary element or data is a key ,value is the no.of time data has been occured

**import:**

from collections import Counter

**Syntax:**

class collections.Counter([iterable-or-mapping])

We can create a counter object by using counter() function

**Ex:**

from collections import Counter

usr\_list = [1,2,3,4,1,2,6,7,3,8,1]

count\_obj = Counter(usr\_list)

print("all element count is:",count\_obj) #all elements count all element count is: Counter({1: 3, 2: 2, 3: 2, 4: 1, 6: 1, 7: 1, 8: 1})

count 1 is: 3

Counter({('India', 89): 2, ('England', 99): 1, ('Australia', 92): 1})

print("count 1 is:",count\_obj[1]) #individual count

team\_scores= [

('India', 89),

('Australia', 92),

('England', 99),

('India', 89)

]

count = Counter(team\_scores)

print(count)

**Output:**

all element count is: Counter({1: 3, 2: 2, 3: 2, 4: 1, 6: 1, 7: 1, 8: 1})

count 1 is: 3

Counter({('India', 89): 2, ('England', 99): 1, ('Australia', 92): 1})

Meanwhile counter is having three extra additional functions they are.

1.Elements

2.Most\_common([n])

3.Subtract([interable-or-mapping])

**Element Fucntion:**

An element function is used get the elements of counter object ,it will return a lsit contains the all elements in that counter object

**Syntax:**

counter\_object.elements()

**Ex:**

from collections import Counter

user\_count\_obj = Counter({10:2,20:4})

print(list(user\_count\_obj.elements()))

**output:**

[10, 10, 20, 20, 20, 20]

In this example we have created the user defined object with the argument as dictionary

Here count of 10 is 2 and count of 20 is 3

**most\_common() Function:**

Basically counter returns the elements in unsort order,but by using most common function you can sort it according to the numbers.

**Syntax:**

**Ex:**

from collections import Counter

usr\_list = [1,2,3,4,5,6,7,8,1,2,3,4,6,7,3,8,9,1]

usr\_count\_obj = Counter(usr\_list)

print(usr\_count\_obj.most\_common())

**Output:**

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

If we observe the output,it is returned the sorted list of data no.of time it has occured

**substract function:**

This function is take mapping(dictionary) or iterable(list) as an argument and then it will substract the element countusing this argument.

**Syntax:**

counter.substract(iterable)

**Ex:**

from collections import Counter

usr\_count\_obj = Counter({10:2,20:4})

user\_dict = {1:1, 2:2}

print(usr\_count\_obj)

usr\_count\_obj.subtract(user\_dict)

print(usr\_count\_obj)

**Output:**

Counter({20: 4, 10: 2})

Counter({10: 1, 20: 2})

If we observe above output the user\_count\_object will having the 10 of 2 counts,20 of 4 counts

after substract it becoms user\_count\_object having the 10 of 1 counts,20 of 2 counts

**defaultdict**

It works similar like regular dictionaries the difference between both is in dictionary if key not found it will returns the keyerror ,in case of defaultdict it will returns the value depends on the which data type it has initialized

**import :**

from collections import defaultdict

**Syntax:**

class collections.defaultdict([default\_factory[, ...]])

Here default\_factory is the type of the data type which is telling for default value if key is not found

**Ex:**

from collections import defaultdict

details = defaultdict(float) #if i change as int it will give the default value as just 0

details['name'] = "Ritika"

details['marks'] = 565

details['fee'] = 50000

print(details)

print(details['id']) #Here we are trying to access the data which is not existing

**Output:**

defaultdict(<class 'float'>, {'name': 'Ritika', 'marks': 565, 'fee': 50000})

0.0

if we see above example float passed as the default\_factory ,as well as we are trying to access the data by using key which is not existing,in this case regualar dictinaries will through the errors,but in default dict it will give the default value.

**Ordereddict**

It is a dictionary where we maintain the key in sorted order means it will not chage in which order inserted,it means in case if anybody change the value also key will not change the value

**import:**

from collections import OrderedDict

**Syntax:**

class collections.OrderedDict([items])

**Ex:**

from collections import OrderedDict

fruits\_type = OrderedDict([

("Apple",200),

("Watermelon",150),

("Grapes",100),

("Mangoes",100)

])

for key, value in fruits\_type.items():

print(key, value)

**Output:**

Apple 200

Watermelon 150

Grapes 100

Mangoes 100