Collections Module

Collections module

1. deque
2. Counter
3. OrderedDict
4. ChainMap

Collections —> deque

The Deque module is a part of collections library. It has the methods for adding and removing elements which can be invoked directly with arguments.

```
1. from collections import deque
2. # declaring the deque
3. list_name = deque()
```

Example

```
1.  from collections import deque
2.
3.  # declaring the deque
4.  fruit_list = deque(['Apple', 'Mango', 'Peaches', 'Banana', 'Papaya'])
5.
6.  # printing the deque
7.  print(fruit_list)
```

Output:

```
deque(['Apple', 'Mango', 'Peaches', 'Banana', 'Papaya'])
```

S. No.	Operation	Description
1	append()	The append() function is utilized for the addition of the data element in its parameter to the right end of the deque.
2	appendleft()	The appendleft() function is utilized for the addition of the data element in its parameter to the left end of the deque.
3	pop()	The pop() function is utilized for the deletion of the data element from the right end of the deque.
4	popleft()	The popleft() function is utilized for the deletion of the data element from the right end of the deque.
5	index(element, begin, end)	The index() function is utilized to return the first index value specified in the parameters, start the search from begin till end index.
6	insert(i, x)	The insert() function is utilized to insert the value described in the parameter 'x' at index number 'i' mentioned in parameters.

7	remove()	The remove() function is utilized to remove the first occurrence of the value specified in the parameters.
8	count()	The count() function is utilized to count the total number of occurrences of the value specified in the parameters.
9	extend(iterable)	The extend() function is utilized to insert multiple data elements at the right end of the deque. The parameter passed is iterable.
10	extendleft(iterable)	The extendleft() function is utilized to insert multiple data elements at the left end of the deque. The parameter passed is iterable. The Order is also reversed as an output of left appends.
11	reverse()	The reverse() function is utilized to reverse the Order of deque data elements.
12	rotate()	The rotate() function is utilized to rotate the deque by the number mentioned in parameters. If the mentioned number is a negative value, then the rotation occurs to the left. Else rotation is to the right.

Example

```
import collections

# declaring the deque
my_deque = collections.deque([10, 20, 30, 40, 50])

# using the append() function to add
# data element at right end
# inserting 60 at the end of the deque
my_deque.append(60)

# printing the resultant deque
print( "The deque after appending at right: " )
print( my_deque )
```

```
# using the appendleft() function to add
# data element at left end
# inserting 70 at the starting of the deque
my_deque.appendleft(70)

# printing the resultant deque
print( "The deque after appending at left: " )
print( my_deque )

# using the pop() function to remove
# data element from the right end
# removing 60 from the right end of deque
my_deque.pop()

# printing the resultant deque
print( "The deque after removing from right: " )
print( my_deque )

# using the popleft() function to remove
# data element from the left end
# removing 70 from the left end of deque
my_deque.popleft()
```

Output

```
The deque after appending at right:
deque([10, 20, 30, 40, 50, 60])
The deque after appending at left:
deque([70, 10, 20, 30, 40, 50, 60])
The deque after removing from right:
deque([70, 10, 20, 30, 40, 50])
The deque after removing from left:
deque([10, 20, 30, 40, 50])
```

Counter

Initializing

Counter supports three forms of initialization.

Its constructor can be called with a sequence of items (iterable), a dictionary

containing keys and counts (mapping, or using keyword arguments mapping string

names to counts (keyword args).

```
import collections
print collections.Counter(['a', 'b', 'c', 'a', 'b', 'b'])
print collections.Counter({'a':2, 'b':3, 'c':1})
print collections.Counter(a=2, b=3, c=1)
```

The results of all three forms of initialization are the same.

It is a collection where elements are stored as dictionary keys and their counts are stored as dictionary values.

Counter takes an iterable (such as a string, a list, or a dict object) and gives the object counts

list, string, and dictionary object, When they are passed into the **Counter class**, it returns a dictionary where the keys are the elements and their counts are the values of the returned dictionary.

```
from collections import Counter


a_list = [1,2,2,3,4,4,4]
a_string = 'data'
a_dict = {'a': 5, 'b':3, 'c':5, 'd':5, 'e':1 }


#counting objects in a list
c_list = Counter(a_list)


#counting characters in a string
c_string = Counter(a_string)


#counting values in a dictionary
c_dict = Counter(a_dict.values())


print (c_list)
print (c_string)
print (c_dict)


Output:
Counter({4: 3, 2: 2, 1: 1, 3: 1})
Counter({'a': 2, 'd': 1, 't': 1})
Counter({5: 3, 3: 1, 1: 1})
```

```
from collections import Counter


a_text = 'When developing your application, you may need to overcome
some tasks involving counting the elements in a container object. You
may need to count word frequencies in a text or you may want to know
the most frequently used number in a list object.'


a_text_splited = a_text.split()
a_counter = Counter(a_text_splited)
most_occur = a_counter.most_common(5)


print(most_occur)
print(sorted(a_counter.elements()))
```

Output:

```
[('may', 3), ('to', 3), ('in', 3), ('a', 3), ('you', 2)]


['When', 'You', 'a', 'a', 'a', 'application,', 'container', 'count',
'counting', 'developing', 'elements', 'frequencies', 'frequently',
'in', 'in', 'in', 'involving', 'know', 'list', 'may', 'may', 'may',
'most', 'need', 'need', 'number', 'object.', 'object.', 'or',
'overcome', 'some', 'tasks', 'text', 'the', 'the', 'to', 'to', 'to',
'used', 'want', 'word', 'you', 'you', 'your']
```

ChainMap

The ChainMap is used to encapsulates the dictionaries into single unit.

The ChainMap is a standard library class, which is located in the collections module.

To use it at first we need to import it the collections standard library module.

The maps and keys() values() functions

The maps is used to display all key value pairs of all the dictionaries from the ChainMap. The keys() method will return the keys from the ChainMap, and values() method returns all the values() of different keys from the ChainMap.

Example Code

```
import collections as col
con_code1 = {'India' : 'IN', 'China' : 'CN'}
con_code2 = {'France' : 'FR', 'United Kingdom' : 'GB'}
chain = col.ChainMap(con_code1, con_code2)
print("Initial Chain: " + str(chain.maps))
print('The keys in the ChainMap: ' + str(list(chain.keys())))
print('The values in the ChainMap: ' + str(list(chain.values())))
```

Output

```
Initial Chain: [{'India': 'IN', 'China': 'CN'}, {'France': 'FR', 'United Kingdom': 'G
The keys in the ChainMap: ['China', 'United Kingdom', 'India', 'France']
The values in the ChainMap: ['CN', 'GB', 'IN', 'FR']
```

The new_child() and reversed method

The new_child() method is used to add another dictionary object to the ChainMap at the beginning. And the reversed method is also can be used to ChainMap to reverse the order of the key-value pairs.

Example Code

```
import collections as col
con_code1 = {'India' : 'IN', 'China' : 'CN'}
con_code2 = {'France' : 'FR', 'United Kingdom' : 'GB'}
code = {'Japan' : 'JP'}
chain = col.ChainMap(con_code1, con_code2)
print("Initial Chain: " + str(chain.maps))
chain = chain.new_child(code)    #Insert New Child
print("Chain after Inserting new Child: " + str(chain.maps))
chain.maps = reversed(chain.maps)
print("Reversed Chain: " + str(chain))
```

collections — Container datatypes — Python 3.10.4 documentation for more information.

The **OrderedDict** is a standard library class, which is located in the collections module.

To use it at first we need to import it the collections standard library module.

```
import collections


#Create ordered dict
my_ord_dict = collections.OrderedDict()
my_ord_dict['AA'] = 11
my_ord_dict['BB'] = 22
my_ord_dict['CC'] = 33
my_ord_dict['DD'] = 44
for item in my_ord_dict.items():
   print(item)
```