**Python Collections Module**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 27 Apr, 2022

The collection Module in Python provides different types of containers. A Container is an object that is used to store different objects and provide a way to access the contained objects and iterate over them. Some of the built-in containers are [Tuple](https://www.geeksforgeeks.org/python-tuples/), [List](https://www.geeksforgeeks.org/python-list/), [Dictionary](https://www.geeksforgeeks.org/python-dictionary/), etc. In this article, we will discuss the different containers provided by the collections module.

***Table of Content:***

* [*Counters*](https://www.geeksforgeeks.org/python-collections-module/#counters)
* [*OrderedDict*](https://www.geeksforgeeks.org/python-collections-module/#ordereddict)
* [*DefaultDict*](https://www.geeksforgeeks.org/python-collections-module/#defaultdict)
* [*ChainMap*](https://www.geeksforgeeks.org/python-collections-module/#chainmap)
* [*NamedTuple*](https://www.geeksforgeeks.org/python-collections-module/#namedtuple)
* [*DeQue*](https://www.geeksforgeeks.org/python-collections-module/#deque)
* [*UserDict*](https://www.geeksforgeeks.org/python-collections-module/#userdict)
* [*UserList*](https://www.geeksforgeeks.org/python-collections-module/#userlist)
* [*UserString*](https://www.geeksforgeeks.org/python-collections-module/#userstring)

**Counters**

A [counter](https://www.geeksforgeeks.org/counters-in-python-set-1/) is a sub-class of the dictionary. It is used to keep the count of the elements in an iterable in the form of an unordered dictionary where the key represents the element in the iterable and value represents the count of that element in the iterable.

**Note:**It is equivalent to bag or multiset of other languages.

**Syntax:**

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

**Initializing Counter Objects**

The counter object can be initialized using the counter() function and this function can be called in one of the following ways:

* With a sequence of items
* With a dictionary containing keys and counts
* With keyword arguments mapping string names to counts

**Example:**

* Python3

|  |
| --- |
| # A Python program to show different  # ways to create Counter  from collections import Counter    # With sequence of items  print(Counter(['B','B','A','B','C','A','B',                 'B','A','C']))    # with dictionary  print(Counter({'A':3, 'B':5, 'C':2}))    # with keyword arguments  print(Counter(A=3, B=5, C=2)) |

**Output:**

Counter({'B': 5, 'A': 3, 'C': 2})

Counter({'B': 5, 'A': 3, 'C': 2})

Counter({'B': 5, 'A': 3, 'C': 2})

**Note:**For more information, refer  [Counters in Python.](https://www.geeksforgeeks.org/counters-in-python-set-1/)

**OrderedDict**

An [OrderedDict](https://www.geeksforgeeks.org/ordereddict-in-python/)is also a sub-class of dictionary but unlike dictionary, it remembers the order in which the keys were inserted.

**Syntax:**

class collections.OrderDict()

**Example:**

* Python3

|  |
| --- |
| # A Python program to demonstrate working  # of OrderedDict    from collections import OrderedDict    print("This is a Dict:\n")  d = {}  d['a'] = 1  d['b'] = 2  d['c'] = 3  d['d'] = 4    for key, value in d.items():      print(key, value)    print("\nThis is an Ordered Dict:\n")  od = OrderedDict()  od['a'] = 1  od['b'] = 2  od['c'] = 3  od['d'] = 4    for key, value in od.items():      print(key, value) |

**Output:**

This is a Dict:

a 1

b 2

c 3

d 4

This is an Ordered Dict:

a 1

b 2

c 3

d 4

While deleting and re-inserting the same key will push the key to the last to maintain the order of insertion of the key.

**Example:**

* Python3

|  |
| --- |
| # A Python program to demonstrate working  # of OrderedDict    from collections import OrderedDict      od = OrderedDict()  od['a'] = 1  od['b'] = 2  od['c'] = 3  od['d'] = 4    print('Before Deleting')  for key, value in od.items():      print(key, value)    # deleting element  od.pop('a')    # Re-inserting the same  od['a'] = 1    print('\nAfter re-inserting')  for key, value in od.items():      print(key, value) |

**Output:**

Before Deleting

a 1

b 2

c 3

d 4

After re-inserting

b 2

c 3

d 4

a 1

**Note:**for more information, refer [OrderedDict in Python](https://www.geeksforgeeks.org/ordereddict-in-python/)

**DefaultDict**

A [DefaultDict](https://www.geeksforgeeks.org/defaultdict-in-python/) is also a sub-class to dictionary. It is used to provide some default values for the key that does not exist and never raises a KeyError.

**Syntax:**

class collections.defaultdict(default\_factory)

default\_factory is a function that provides the default value for the dictionary created. If this parameter is absent then the KeyError is raised.

**Initializing DefaultDict Objects**

DefaultDict objects can be initialized using DefaultDict() method by passing the data type as an argument.

**Example:**

* Python3

|  |
| --- |
| # Python program to demonstrate  # defaultdict      from collections import defaultdict      # Defining the dict  d = defaultdict(int)    L = [1, 2, 3, 4, 2, 4, 1, 2]    # Iterate through the list  # for keeping the count  for i in L:        # The default value is 0      # so there is no need to      # enter the key first      d[i] += 1    print(d) |

**Output:**

defaultdict(<class 'int'>, {1: 2, 2: 3, 3: 1, 4: 2})

**Example 2:**

* Python3

|  |
| --- |
| # Python program to demonstrate  # defaultdict      from collections import defaultdict      # Defining a dict  d = defaultdict(list)    for i in range(5):      d[i].append(i)    print("Dictionary with values as list:")  print(d) |

**Output:**

*Dictionary with values as list:   
defaultdict(<class ‘list’>, {0: [0], 1: [1], 2: [2], 3: [3], 4: [4]})*

**Note:** For more information, refer [Defaultdict in Python](https://www.geeksforgeeks.org/defaultdict-in-python/)

**ChainMap**

A [ChainMap](https://www.geeksforgeeks.org/chainmap-in-python/) encapsulates many dictionaries into a single unit and returns a list of dictionaries.

**Syntax:**

class collections.ChainMap(dict1, dict2)

**Example:**

* Python3

|  |
| --- |
| # Python program to demonstrate  # ChainMap      from collections import ChainMap      d1 = {'a': 1, 'b': 2}  d2 = {'c': 3, 'd': 4}  d3 = {'e': 5, 'f': 6}    # Defining the chainmap  c = ChainMap(d1, d2, d3)    print(c) |

**Output:**

ChainMap({'a': 1, 'b': 2}, {'c': 3, 'd': 4}, {'e': 5, 'f': 6})

**Accessing Keys and Values from ChainMap**

Values from ChainMap can be accessed using the key name. They can also be accessed by using the keys() and values() method.

**Example:**

* Python3

|  |
| --- |
| # Python program to demonstrate  # ChainMap      from collections import ChainMap      d1 = {'a': 1, 'b': 2}  d2 = {'c': 3, 'd': 4}  d3 = {'e': 5, 'f': 6}    # Defining the chainmap  c = ChainMap(d1, d2, d3)    # Accessing Values using key name  print(c['a'])    # Accessing values using values()  # method  print(c.values())    # Accessing keys using keys()  # method  print(c.keys()) |

**Output:**

*1   
ValuesView(ChainMap({‘a’: 1, ‘b’: 2}, {‘c’: 3, ‘d’: 4}, {‘e’: 5, ‘f’: 6}))   
KeysView(ChainMap({‘a’: 1, ‘b’: 2}, {‘c’: 3, ‘d’: 4}, {‘e’: 5, ‘f’: 6}))*

**Adding new dictionary**

A new dictionary can be added by using the **new\_child()** method. The newly added dictionary is added at the beginning of the ChainMap.

**Example:**

* Python3

|  |
| --- |
| # Python code to demonstrate ChainMap and  # new\_child()    import collections    # initializing dictionaries  dic1 = { 'a' : 1, 'b' : 2 }  dic2 = { 'b' : 3, 'c' : 4 }  dic3 = { 'f' : 5 }    # initializing ChainMap  chain = collections.ChainMap(dic1, dic2)    # printing chainMap  print ("All the ChainMap contents are : ")  print (chain)    # using new\_child() to add new dictionary  chain1 = chain.new\_child(dic3)    # printing chainMap  print ("Displaying new ChainMap : ")  print (chain1) |

**Output:**

All the ChainMap contents are :

ChainMap({'a': 1, 'b': 2}, {'b': 3, 'c': 4})

Displaying new ChainMap :

ChainMap({'f': 5}, {'a': 1, 'b': 2}, {'b': 3, 'c': 4})

**Note:**For more information, refer [ChainMap in Python](https://www.geeksforgeeks.org/chainmap-in-python/)

**NamedTuple**

A [NamedTuple](https://www.geeksforgeeks.org/namedtuple-in-python/) returns a tuple object with names for each position which the ordinary tuples lack. For example, consider a tuple names student where the first element represents fname, second represents lname and the third element represents the DOB. Suppose for calling fname instead of remembering the index position you can actually call the element by using the fname argument, then it will be really easy for accessing tuples element. This functionality is provided by the NamedTuple.

**Syntax:**

class collections.namedtuple(typename, field\_names)

**Example:**

* Python3

|  |
| --- |
| # Python code to demonstrate namedtuple()    from collections import namedtuple    # Declaring namedtuple()  Student = namedtuple('Student',['name','age','DOB'])    # Adding values  S = Student('Nandini','19','2541997')    # Access using index  print ("The Student age using index is : ",end ="")  print (S[1])    # Access using name  print ("The Student name using keyname is : ",end ="")  print (S.name) |

**Output:**

The Student age using index is : 19

The Student name using keyname is : Nandini

**Conversion Operations**

**1. \_make():** This function is used to return a namedtuple() from the iterable passed as argument.

**2. \_asdict():** This function returnsthe [OrdereDict()](https://www.geeksforgeeks.org/ordereddict-in-python/) as constructed from the mapped values of namedtuple().

**Example:**

* Python3

|  |
| --- |
| # Python code to demonstrate namedtuple() and  # \_make(), \_asdict()      from collections import namedtuple    # Declaring namedtuple()  Student = namedtuple('Student',['name','age','DOB'])    # Adding values  S = Student('Nandini','19','2541997')    # initializing iterable  li = ['Manjeet', '19', '411997' ]    # initializing dict  di = { 'name' : "Nikhil", 'age' : 19 , 'DOB' : '1391997' }    # using \_make() to return namedtuple()  print ("The namedtuple instance using iterable is  : ")  print (Student.\_make(li))    # using \_asdict() to return an OrderedDict()  print ("The OrderedDict instance using namedtuple is  : ")  print (S.\_asdict()) |

**Output:**

The namedtuple instance using iterable is :

Student(name='Manjeet', age='19', DOB='411997')

The OrderedDict instance using namedtuple is :

OrderedDict([('name', 'Nandini'), ('age', '19'), ('DOB', '2541997')])

**Note:**For more  information, refer [NamedTuple in Python](https://www.geeksforgeeks.org/namedtuple-in-python/)

**Deque**

[Deque (Doubly Ended Queue)](https://www.geeksforgeeks.org/deque-in-python/) is the optimized list for quicker append and pop operations from both sides of the container. It provides O(1) time complexity for append and pop operations as compared to list with O(n) time complexity.

**Syntax:**

class collections.deque(list)

This function takes the list as an argument.

**Example:**

* Python3

|  |
| --- |
| # Python code to demonstrate deque      from collections import deque    # Declaring deque  queue = deque(['name','age','DOB'])    print(queue) |

**Output:**

deque(['name', 'age', 'DOB'])

**Inserting Elements**

Elements in deque can be inserted from both ends. To insert the elements from right append() method is used and to insert the elements from the left appendleft() method is used.

**Example:**

* Python3

|  |
| --- |
| # Python code to demonstrate working of  # append(), appendleft()      from collections import deque    # initializing deque  de = deque([1,2,3])    # using append() to insert element at right end  # inserts 4 at the end of deque  de.append(4)    # printing modified deque  print ("The deque after appending at right is : ")  print (de)    # using appendleft() to insert element at right end  # inserts 6 at the beginning of deque  de.appendleft(6)    # printing modified deque  print ("The deque after appending at left is : ")  print (de) |

**Output:**

The deque after appending at right is :

deque([1, 2, 3, 4])

The deque after appending at left is :

deque([6, 1, 2, 3, 4])

**Removing Elements**

Elements can also be removed from the deque from both the ends. To remove elements from right use pop() method and to remove elements from the left use popleft() method.

**Example:**

* Python3

|  |
| --- |
| # Python code to demonstrate working of  # pop(), and popleft()    from collections import deque    # initializing deque  de = deque([6, 1, 2, 3, 4])    # using pop() to delete element from right end  # deletes 4 from the right end of deque  de.pop()    # printing modified deque  print ("The deque after deleting from right is : ")  print (de)    # using popleft() to delete element from left end  # deletes 6 from the left end of deque  de.popleft()    # printing modified deque  print ("The deque after deleting from left is : ")  print (de) |

**Output:**

The deque after deleting from right is :

deque([6, 1, 2, 3])

The deque after deleting from left is :

deque([1, 2, 3])

**Note:** For more information, refer [Deque in Python.](https://www.geeksforgeeks.org/deque-in-python/)

**UserDict**

[UserDict](https://www.geeksforgeeks.org/collections-userdict-in-python/) is a dictionary-like container that acts as a wrapper around the dictionary objects. This container is used when someone wants to create their own dictionary with some modified or new functionality.

**Syntax:**

class collections.UserDict([initialdata])

**Example:**

* Python3

|  |
| --- |
| # Python program to demonstrate  # userdict      from collections import UserDict      # Creating a Dictionary where  # deletion is not allowed  class MyDict(UserDict):        # Function to stop deletion      # from dictionary      def \_\_del\_\_(self):          raise RuntimeError("Deletion not allowed")        # Function to stop pop from      # dictionary      def pop(self, s = None):          raise RuntimeError("Deletion not allowed")        # Function to stop popitem      # from Dictionary      def popitem(self, s = None):          raise RuntimeError("Deletion not allowed")    # Driver's code  d = MyDict({'a':1,      'b': 2,      'c': 3})    d.pop(1) |

**Output:**

Traceback (most recent call last):

File "/home/f8db849e4cf1e58177983b2b6023c1a3.py", line 32, in <module>

d.pop(1)

File "/home/f8db849e4cf1e58177983b2b6023c1a3.py", line 20, in pop

raise RuntimeError("Deletion not allowed")

RuntimeError: Deletion not allowed

Exception ignored in: <bound method MyDict.\_\_del\_\_ of {'a': 1, 'b': 2, 'c': 3}>

Traceback (most recent call last):

File "/home/f8db849e4cf1e58177983b2b6023c1a3.py", line 15, in \_\_del\_\_

RuntimeError: Deletion not allowed

Note: For more information, refer [UserDict in Python](https://www.geeksforgeeks.org/collections-userdict-in-python/)

**UserList**

[UserList](https://www.geeksforgeeks.org/collections-userlist-in-python/)is a list like container that acts as a wrapper around the list objects. This is useful when someone wants to create their own list with some modified or additional functionality.

**Syntax:**

class collections.UserList([list])

**Example:**

* Python3

|  |
| --- |
| # Python program to demonstrate  # userlist      from collections import UserList      # Creating a List where  # deletion is not allowed  class MyList(UserList):        # Function to stop deletion      # from List      def remove(self, s = None):          raise RuntimeError("Deletion not allowed")        # Function to stop pop from      # List      def pop(self, s = None):          raise RuntimeError("Deletion not allowed")    # Driver's code  L = MyList([1, 2, 3, 4])    print("Original List")    # Inserting to List"  L.append(5)  print("After Insertion")  print(L)    # Deleting From List  L.remove() |

**Output:**

Original List

After Insertion

[1, 2, 3, 4, 5]

Traceback (most recent call last):

File "/home/c90487eefa7474c0566435269f50a52a.py", line 33, in <module>

L.remove()

File "/home/c90487eefa7474c0566435269f50a52a.py", line 15, in remove

raise RuntimeError("Deletion not allowed")

RuntimeError: Deletion not allowed

**Note:** For more information, refer [UserList in Python](https://www.geeksforgeeks.org/collections-userlist-in-python/)

**UserString**

[UserString](https://www.geeksforgeeks.org/collections-userstring-in-python/) is a string like container and just like UserDict and UserList it acts as a wrapper around string objects. It is used when someone wants to create their own strings with some modified or additional functionality.

**Syntax:**

class collections.UserString(seq)

**Example:**

* Python3

|  |
| --- |
| # Python program to demonstrate  # userstring      from collections import UserString      # Creating a Mutable String  class Mystring(UserString):        # Function to append to      # string      def append(self, s):          self.data += s        # Function to remove from      # string      def remove(self, s):          self.data = self.data.replace(s, "")    # Driver's code  s1 = Mystring("Geeks")  print("Original String:", s1.data)    # Appending to string  s1.append("s")  print("String After Appending:", s1.data)    # Removing from string  s1.remove("e")  print("String after Removing:", s1.data) |

**Output:**

Original String: Geeks

String After Appending: Geekss

String after Removing: Gkss