1. Deque :

A double-ended queue, or deque, has the feature of adding and removing elements from either end. 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. In the below program we import the collections module and declare a deque. Without need of any class we use the in-built implement these methods directly.

1. DefaultDict :

Defaultdict is a container like dictionaries present in the module collections. Defaultdict is a sub-class of the dictionary class that returns a dictionary-like object. The functionality of both dictionaries and defualtdict are almost same except for the fact that defualtdict never raises a KeyError. It provides a default value for the key that does not exists.

1. NamedTuple :

Python supports a type of container like dictionaries called “namedtuple()” present in module, “collections“. Like dictionaries they contain keys that are hashed to a particular value. But on contrary, it supports both access from key value and iteration, the functionality that dictionaries lack.

Namedtuples are immutable

1. OrderedDict :

An OrderedDict is a dictionary subclass that remembers the order that keys were first inserted. The only difference between dict() and OrderedDict() is that:

OrderedDict preserves the order in which the keys are inserted. A regular dict doesn’t track the insertion order, and iterating it gives the values in an arbitrary order. By contrast, the order the items are inserted is remembered by OrderedDict.

1. Key value Change: If the value of a certain key is changed, the position of the key remains unchanged in OrderedDict.
2. Deletion and Re-Inserting: Deleting and re-inserting the same key will push it to the back as OrderedDict, however, maintains the order of insertion.
3. Counter :

Counter is a container included in the collections module. Now you all must be wondering what is a container. Don’t worry first let’s discuss about the container.

What is Container?

Containers are objects that hold objects. They provide a way to access the contained objects and iterate over them. Examples of built in containers are Tuple, list, and dictionary. Others are included in Collections module.

A Counter is a subclass of dict. Therefore it is an unordered collection where elements and their respective count are stored as a dictionary. This is equivalent to a bag or multiset of other languages.

1. ChainMap

Python contains a container called “ChainMap” which encapsulates many dictionaries into one unit. ChainMap is member of module “collections“.

keys() :- This function is used to display all the keys of all the dictionaries in ChainMap.

values() :- This function is used to display values of all the dictionaries in ChainMap.

maps() :- This function is used to display keys with corresponding values of all the dictionaries in ChainMap.

1. Iterators :

An iterator is an object which contains a countable number of values and it is used to iterate over iterable objects like list, tuples, sets, etc. Iterators are implemented using a class and a local variable for iterating is not required here, It follows lazy evaluation where the evaluation of the expression will be on hold and stored in the memory until the item is called specifically which helps us to avoid repeated evaluation. As lazy evaluation is implemented, it requires only 1 memory location to process the value and when we are using a large dataset then, wastage of RAM space will be reduced the need to load the entire dataset at the same time will not be there.

Using an iterator-

1. iter() keyword is used to create an iterator containing an iterable object.
2. next() keyword is used to call the next element in the iterable object.
3. After the iterable object is completed, to use them again reassign them to the same object.
4. Generators:

It is another way of creating iterators in a simple way where it uses the keyword “yield” instead of returning it in a defined function. Generators are implemented using a function. Just as iterators, generators also follow lazy evaluation. Here, the yield function returns the data without affecting or exiting the function. It will return a sequence of data in an iterable format where we need to iterate over the sequence to use the data as they won’t store the entire sequence in the memory.

**Table of difference between Iterator vs Generators**

| **Iterator** | **Generator** |
| --- | --- |
| Class is used to implement an iterator | Function is used to implement a generator. |
| Local Variables aren’t used here. | All the local variables before the yield function are stored. |
| Iterators are used mostly to iterate or convert other objects to an iterator using iter() function. | Generators are mostly used in loops to generate an iterator by returning all the values in the loop without affecting the iteration of the loop |
| Iterator uses iter() and next() functions | Generator uses yield keyword |
| Every iterator is not a generator | Every generator is an iterator |