About Collections Module

In [ ]:
```
Inbuilt Containers(Collections of elements)--> list,tuple,set,dictionary
Collections --> it is also a module. in which we are having different type of containers.Few of the containers are mentioned
                below:
                    --> OrderedDict
                    --> Counter
                    --> Chainmap
                    --> Dequeue
                    --> DefaultDict
```

About Counter() Function

In [ ]:
```
Counters --> will return a dictionary.
         --> The behaviour of counter function is varied.
                1.You can passing any sequence like list --> Frequence of each character in form of dictionary
                2.You are pasing dictionary  --> return dictionary
                3.You are passing keyword argument --> return dictionary
```

Example(By passing sequence of items)

In [6]:
```
#Example: By passing sequence of items
from collections import Counter
print(Counter(["B","C","M","A","M","B","C"]))
print(Counter({"B","C","M","A","M","B","C"}))
#If you are passing any sequence then it will give you the frequency of each chatavyer
```

```
Counter({'B': 2, 'C': 2, 'M': 2, 'A': 1})
Counter({'B': 2, 'C': 2, 'M': 2, 'A': 1})
```

Example(By passing sequence of dictionary)

In [4]:
```
#Example: By passing sequence of dictionary
from collections import Counter
print(Counter({1:2,"3":4,5:6,1:4}))
```

```
Counter({5: 6, 1: 4, '3': 4})
```

Example(By passing keyword argument)

In [5]:
```
#By passing keyword argument
from collections import Counter
print(Counter(A=3,B=4,C=5))
```

```
Counter({'C': 5, 'B': 4, 'A': 3})
```

Ordereddict

In [ ]:
```
Ordereddict --> OrderedDict is a dict subclass that preserves the order in which key-value pairs,
                commonly known as items, are inserted into the dictionary.
```

Example

In [1]:
```
#Ordered Dict
from collections import OrderedDict
print("Normal Dictionary")
d={}
d["a"]=1
d["b"]=2
d["c"]=98
d["d"]=4
d["e"]=5
print(d)
for k,v in d.items():
    print(k,v)
print("Ordered Dictionary")
od=OrderedDict()
od["a"]=1
od["b"]=2
od["d"]=4
od["c"]=3
print(od)
for k,v in od.items():
    print(k,v)
```

```
Normal Dictionary
{'a': 1, 'b': 2, 'c': 98, 'd': 4, 'e': 5}
a 1
b 2
c 98
d 4
e 5
Ordered Dictionary
OrderedDict([('a', 1), ('b', 2), ('d', 4), ('c', 3)])
a 1
b 2
d 4
c 3
```

deletion and reinsertion In Ordereddict

In [2]:
```
#deletion and reinsertion
od=OrderedDict()
od["a"]=1
od["b"]=2
od["d"]=4
od["c"]=3
print(od)
od.pop("a")
od["a"]=1
print(od)
```

```
OrderedDict([('a', 1), ('b', 2), ('d', 4), ('c', 3)])
OrderedDict([('b', 2), ('d', 4), ('c', 3), ('a', 1)])
```

ChainMap()

In [ ]:
```
Chain map --> Combine two or more dictionary into single one (Encapsulate two or more dictionary into single one)
```

Example

In [16]:
```
#ChainMap --> combining two or more dictionary
from collections import ChainMap
d1={1:2,3:4}
d2={4:5,5:6,7:8}
d3={70:56,56:78,78:89}
c=ChainMap(d1,d2,d3)
print(c)
```

```
ChainMap({1: 2, 3: 4}, {4: 5, 5: 6, 7: 8}, {70: 56, 56: 78, 78: 89})
```

deque()

In [ ]:
```
Dequeue -->(Double Ended Queue) --> insertion and deletion can be done from both end(front and rear)
```

Example

In [18]:
```
from collections import deque
queue = deque(["name","class","rollno"])
print(queue)
```

```
deque(['name', 'class', 'rollno'])
```

Insertion in Deque

In [ ]:
```
for Insertion  Two Functions are There:
    append  --> add at the last of the dequeue
    appendleft --> add at the first position of the dequeue
```

Example

In [3]:
```
from collections import deque
q = deque(["name","class","rollno"])
q.append("hello")
print(q)
```

```
deque(['name', 'class', 'rollno', 'hello'])
```

In [4]:
```
from collections import deque
q = deque(["name","class","rollno"])
q.appendleft("hello")
print(q)
```

```
deque(['hello', 'name', 'class', 'rollno'])
```

Deletion in Deque

In [ ]:
```
for Deletion Two Functions are There:
    pop --> delete from the last
    popleft --> delete from the front
```

Example

In [22]:
```
#deletion from last
from collections import deque
q = deque(["name","class","rollno"])
q.pop()
print(q)
```

```
deque(['name', 'class'])
```

In [23]:
```
#deletion from front
from collections import deque
q = deque(["name","class","rollno"])
q.popleft()
print(q)
```

```
deque(['class', 'rollno'])
```

defaultdict()

In [ ]:
```
default dict --> A defaultdict works exactly like a normal dict, but it is initialized with a function ("default factory")
                 that takes no arguments and provides the default value for a nonexistent key.
             --> A defaultdict will never raise a KeyError. Any key that does not exist gets the value returned by the default
                 factory
```

Example of Normal dict

In [5]:
```
d = {1:2,3:4,5:6,7:8}
d[9]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [5], in <cell line: 2>()
      1 d = {1:2,3:4,5:6,7:8}
----> 2 d[9]

KeyError: 9
```

Example of Defaultdict

In [7]:
```
from collections import defaultdict
d=defaultdict(lambda:"This key is not present please enter valid key")
d[1]=2
d[2]=4
d[99]
```

Out[7]:
```
'This key is not present please enter valid key'
```

Normaldict vs Default dict

In [ ]:
```
Normal dict --> in case of normal dict if the key is not present then you will get an error that is key error
Default dict --> if the key is not present then you will not get any error
```

About Datetime Module

In [ ]:
```
--> it is also a module in python which is having certain functions related to date time
--> datetime in Python is the combination between dates and times.
--> The attributes of this class are similar to both date and separate classes.
--> These attributes include day, month, year, minute, second, microsecond, hour, and tzinfo.
```

Examples

today() --> returns Today's Date

In [8]:
```
from datetime import date
today=date.today()
print(today)
```

```
2022-11-20
```

now() --> return current date and time

In [9]:
```
from datetime import datetime
today=datetime.now()
print(today)
```

```
2022-11-20 21:15:30.281864
```

pytz Module--> return specific time zone Time

In [10]:
```
#Specific time zone
import pytz
import datetime
current_time = datetime.datetime.now(pytz.timezone("Asia/Karachi"))
print(current_time)
```

```
2022-11-20 20:46:04.475012+05:00
```

How we can generate the Execution time of a code?

In [14]:
```
import time
start = time.time()
a = 0
for i in range(1000):
    a += (i**100)
end = time.time()

print("The time of execution of above program is :",(end-start) * 10**3, "ms")
```

```
The time of execution of above program is : 3.0024051666259766 ms
```