Unit 5 Mutable Data Structure

mutable objects are easy to change.

Use of mutable objects is recommended when there is a need to change the size or content of the object.

These are of type list, dict, set . Custom classes are generally mutable.

Lists

Python lists are one of the most versatile data types that allow us to work with multiple elements at once.

- As the part of programming requirement, we have to store our data permanently for future purpose. For this requirement we should go for files.
- If we want to represent a group of individual objects as a single entity where insertion order is preserved and duplicates and heterogeneous objects are allowed, then we should go for List.
- List is dynamic because based on our requirement we can increase the size and decrease the size.
- In List the elements will be placed within square brackets and with comma separator.
- We can differentiate duplicate elements by using index and we can preserve insertion order by using index. Hence index will play very important role.
- Python supports both positive and negative indexes. +ve index means from left to right where as negative index means right to left.
- List objects are mutable. i.e we can change the content.

In [1]:
```
#Example of Lists as mutability
# Python code to test that
# lists are mutable
color = ["red", "blue", "green"]
print(color)

color[0] = "pink"
color[-1] = "orange"
print(color)
```

```
['red', 'blue', 'green']
['pink', 'blue', 'orange']
```

Declaring (creating of lists) and acessing through for loop

In [2]:
```
#Creating a empty list
list=[]
print(list)
print(type(list))
```

```
[]
<class 'list'>
```

In [3]:
```
#Creating a list through already known elements.
list=[1,4,7,8]
print(list)
print(type(list))
```

```
[1, 4, 7, 8]
<class 'list'>
```

In [4]:
```
#Creating a list through already known elements.
list=["A","B","C","D"]
print(list)
print(type(list))
```

```
['A', 'B', 'C', 'D']
<class 'list'>
```

In [5]:
```
#Creating a list through user defined values:
list=eval(input("Enter the list elements:"))
print(list)
print(type(list))
```

```
Enter the list elements:[1,2,3,4]
[1, 2, 3, 4]
<class 'list'>
```

eval is a built in function which parses the expression argument and evaluates it as a python expression. It takes input as a string and returns as an integer.

In [1]:
```
#Creating a list with list() Function:
l=list(range(0,5,1))
print(l)
print(type(l))
```

```
[0, 1, 2, 3, 4]
<class 'list'>
```

In [2]:
```
#Creation of list with split() function:
s="Python is easier and user friendly software...."
l=s.split()
print(l)
print(type(l))
```

```
['Python', 'is', 'easier', 'and', 'user', 'friendly', 'software....']
<class 'list'>
```

Accessing lists

There are various methods through which the lists can be accessed.

1. By using index
2. By using Slice Operator
3. By using for loop
4. By using while loop

1. accessing list using index:

- List follows zero based index. ie index of first element is zero.

- List supports both +ve and -ve indexes.
- +ve index meant for Left to Right.
- -ve index meant for Right to Left.

In [3]:
```
#Example of accessing list using index:
l=[1,3,5,7]
print(l[2])
print(l[0])
print(l[-2])
print(l[6])
```

```
5
1
5
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [3], line 6
      4 print(l[0])
      5 print(l[-2])
----> 6 print(l[6])

IndexError: list index out of range
```

1. By using Slice Operator:

- Syntax: list2 = list1[start:stop:step]
- Start: It indicates the Index where slice has to Start and Default Value is 0.
- Stop: It indicates the Index where slice has to End and Default Value is max allowed Index of List ie Length of the List.
- Step: Increment value and Default Value is 1.

In [4]:
```
#Example for slicing operator
n=[1,2,3,5,7,8]
print(n[1:5:2])
print(n[3:5])
print(n[1::5])
```

```
[2, 5]
[5, 7]
[2]
```

In [5]:
```
#Loop through the list by using for loop:
n=[5,7,9,2,3,8]
for i in n:
    print(i)
```

```
5
7
9
2
3
8
```

In [10]:
```
#Loop through the list by using while loop:
n = [5,7,8,6,9]
i = 0
```

```
while i < len(n):
    print(n[i])
    i=i+1
```

```
5
7
8
6
9
```

In [11]:
```
#Loop accessing through the list items by index numbers:
l=["A","B","c"]
x=len(l)
for i in range(x):
    print(l[i])
```

```
A
B
c
```

Important functions in list:

1. Append Function: We can use append() function to add item at the end of the list.

Syntax: variablename.append()

In [12]:
```
#Example of list function using append() function.
list=["Mango","Banana","Apple"]
list.append(1)
list.append("Strawberry")
list.append("Cherry")
print(list)
```

```
['Mango', 'Banana', 'Apple', 1, 'Strawberry', 'Cherry']
```

1. Extend function: To add all items of one list to another list

Syntax: list1.extend(list2)

all items of list2 will be added to list1

In [13]:
```
#Example of extend() function:
l1=["a","b","d"]
l2=["x","y","z"]
l1.extend(l2)
print(l1)
```

```
['a', 'b', 'd', 'x', 'y', 'z']
```

In [14]:
```
#Example of extend() function with string as an input
l1=["Aman","Daman"]

l1.extend("Kumar")
print(l1)
```

```
['Aman', 'Daman', 'K', 'u', 'm', 'a', 'r']
```

1. Insert() function in list- To insert item at a specified position.

Syntax of inser() is:

list.insert(index,value)

In [20]:
```
#Example of insert() function
a=[1,4,5,9,10]
a.insert(2,7)
print(a)
#It shifts and it does not replace the value at the specified index.
```

```
[1, 4, 7, 5, 9, 10]
```

In [21]:
```
#Example of insert() function
a=[1,4,5,9,10]
a.insert(10,7)
print(a)
#It always adds at the last if the index size is smaller than specified one.
```

```
[1, 4, 5, 9, 10, 7]
```

In [22]:
```
#Example of insert() function
a=[1,4,5,9,10]
a.insert(-10,7)
print(a)
#It always adds at the first if the index size is larger than specified one.
```

```
[7, 1, 4, 5, 9, 10]
```

1. Count() function in list- It returns the number of occurrences of specified item in the list.

syntax of count() is:

list.count(variable)

In [15]:
```
#Example of count() function in list
a=[1,2,3,3,3,4,4,4,5,6]
print(a.count(1))
print(a.count(2))
print(a.count(3))
print(a.count(5))
```

```
1
1
3
1
```

1. Index() function in list- It returns the index of 1st occurrence of specified item in the list.

syntax of index() is:

index.count(variable)

In [19]:
```
#Example of count() function in list
a=[1,2,3,3,3,4,4,4,5,6]
print(a.index(1))
print(a.index(2))
print(a.index(3))
print(a.index(5))
print(a.index(7))
```

```
0
1
2
8
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [19], line 7
      5 print(a.index(3))
      6 print(a.index(5))
----> 7 print(a.index(7))

ValueError: 7 is not in list
```

6.remove() function in python -We can use this function to remove specified item from the list. If the item present multiple items then only 1st occurence will be removed.

Syntax for remove()

list.remove(element)

In [23]:
```
#Example of remove() function
n=["A","B","C","D"]
n.remove("C")
print(n)
```

```
['A', 'B', 'D']
```

In [24]:
```
#Example of remove() function
n=["A","B","C","D"]
n.remove("E")
print(n)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In [24], line 3
      1 #Example of remove() function
      2 n=["A","B","C","D"]
----> 3 n.remove("E")
      4 print(n)

ValueError: list.remove(x): x not in list
```

1. pop() function in Python: It removes and returns the last element of the list. This is only function which manipulates list and returns some element. It prints the popped element.

Syntax of pop() element:

a.pop()

or a.pop(index)

In [25]:
```
#Example of pop() function
a=[1,5,4,8]
print(a.pop())
print(a.pop())
print(a)
```

```
8
4
[1, 5]
```

In [26]:
```
#Example of pop() function
a=[]
print(a.pop())

print(a)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In [26], line 3
      1 #Example of pop() function
      2 a=[]
----> 3 print(a.pop())
      5 print(a)

IndexError: pop from empty list
```

In [27]:
```
#Example of pop() function
a=[5,10,15,20,25]
print(a.pop(1))
print(a.pop(3))
print(a)
```

```
10
25
[5, 15, 20]
```

reverse() function in Python- It is used to reverse the order of elements in list.

Syntax for reverse():

list.reverse()

In [28]:
```
#Example of reverse() function

a=[2,7,3,6,5]
a.reverse()
print(a)
```

```
[5, 6, 3, 7, 2]
```

Sort() function in Python- It is used to sort the order of elements in list.

Syntax for sort():

list.sort()

In list by default insertion order is preserved.

If want to sort the elements of list according to default natural sorting order then we should go for sort() method. For numbers: Default Natural sorting Order is Ascending Order.

For Strings : Default Natural sorting order is Alphabetical Order.

In [29]:
```
#Example of sorting()
n = [2,5,15,1,0]
n.sort()
print(n)
```

```
[0, 1, 2, 5, 15]
```

In [30]:
```
#Example of sorting() in reverse order:
n = [2,5,15,1,0]
n.sort(reverse=True)
print(n)
```

```
[15, 5, 2, 1, 0]
```

In [31]:
```
#Example of sorting() in reverse order:
n = ["A","B","C","D"]
n.sort(reverse=True)
print(n)
```

```
['D', 'C', 'B', 'A']
```

In [33]:
```
#Example of sorting() in reverse order:
n = ["B","A","D","C"]
n.sort(reverse=False)
print(n)
```

```
['A', 'B', 'C', 'D']
```

Dictionaries

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

Example: rollno-name

phone no. -address

Duplicate keys are not allowed but values can be duplicated.

Heterogeneous objects are allowed for both key and values.

Insertion order is not preserved.

Dictionaries are mutable.

Indexing and slicing concepts are not applicable.

Dictionaries are written with curly brackets, and have keys and values:

Create a dictionary

Syntax for creating a dictionary is d={key1:value1, key2:value2,key3=value3,...}

Or d=dict()

dict keyword is used for dictionary.

To create an empty dictionary

d={} or d=dict()

In [2]:
```
#Example for create empty dictionary
d={}
print(type(d))
```

```
<class 'dict'>
```

In [3]:
```
#Example for creating another dictionaries.
d[1]="Atharva"
d[2]="Python"
d[3]="Anaconda"
print(d)
```

```
{1: 'Atharva', 2: 'Python', 3: 'Anaconda'}
```

In [4]:
```
#Example for creating dictionary using key:value syntax is:
d={1:"Atharva",2:"Python",3:"Anaconda"}
print(d)
```

```
{1: 'Atharva', 2: 'Python', 3: 'Anaconda'}
```

Accessing a dictionary:

1. Access elements by using keys.
2. Using for loop Accessing using for loop will be taught afterwards.

In [5]:
```
#Accessing a dictionary using keys.
d={1:"Atharva",2:"Python",3:"Anaconda"}
print(d[1])
print(d[2])
print(d[3])
print(d[4])
#Note here in d[1],d[2],d[3] 1,2,3 represents the keys and not index
```

```
Atharva
Python
Anaconda
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In [5], line 6
      4 print(d[2])
      5 print(d[3])
----> 6 print(d[4])

KeyError: 4
```

Updating the element in a dictionary:

d[key]=value If the key is not available then a new entry will be added to the dictionary with the specified key-value pair.

If the key is already available then odd value will be replaced with the new value.

In [8]:
```
#Example for updating the element in a dictionary.
d={1:"Atharva",2:"Python",3:"Anaconda"}
print(d)
d[4]="Sumit"
print(d)
d[1]="Sun"
print(d)
```

```
{1: 'Atharva', 2: 'Python', 3: 'Anaconda'}
{1: 'Atharva', 2: 'Python', 3: 'Anaconda', 4: 'Sumit'}
{1: 'Sun', 2: 'Python', 3: 'Anaconda', 4: 'Sumit'}
```

Clear function in a dictionary

The clear() method empties the dictionary:

Syntax for clear function is:

dictionary_name.clear()

If we print it always returns an empty dictionary.

In [9]:
```
#Example of clear() function in dictionary.
thisdict = {"brand": "Ford","model": "Mustang","year": 1964}
thisdict.clear()
print(thisdict)
```

```
{}
```

copy() function in dictionary

You cannot copy a dictionary simply by typing dict2 = dict1, because: dict2 will only be a reference to dict1, and changes made in dict1 will automatically also be made in dict2.

There are ways to make a copy, one way is to use the built-in Dictionary method copy().

Syntax for copy() function is:

new_dictionary=dictionary_name.copy()

or can create a copy using

built-in function dict()

new_dictionary=dict(dictionary_name)

In [10]:
```
#Example of using a copy() function in dictionary:
thisdict = {"brand": "Ford","model": "Mustang","year": 1964}
mydict = thisdict.copy()
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

In [11]:
```
#Example of using a inbuilt function dict() in dictionary:
thisdict = {"brand": "Ford","model": "Mustang","year": 1964}
mydict = dict(thisdict)
print(mydict)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

get() function in dictionary

The get() method returns the value of the item with the specified key.

Syntax

dictionary.get(keyname, value)

Parameter Values

Parameter Description

keyname Required. The keyname of the item you want to return the value from value Optional. A value to return if the specified key does not exist. Default value None

In [12]:
```
#Example of usign get() in dictionary
car = {"brand": "Ford","model": "Mustang","year": 1964}

x = car.get("model")

print(x)
```

```
Mustang
```

In [13]:
```
#Example of using get() in dictionary
car = {"brand": "Ford","model": "Mustang","year": 1964}

x = car.get("prize")

print(x)
#In this example key named "prize" is not available then also no error will be disp
#but None output will come as no such key exist.
```

```
None
```

In [14]:
```
#Example of using get() in dictionary
car = {"brand": "Ford","model": "Mustang","year": 1964}

x = car.get("prize",15000)

print(x)
#In this example key named "prize" is not available then as we have provided the va
#while calling get() function then the output will come as it is.
```

```
15000
```

In [15]:
```
#Example of using get() function in dictionary.
#Example of using get() in dictionary
car = {"brand": "Ford","model": "Mustang","year": 1964}

x = car.get("year",1984)

print(x)
#In this example key named year is already available and we are trying to update
#then it will show the previous old one and not a new one.
#If the key is not available then and then only it will show the new value.
```

```
1964
```

keys() function in dictionary.

Also we can access keys using for loop

The keys() method returns a view object. The view object contains the keys of the dictionary, as a list.

The view object will reflect any changes done to the dictionary.

Syntax

dictionary.keys()

In [16]:
```
#Example of returing keys using keys() function in dictionary.
d={1:"Atharva",2:"Python",3:"Anaconda"}
print(d.keys())
for i in d.keys():
    print(i)
```

```
dict_keys([1, 2, 3])
1
2
3
```

values() function in dictionary.

Also we can access values using for loop

The values() method returns a view object. The view object contains the values of the dictionary, as a list.

The view object will reflect any changes done to the dictionary

Syntax

dictionary.values()

In [17]:
```
#Example of returing keys using keys() function in dictionary.
d={1:"Atharva",2:"Python",3:"Anaconda"}
print(d.values())
for i in d.values():
    print(i)
```

```
dict_values(['Atharva', 'Python', 'Anaconda'])
Atharva
Python
Anaconda
```

items() function in dictionary

Both keys and values can be accessed using items() function in dictionary.

The items() method returns a view object. The view object contains the key-value pairs of the dictionary, as tuples in a list.

The view object will reflect any changes done to the dictionary

Syntax

dictionary.items()

In [19]:
```
#Example of returing keys using items() function in dictionary.
d={1:"Atharva",2:"Python",3:"Anaconda"}
```

```
for i,j in d.items():
    print(i,"-",j)
```

```
1 - Atharva
2 - Python
3 - Anaconda
```

update() function in dictionary

The update() method inserts the specified items to the dictionary.

The specified items can be a dictionary, or an iterable object with key value pairs.

Syntax

dictionary.update(iterable)

Parameter Values

Parameter Description

iterable A dictionary or an iterable object with key value pairs, that will be inserted to the dictionary

In [22]:
```
#Example of updating the elements in one dictionary from the new one.
d={1:"Atharva",2:"Python",3:"Anaconda"}
x={4:"Aman",5:"Chinmay"}
d.update(x)
print(d)
print(x)

#note the change in both a update in d and no effect in x.
```

```
{1: 'Atharva', 2: 'Python', 3: 'Anaconda', 4: 'Aman', 5: 'Chinmay'}
{4: 'Aman', 5: 'Chinmay'}
```

In [23]:
```
#Example of update() function if key is not available
car = {"brand": "Ford","model": "Mustang","year": 1964}

car.update({"color": "White"})

print(car)
```

```
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'White'}
```

In [24]:
```
#Example of update() if key is available
d={1:"Atharva",2:"Python",3:"Anaconda"}
x={2:"Aman",5:"Chinmay"}
d.update(x)
print(d)
print(x)

#note the change in both a update in d and no effect in x.
```

```
{1: 'Atharva', 2: 'Aman', 3: 'Anaconda', 5: 'Chinmay'}
{2: 'Aman', 5: 'Chinmay'}
```

setdefault() function in dictionary

The setdefault() method returns the value of the item with the specified key.

If the key does not exist, insert the key, with the specified value, see example below

Syntax

dictionary.setdefault(keyname, value)

Parameter Values Parameter Description

keyname Required. The keyname of the item you want to return the value from value Optional. If the key exist, this parameter has no effect. If the key does not exist, this value becomes the key's value Default value None

In [25]:
```
#Example of setdefault() function in dictionary
car = {"brand": "Ford","model": "Mustang","year": 1964}

x = car.setdefault("model", "Bronco")

print(x)
```

```
Mustang
```

In [27]:
```
#Example of setdefault() function in dictionary.
car = {"brand": "Ford","model": "Mustang","year": 1964}

x = car.setdefault("color", "white")

print(x)
```

```
white
```

Sets in Python

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is unordered, unchangeable*, and unindexed.

A Set is an unordered collection data type that is iterable, mutable and has no duplicate elements.

Set are represented by { } (values enclosed in curly braces)

The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set. This is based on a data structure known as a hash table. Since sets are unordered, we cannot access items using indexes like we do in lists.

However, a set itself is mutable. We can add or remove items from it.

Sets can also be used to perform mathematical set operations like union, intersection, symmetric difference, etc.

As set is unordered index cannot be used for accessing the set element.

Creating Python Sets

A set is created by placing all the items (elements) inside curly braces {}, separated by comma, or by using the built-in set() function.

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.

In [32]:
```
#Example of creating a set
thisset = {"apple", "banana", "cherry"}
print(thisset)
print(type(thisset))
```

```
{'cherry', 'banana', 'apple'}
<class 'set'>
```

In [29]:
```
#Example of set() in which duplicates are not allowed.
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

```
{'cherry', 'banana', 'apple'}
```

In [30]:
```
#Example of set() of any datatype.
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

```
print(set1)
print(set2)
print(set3)
```

```
{'cherry', 'banana', 'apple'}
{1, 3, 5, 7, 9}
{False, True}
```

In [31]:
```
#Example of set() of any datatype.
set1 = {"abc", 34, True, 40, "male"}
print(set1)
```

```
{True, 34, 'male', 40, 'abc'}
```

In [34]:
```
#Example of creating a set using inbuilt function set()
l=[1,2,3,4]
s=set(l)
print(s)
```

```
{1, 2, 3, 4}
```

In [35]:
```
#Example of {}
a={}
print(type(a))
#a={} is considered as an empty dictionary and not as an empty set.
```

```
<class 'dict'>
```

In [36]:
```
#Example of creating empty set using set() inbuilt function.
a=set()
print(type(a))
```

```
<class 'set'>
```

Mathematical Operations on set

1. Union
2. Intersection
3. Difference()
4. Symmetric Difference()

1. union() Method The union() method returns a set that contains all items from the original set, and all items from the specified set(s).

You can specify as many sets you want, separated by commas.

It does not have to be a set, it can be any iterable object.

If an item is present in more than one set, the result will contain only one appearance of this item.

Syntax

set.union(set1, set2...)

Parameter Values

Parameter Description

set1 Required. The iterable to unify with set2 Optional. The other iterable to unify with. You can compare as many iterables as you like. Separate each iterable with a comma

Symbol for union or |.

In [37]:
```
#Example of union() in set.
x = {"a", "b", "c"}
y = {"f", "d", "a"}
z = {"c", "d", "e"}

result = x.union(y, z)

print(result)
```

```
{'b', 'f', 'c', 'e', 'a', 'd'}
```

1. intersection() Method The intersection() method returns a set that contains the similarity between two or more sets.

Meaning: The returned set contains only items that exist in both sets, or in all sets if the comparison is done with more than two sets.

Syntax

set.intersection(set1, set2 ... etc)

Parameter Values Parameter Description

set1 Required. The set to search for equal items in set2 Optional. The other set to search for equal items in. You can compare as many sets you like. Separate the sets with a comma

Symbol for intersection or &.

In [38]:
```
#Example for intersection() in a set.
x = {"a", "b", "c"}
y = {"c", "d", "e"}
z = {"f", "g", "c"}

result = x.intersection(y, z)

print(result)
```

```
{'c'}
```

1. difference() method

The difference() method returns a set that contains the difference between two sets.

Meaning: The returned set contains items that exist only in the first set, and not in both sets.

Syntax

set.difference(set)

Parameter Values Parameter Description set Required. The set to check for differences in

Symbol for difference is difference or -.

In [39]:
```
#Example of difference() in a set.
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = y.difference(x)

print(z)
```

```
{'microsoft', 'google'}
```

1. Symmetric Difference() in a set.

The symmetric_difference() method returns a set that contains all items from both set, but not the items that are present in both sets.

Meaning: The returned set contains a mix of items that are not present in both sets.

Syntax

set.symmetric_difference(set)

Parameter Values

Parameter Description

set Required. The set to check for matches in

Symbol for symmetric_difference is symmetric_difference or -.

In [40]:
```
#Example of symmetric difference mathematical operation in a set.
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}

z = x.symmetric_difference(y)

print(z)
```

```
{'microsoft', 'google', 'cherry', 'banana'}
```

issubset() method in set.

The issubset() method returns True if all items in the set exists in the specified set, otherwise it retuns False.

Syntax

set.issubset(set)

Parameter Values Parameter Description set Required. The set to search for equal items in

In [41]:
```
#Example of issubset() in set
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b"}

z = x.issubset(y)

print(z)
```

```
False
```

In [42]:
```
#Example of issubset() in set
x = {"a", "b", "c"}
y = {"f", "e", "d", "c", "b","a"}

z = x.issubset(y)

print(z)
```

```
True
```

issuperset() method in set. The issuperset() method returns True if all items in the specified set exists in the original set, otherwise it retuns False.

Syntax set.issuperset(set)

Parameter Values Parameter Description set Required. The set to search for equal items in

In [43]:
```
#Example of issuperset() in set

x = {"f", "e", "d", "c", "b"}
y = {"a", "b", "c"}

z = x.issuperset(y)

print(z)
```

```
False
```

In [44]:
```
#Example of issuperset() in set

x = {"f", "e", "d", "c", "b","a"}
y = {"a", "b", "c"}

z = x.issuperset(y)

print(z)
```

```
True
```

copy() function in set The copy() method copies the set.

Syntax set.copy()

In [45]:
```
#Example of copy() function in set
fruits = {"apple", "banana", "cherry"}
```

```
x = fruits.copy()

print(x)
```

```
{'cherry', 'banana', 'apple'}
```

frozenset() in set.

Python frozenset() Method creates an immutable Set object from an iterable. It is a built-in Python function. As it is a set object therefore we cannot have duplicate values in the frozenset.

frozenset() in Python

Syntax : frozenset(iterable_object_name)

Parameter : iterable_object_name

This function accepts iterable object as input parameter. Return : Returns an equivalent frozenset object.

Using frozenset() Method on tuple If no parameters are passed to frozenset() function, then it returns an empty frozenset type object in Python.

In [46]:
```
#Example of frozenset() in set.
l = ["Geeks", "for", "Geeks"]

# converting tuple to frozenset
fnum = frozenset(l)

# printing empty frozenset object
print("frozenset Object is : ", fnum)
```

```
frozenset Object is :  frozenset({'Geeks', 'for'})
```

In [47]:
```
#Example of frozenset() to show immutable() in set.
# creating a list
favourite_subject = ["OS", "DBMS", "Algo"]

# creating a frozenset
f_subject = frozenset(favourite_subject)

# below line will generate error
f_subject[1] = "Networking"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In [47], line 9
      6 f_subject = frozenset(favourite_subject)
      8 # below line will generate error
----> 9 f_subject[1] = "Networking"

TypeError: 'frozenset' object does not support item assignment
```

lambda functions

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax lambda arguments : expression The expression is executed and the result is returned:

Lambda functions can be used whenever function arguments are necessary. In addition to other forms of formulations in functions, it has a variety of applications in certain coding domains. It's important to remember that according to syntax, lambda functions are limited to a single statement.

In [49]:
```
#Example: Lambda Function Example

# Python program to demonstrate # lambda functions
string ='GeeksforGeeks'
# lambda returns a function object
print(lambda string : string)
```

```
<function <lambda> at 0x000002BF9AC46F80>
```

In this above example, the lambda is not being called by the print function but simply returning the function object and the memory location where it is stored. So, to make the print to print the string first we need to call the lambda so that the string will get pass the print.

Difference Between Lambda functions and def defined function Let's look at this example and try to understand the difference between a normal def defined function and lambda function.

In [50]:
```
#Program with a normal function
def squareit(n):
    return n*n
```

In [51]:
```
#Program with a lambda function to find square of a given number:
s=lambda n:n*n
print("The square of 4 is:",s(4))
print("The square of 5 is:",s(5))
```

```
The square of 4 is: 16
The square of 5 is: 25
```

In [52]:
```
#Python Lambda Function with if-else
# Example of lambda function using if-else
Max = lambda a, b : a if(a > b) else b

print(Max(1, 2))
```

```
2
```

Using lambda() Function with filter() The filter() function in Python takes in a function and a list as arguments. This offers an elegant way to filter out all the elements of a sequence "sequence", for which the function returns True. Here is a small program that returns the odd numbers from an input list:

In [53]:
```
# Python code to illustrate # filter() with lambda()

li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(filter(lambda x: (x%2 != 0) , li))
print(final_list)
```

```
[5, 7, 97, 77, 23, 73, 61]
```

In [54]:
```
# Python 3 code to people above 18 yrs
ages = [13, 90, 17, 59, 21, 60, 5]
adults = list(filter(lambda age: age>18, ages))
print(adults)
```

```
[90, 59, 21, 60]
```

Using lambda() Function with map() The map() function in Python takes in a function and a list as an argument. The function is called with a lambda function and a list and a new list is returned which contains all the lambda modified items returned by that function for each item. Example:

In [55]:
```
# Python code to illustrate # map() with lambda()
# to get double of a list.
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(map(lambda x: x*2, li))
print(final_list)
```

```
[10, 14, 44, 194, 108, 124, 154, 46, 146, 122]
```

In [56]:
```
# Python program to demonstrate # use of lambda() function
# with map() function
animals = ['dog', 'cat', 'parrot', 'rabbit']
# here we intend to change all animal names # to upper case and return the same

uppered_animals = list(map(lambda animal: str.upper(animal), animals))
print(uppered_animals)
```

```
['DOG', 'CAT', 'PARROT', 'RABBIT']
```

Using lambda() Function with reduce() The reduce() function in Python takes in a function and a list as an argument. The function is called with a lambda function and an iterable and a new reduced result is returned. This performs a repetitive operation over the pairs of the iterable. The reduce() function belongs to the functools module.

In [57]:
```
# Python code to illustrate # reduce() with lambda() # to get sum of a list
from functools import reduce
li = [5, 8, 10, 20, 50, 100]
sum = reduce((lambda x, y: x + y), li)
print (sum)
```

```
193
```

In [58]:
```
# python code to demonstrate working of reduce() # with a lambda function
# importing functools for reduce()
import functools
# initializing list
lis = [ 1 , 3, 5, 6, 2, ]
# using reduce to compute maximum element from list
print ("The maximum element of the list is : ",end="")
print (functools.reduce(lambda a,b : a if a > b else b, lis))
```

```
The maximum element of the list is : 6
```

In [60]:
```
#1.     Write a Python program to find the sum of all the elements in the list.
total = 0
list1 = [2,6,9,10,56]

for num in range(0, len(list1)):
    total = total + list1[num]
print("Sum of all elements in given list: ", total)
```