UNIT NO. 3

DATA STRUCTURES IN PYTHON

Python List

1) A list in Python is used to store the sequence of various types of data.
2) Python lists are mutable type its mean we can modify its element after it created.
3) Python consists of six data-types that are capable to store the sequences, but the most common and reliable type is the list.
4) A list can be defined as a collection of values or items of different types. The items in the list are separated with the comma (,) and enclosed with the square brackets [].

A list can be define as below

```
L1 = ["Neer", 21, "Nashik"]
L2 = [1, 2, 3, 4, 5, 6]
```

If we try to print the type of L1, L2, and L3 using type() function then it will come out to be a list.

```
print(type(L1))
print(type(L2))
```

Characteristics of Lists

1) The lists are ordered.
2) The element of the list can access by index.
3) The lists are the mutable type.
4) The lists are mutable types.
5) A list can store the number of various elements.

Let's check the first statement that lists are the ordered.

```
a = [1,2,"Neer",4.50,"Pratyush",5,6]
b = [1,2,5,"Neer",4.50,"Pratyush",6]
a ==b
```

Output: false

Both lists have consisted of the same elements, but the second list changed the index position of the 5th element that violates the order of lists. When compare both lists it returns the false.

Lists maintain the order of the element for the lifetime. That's why it is the ordered collection of objects

```
a = [1, 2,"Neer", 4.50,"Pratyush",5, 6]
b = [1, 2,"Neer", 4.50,"Pratyush",5, 6]
a == b
```

Output: True

Let's have a look at the list example in detail.

```
emp = ["John", 102, "USA"]
Dep1 = ["CS",10]
Dep2 = ["IT",11]
HOD_CS = [10,"Mr. Holding"]
HOD_IT = [11, "Mr. Bewon"]
print("printing employee data...")
print("Name : %s, ID: %d, Country: %s"%(emp[0],emp[1],emp[2]))
print("printing departments...")
print("Department 1:\nName: %s, ID: %d\nDepartment 2:\nName: %s, ID: %s"%(Dep1[0],
Dep2[1],Dep2[0],Dep2[1]))
print("HOD Details ....")
print("CS HOD Name: %s, Id: %d"%(HOD_CS[1],HOD_CS[0]))
print("IT HOD Name: %s, Id: %d"%(HOD_IT[1],HOD_IT[0]))
print(type(emp),type(Dep1),type(Dep2),type(HOD_CS),type(HOD_IT))
```

Output:

```
printing employee data...
Name : John, ID: 102, Country: USA
printing departments...
Department 1:
Name: CS, ID: 11
Department 2:
Name: IT, ID: 11
HOD Details ....
CS HOD Name: Mr. Holding, Id: 10
IT HOD Name: Mr. Bewon, Id: 11
<class 'list'> <class 'list'> <class 'list'> <class 'list'> <class 'list'>
```

1) Accessing Values in List

1) The indexing is processed in the same way as it happens with the strings. The elements of the list can be accessed by using the slice operator [ ].

2) The index starts from 0 and goes to length - 1. The first element of the list is stored at the 0th index, the second element of the list is stored at the 1st index, and so on.

List = [0, 1, 2, 3, 4, 5]

0	**1**	**2**	**3**	**4**	**5**

List[0] = 0

List[1] = 1

List[2] = 2

List[3] = 3

List[4] = 4

List[5] = 5

List[0:] = [0,1,2,3,4,5]

List[:] = [0,1,2,3,4,5]

List[2:4] = [2, 3]

List[1:3] = [1, 2]

List[:4] = [0, 1, 2, 3]

We can get the sub-list of the list using the following syntax.

1. list_varible(start:stop:step)
   - The **start** denotes the starting index position of the list.
   - The **stop** denotes the last index position of the list.
   - The **step** is used to skip the nth element within a **start:stop**

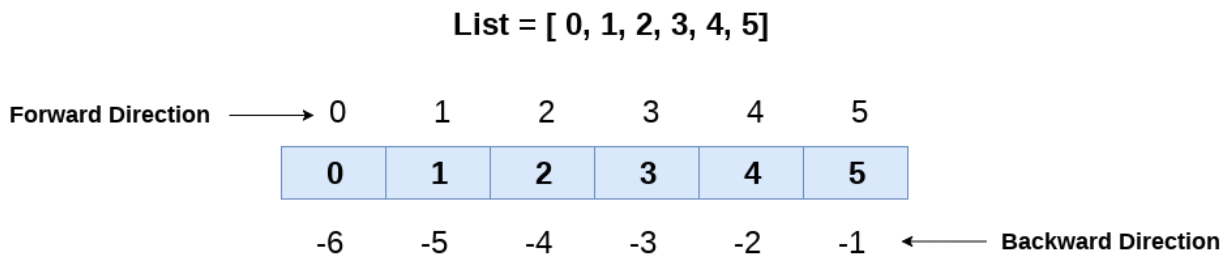
Consider the following example:

```
list = [1,2,3,4,5,6,7]
print(list[0])
print(list[1])
print(list[2])
print(list[3])
print(list[0:6])                    # Slicing the elements
print(list[:])
print(list[2:5])
print(list[1:6:2])
```

Output:

```
1
2
3
4
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6, 7]
[3, 4, 5]
[2, 4, 6]
```

Unlike other languages, Python provides the flexibility to use the negative indexing also. The negative indices are counted from the right. The last element (rightmost) of the list has the index -1; its adjacent left element is present at the index -2 and so on until the left-most elements are encountered.



Let's have a look at the following example where we will use negative indexing to access the elements of the list.

```
list = [1,2,3,4,5]
print(list[-1])
print(list[-3:])
print(list[:-1])
print(list[-3:-1])
```

Output:

```
5
[3, 4, 5]
[1, 2, 3, 4]
[3, 4]
```

As we discussed above, we can get an element by using negative indexing. In the above code, the first print statement returned the rightmost element of the list. The second print statement returned the sub-list, and so on.

2] Updating List values

1) Lists are the most versatile data structures in Python since they are mutable, and their values can be updated by using the slice and assignment operator.

2) Python also provides append() and insert() methods, which can be used to add values to the list.

3)Ex.

```
 list = [1, 2, 3, 4, 5, 6]
print(list)
list[2] = 10                    # It will assign value to the value to the second index
print(list)
list[1:3] = [89, 78]        # Adding multiple-element
print(list)
list[-1] = 25              # It will add value at the end of the list
print(list)
```

Output:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 10, 4, 5, 6]
[1, 89, 78, 4, 5, 6]
[1, 89, 78, 4, 5, 25]
```

3] Deleting Values in List

1) The list elements can also be deleted by using the **del** keyword.

2) Python also provides us the **remove()** method if we do not know which element is to be deleted from the list.

Consider the following example to delete the list elements.

```
list = [‘Python’, ‘Java’,  22616, 22]
print(list)
del list[2]
print(After deleting value at index 2)
print(list)
```

Python List Operations

The concatenation (+) and repetition (*) operators work in the same way as they were working with the strings.

Let's see how the list responds to various operators.

1. Consider a Lists l1 = [1, 2, 3, 4], **and** l2 = [5, 6, 7, 8] to perform operation.

Operator	Description	Example
Repetition	The repetition operator enables the list elements to be repeated multiple times.	L1*2 = [1, 2, 3, 4, 1, 2, 3, 4]
Concatenation	It concatenates the list mentioned on either side of the operator.	l1+l2 = [1, 2, 3, 4, 5, 6, 7, 8]
Membership	It returns true if a particular item exists in a particular list otherwise false.	print(2 in l1) prints True.
Iteration	The for loop is used to iterate over the list elements.	for i in l1: print(i) **Output** 1 2 3 4
Length	It is used to get the length of the list	len(l1) = 4

Iterating a List

A list can be iterated by using a for - in loop. A simple list containing four strings, which can be iterated as follows.

```
list = ["Neer", "Pratyush", "ABC", "XYZ"]
for i in list:
 print(i)                    # The i variable will iterate over the elements of the List
                             and contains each element in each iteration.
```

Output:

```
Neer
Pratyush
ABC
XYZ
```

Adding elements to the list

Python provides append() function which is used to add an element to the list. However, the append() function can only add value to the end of the list.

Consider the following example in which, we are taking the elements of the list from the user and printing the list on the console.

```
    l =[]                                                  #Declaring the empty list
    n = int(input("Enter the number of elements in the list:"))
    for i in range(0,n):                       # for loop to take the input
    append(input("Enter the item:"))
    print("printing the list items..")
    for i in l:                            # traversal loop to print the list items
print(i, end = "  ")
```

Output:

```
Enter the number of elements in the list:5
Enter the item:25
Enter the item:46
Enter the item:12
Enter the item:75
Enter the item:42
printing the list items
25  46  12  75  42
```

Removing elements from the list

Python provides the **remove()** function which is used to remove the element from the list. Consider the following example to understand this concept.

Example -

```
 list = [0,1,2,3,4]
print("printing original list: ");
for i in list:
    print(i,end=" ")
list.remove(2)
print("\nprinting the list after the removal of first element...")
for i in list:
    print(i,end=" ")
```

Output:

```
printing original list:
0 1 2 3 4
printing the list after the removal of first element...
0 1 3 4
```

Python List Built-in functions

SN	Function	Description	Example
1	cmp(list1, list2)	It compares the elements of both the lists.	This method is not used in the Python 3 and the above versions.

2	len(list)	It is used to calculate the length of the list.	L1 = [1,2,3,4,5,6,7,8]
			print(len(L1))
			o/p: 8

3	max(list)	It returns the maximum element of the list.	L1 = [12,34,26,48,72]
			print(max(L1))

			o/p: 72

4	min(list)	It returns the minimum element of the list.	L1 = [12,34,26,48,72]
			print(min(L1))
			o/p: 12

5	list(seq)	It converts any sequence to the list.	str = "Johnson"
			s = list(str)
			print(type(s))
			o/p: <class list>

Let's have a look at the few list examples.

Example: 1- Write the program to remove the duplicate element of the list

```
list1 = [1,2,2,3,55,98,65,65,13,29]
list2 = []                  # Declare an empty list that will store unique values
for i in list1:
   if i not in list2:
      list2.append(i)
print(list2)
```

Output:

```
[1, 2, 3, 55, 98, 65, 13, 29]
```

Example:2- Write a program to find the sum of the element in the list

```
list1 = [3,4,5,9,10,12,24]
sum = 0
for i in list1:
   sum = sum+i
print("The sum is:",sum)
```

Output:

```
The sum is: 67
```

Example: 3- Write the program to find the lists consist of at least one common element.

```
list1 = [1,2,3,4,5,6]
list2 = [7,8,9,2,10]
for x in list1:
    for y in list2:
        if x == y:
            print("The common element is:",x)
```

Output:

```
The common element is: 2
```

Python Tuple

1) Python Tuple is used to store the sequence of immutable Python objects.

2) The tuple is similar to lists since the value of the items stored in the list can be changed, whereas the tuple is immutable, and the value of the items stored in the tuple cannot be changed.

Creating a tuple

A tuple can be written as the collection of comma-separated (,) values enclosed with the small () brackets. The parentheses are optional but it is good practice to use. A tuple can be defined as follows.

```
T1 = (6, "Pratyush", 14)
T2 = ("Apple", "Banana", "Orange")
T3 = 10,20,30,40,50

print(type(T1))
print(type(T2))
print(type(T3))
```

Output:

```
<class 'tuple'>
<class 'tuple'>
```

```
<class 'tuple'>
```

An empty tuple can be created as follows.

T4 = ( )

Creating a tuple with single element is slightly different. We will need to put comma after the element to declare the tuple.

```
tup1 = ("Python")
print(type(tup1))
#Creating a tuple with single element
tup2 = ("Python",)
print(type(tup2))
```

Output:

```
<class 'str'>
<class 'tuple'>
```

A tuple is indexed in the same way as the lists. The items in the tuple can be accessed by using their specific index value.

Consider the following example of tuple:

Ex- 1

```
tuple1 = (10, 20, 30, 40, 50, 60)
  print(tuple1)
  count = 0
  for i in tuple1:
      print("tuple1[%d] = %d"%(count, i))
      count = count+1
```

Ex - 2

```
tuple1 = tuple(input("Enter the tuple elements ..."))
print(tuple1)
count = 0
for i in tuple1:
    print("tuple1[%d] = %s"%(count, i))
```

```
        count = count+1
```

A tuple is indexed in the same way as the lists. The items in the tuple can be accessed by using their specific index value.

We will see all these aspects of tuple in this section of the tutorial.

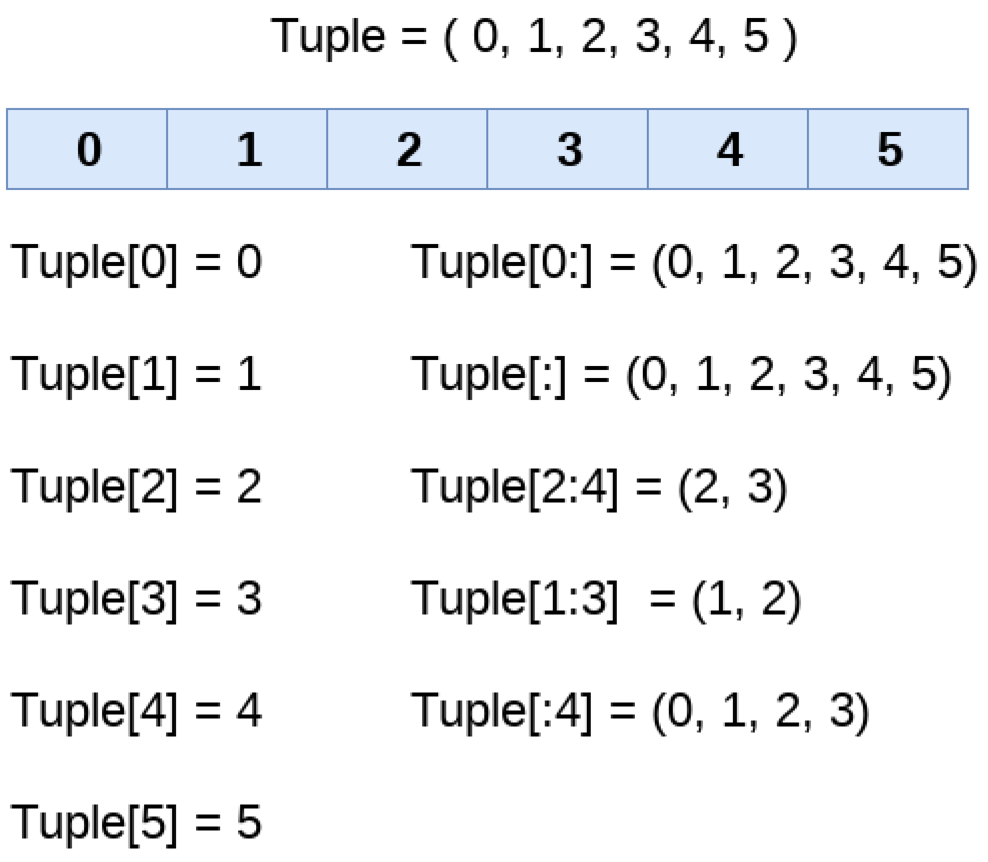
1) Accessing Values in Tuples:

1) To access values in tuples use the square brackets for slicing along with the index or indices to obtain values available at that index.

2) The indexing and slicing in the tuple are similar to lists. The indexing in the tuple starts from 0 and goes to length(tuple) - 1.

3) The items in the tuple can be accessed by using the index [] operator. Python also allows us to use the colon operator to access multiple items in the tuple.

4) Consider the following image to understand the indexing and slicing in detail.



Consider the following example:

```
tup = (1,2,3,4,5,6,7)
print(tup[0])
print(tup[1])
print(tup[2])
# It will give the IndexError
print(tup[8])
```

Output:

```
1
2
3
tuple index out of range
```

In the above code, the tuple has 7 elements which denote 0 to 6. We tried to access an element outside of tuple that raised an **IndexError**.

```
tuple = (1,2,3,4,5,6,7)
print(tuple[1:])              #element 1 to end
print(tuple[:4])              #element 0 to 3 element
print(tuple[1:5])             #element 1 to 4 element
print(tuple[0:6:2])           # element 0 to 6 and take step of 2
```

Output:

```
(2, 3, 4, 5, 6, 7)
(1, 2, 3, 4)
(1, 2, 3, 4)
(1, 3, 5)
```

2) Updating Tuples:

1) Tuples are immutable which means you cannot update or change the values of tuples elements.

2) you are able to take portions of exiting tuples to create new tuples.

3) e.g.

```
tup1=(11,17,27)

tup2=('abc', 'xyz')

tup3=tup1+tup2;

print tup3
```

3) Deleting Tuple

1) Unlike lists, the tuple items cannot be deleted by using the **del** keyword as tuples are immutable.

2) To delete an entire tuple, we can use the **del** keyword with the tuple name.

3) Consider the following example.

```
tuple1 = (1, 2, 3, 4, 5, 6)
print(tuple1)
del tuple1[0]
print(tuple1)
del tuple1
print(tuple1)
```

Basic Tuple operations

The operators like concatenation (+), repetition (*), Membership (in) works in the same way as they work with the list. Consider the following table for more detail.

Let's say Tuple t = (1, 2, 3, 4, 5) and Tuple t1 = (6, 7, 8, 9) are declared.

Operator	Description	Example

Repetition	The repetition operator enables the tuple elements to be repeated multiple times.	T1*2 = (1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
Concatenation	It concatenates the tuple mentioned on either side of the operator.	T1+T2 = (1, 2, 3, 4, 5, 6, 7, 8, 9)
Membership	It returns true if a particular item exists in the tuple otherwise false	print (2 in T1) prints True.
Iteration	The for loop is used to iterate over the tuple elements.	for i in T1: print(i) **Output** 1 2 3 4 5
Length	It is used to get the length of the tuple.	len(T1) = 5

Python Tuple inbuilt functions

SN	Function	Description
1	cmp(tuple1, tuple2)	It compares two tuples and returns true if tuple1 is greater than tuple2 otherwise false.
2	len(tuple)	It calculates the length of the tuple.
3	max(tuple)	It returns the maximum element of the tuple
4	min(tuple)	It returns the minimum element of the tuple.

5	tuple(seq)	It converts the specified sequence to the tuple.

Where use tuple?

Using tuple instead of list is used in the following scenario.

1. Using tuple instead of list gives us a clear idea that tuple data is constant and must not be changed.

2. Tuple can simulate a dictionary without keys. Consider the following nested structure, which can be used as a dictionary.

```
1. [(101, "John", 22), (102, "Mike", 28),  (103, "Dustin", 30)]
```

List vs. Tuple

SN	List	Tuple
1	The literal syntax of list is shown by the [].	The literal syntax of the tuple is shown by the ().
2	The List is mutable.	The tuple is immutable.
3	The List has the a variable length.	The tuple has the fixed length.
4	The list provides more functionality than a tuple.	The tuple provides less functionality than the list.
5	The list is used in the scenario in which we need to store the simple collections with no constraints where the value of the items can be changed.	The tuple is used in the cases where we need to store the read-only collections i.e., the value of the items cannot be changed. It can be used as the key inside the dictionary.
6	The lists are less memory efficient than a tuple.	The tuples are more memory efficient because of its

		immutability.

Python Set

1) A Python set is the collection of the unordered element.

2) Each element in the set must be unique, immutable, and the sets remove the duplicate elements.

3) Sets are mutable which means we can modify it after its creation.

4) Unlike other collections in Python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index.

5) However, we can print them all together, or we can get the list of elements by looping through the set.

Creating a set

The set can be created by enclosing the comma-separated immutable elements with the curly braces {}. Python also provides the set() method, which can be used to create the set by the passed sequence.

Ex 1: Using curly braces

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"}
    print(Days)
    print(type(Days))
    print("looping through the set elements ... ")
    for i in Days:
        print(i)
```

Output

```
{'Friday', 'Tuesday', 'Monday', 'Saturday', 'Thursday', 'Sunday', 'Wednesday'}
<class 'set'>
```

```
looping through the set elements ...
Friday
Tuesday
Monday
Saturday
Thursday
Sunday
Wednesday
```

Ex 2: Using set() method

```
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"])
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
 print(i)
```

Output:

```
{'Friday', 'Wednesday', 'Thursday', 'Saturday', 'Monday', 'Tuesday', 'Sunday'}
<class 'set'>
looping through the set elements ...
Friday
Wednesday
Thursday
Saturday
Monday
Tuesday
Sunday
```

It can contain any type of element such as integer, float, tuple etc. But mutable elements (list, dictionary, set) can't be a member of set. Consider the following example.

```
# Creating a set which have immutable elements
set1 = {1,2,3, "Python", 20.5, 14}
print(type(set1))
#Creating a set which have mutable element
set2 = {1,2,3,["Python",4]}
print(type(set2))
```

In the above code, we have created two sets, the set **set1** have immutable elements and set2 have one mutable element as a list. While checking the type of set2, it raised an error, which means set can contain only immutable elements.

Creating an empty set is a bit different because empty curly {} braces are also used to create a dictionary as well. So Python provides the set() method used without an argument to create an empty set.

```
# Empty curly braces will create dictionary
set3 = {}
print(type(set3))

# Empty set using set() function
set4 = set()
print(type(set4))
```

Output:

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

Let's see what happened if we provide the duplicate element to the set.

```
set5 = {1,2,4,4,5,8,9,9,10}
print("Return set with unique elements:",set5)
```

Output:

```
Return set with unique elements: {1, 2, 4, 5, 8, 9, 10}
```

In the above code, we can see that **set5** consisted of multiple duplicate elements when we printed it remove the duplicity from the set.

Adding items to the set

Python provides the **add()** method and **update()** method which can be used to add some particular item to the set. The add() method is used to add a single element whereas the update() method is used to add multiple elements to the set. Consider the following example.

Example: 1 - Using add() method

```
Months = set(["January","February", "March", "April", "May", "June"])
```

```
print("\nprinting the original set ... ")
print(months)
print("\nAdding other months to the set...");
Months.add("July");
Months.add ("August");
print("\nPrinting the modified set...");
print(Months)
print("\nlooping through the set elements ... ")
for i in Months:
    print(i)
```

To add more than one item in the set, Python provides the **update()** method. It accepts iterable as an argument.

Consider the following example.

Example - 2 Using update() function

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nupdating the original set ... ")
\Months.update(["July","August","September","October"]);
print("\nprinting the modified set ... ")
print(Months);
```

Removing items from the set

Python provides the **discard()** method and **remove()** method which can be used to remove the items from the set. The difference between these function, using discard() function if the item does not exist in the set then the set remain unchanged whereas remove() method will through an error.

Consider the following example.

Example-1 Using discard() method

```
months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(months)
print("\nRemoving some months from the set...");
months.discard("January");
months.discard("May");
print("\nPrinting the modified set...");
print(months)
print("\nlooping through the set elements ... ")
for i in months:
    print(i)
```

Python provides also the **remove()** method to remove the item from the set. Consider the following example to remove the items using **remove()** method.

Example-2 Using remove() function

```
months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(months)
print("\nRemoving some months from the set...");
months.remove("January");
months.remove("May");
print("\nPrinting the modified set...");
print(months)
```

We can also use the pop() method to remove the item. Generally, the pop() method will always remove the last item but the set is unordered, we can't determine which element will be popped from set.

Consider the following example to remove the item from the set using pop() method.

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving some months from the set...");
```

```
Months.pop();
Months.pop();
print("\nPrinting the modified set...");
print(Months)
```

In the above code, the last element of the **Month** set is **March** but the pop() method removed the **June and January** because the set is unordered and the pop() method could not determine the last element of the set.

Python provides the clear() method to remove all the items from the set.

Consider the following example.

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving all the items from the set...");
Months.clear()
print("\nPrinting the modified set...")
print(Months)
```

Difference between discard() and remove()

Despite the fact that **discard()** and **remove()** method both perform the same task, There is one main difference between discard() and remove().

If the key to be deleted from the set using discard() doesn't exist in the set, the Python will not give the error. The program maintains its control flow.

On the other hand, if the item to be deleted from the set using remove() doesn't exist in the set, the Python will raise an error.

Consider the following example.

Example-

```
Months = set(["January","February", "March", "April", "May", "June"])
print("\nprinting the original set ... ")
print(Months)
print("\nRemoving items through discard() method...");
```

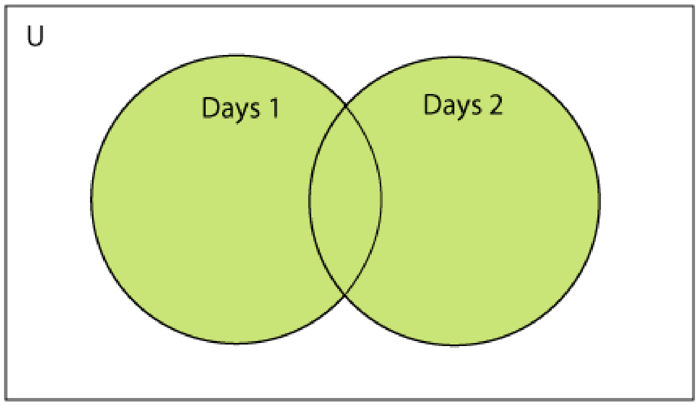
```
Months.discard("Feb"); #will not give an error although the key feb is not available in the set
print("\nprinting the modified set...")
print(Months)
print("\nRemoving items through remove() method...");
Months.remove("Jan") #will give an error as the key jan is not available in the set.
print("\nPrinting the modified set...")
print(Months)
```

Python Set Operations

Set can be performed mathematical operation such as union, intersection, difference, and symmetric difference. Python provides the facility to carry out these operations with operators or methods. We describe these operations as follows.

1) Union of two Sets

The union of two sets is calculated by using the pipe (|) operator. The union of the two sets contains all the items that are present in both the sets.



Consider the following example to calculate the union of two sets.
Ex 1: using union | operator

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday", "Sunday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1|Days2) #printing the union of the sets
```

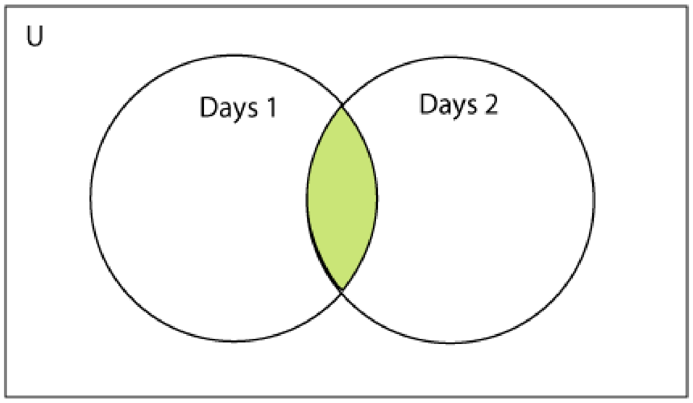
Python also provides the **union()** method which can also be used to calculate the union of two sets. Consider the following example.

Ex 2: using union() method

```
Days1 = {"Monday","Tuesday","Wednesday","Thursday"}
Days2 = {"Friday","Saturday","Sunday"}
print(Days1.union(Days2)) #printing the union of the sets
```

2) Intersection of two sets

The intersection of two sets can be performed by the **and &** operator or the **intersection() function**. The intersection of the two sets is given as the set of the elements that common in both sets.



Consider the following example.

Example 1: Using & operator

```
Days1 = {"Monday","Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday","Tuesday","Sunday", "Friday"}
print(Days1&Days2) #prints the intersection of the two sets
```

Example 2: Using intersection() method

```
set1 = {"Devansh","John", "David", "Martin"}
set2 = {"Steve", "Milan", "David", "Martin"}
print(set1.intersection(set2)) #prints the intersection of the two sets
```

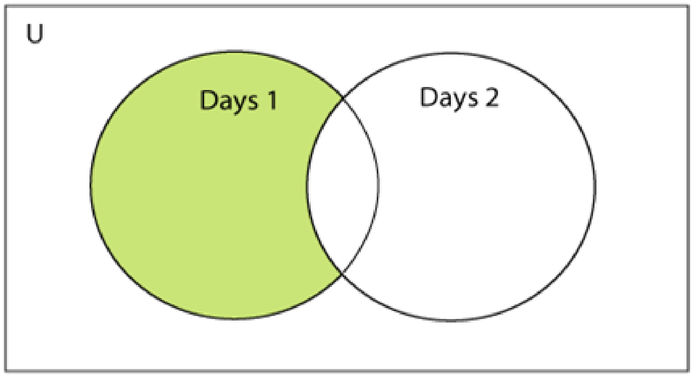
Example 3:

```
set1 = {1,2,3,4,5,6,7}
```

```
set2 = {1,2,20,32,5,9}
set3 = set1.intersection(set2)
print(set3)
```

3) Difference between the two sets

The difference of two sets can be calculated by using the subtraction (-) operator or **intersection()** method. Suppose there are two sets A and B, and the difference is A-B that denotes the resulting set will be obtained that element of A, which is not present in the set B.



Consider the following example.

Example 1 : Using subtraction (-) operator

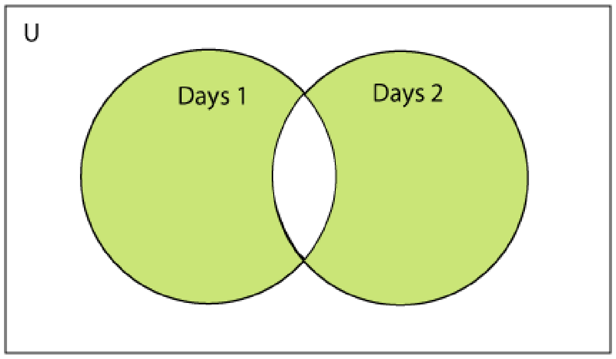
```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1-Days2) #{"Wednesday", "Thursday" will be printed}
```

Example 2 : Using difference() method

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday", "Sunday"}
print(Days1.difference(Days2)) # prints the difference of the two sets Days1 and Days2
```

Symmetric Difference of two sets

The symmetric difference of two sets is calculated by ^ operator or **symmetric_difference()** method. Symmetric difference of sets, it removes that element which is present in both sets. Consider the following example:



Example - 1: Using ^ operator

```
a = {1,2,3,4,5,6}
b = {1,2,9,8,10}
c = a^b
print(c)
```

Example - 2: Using symmetric_difference() method

```
a = {1,2,3,4,5,6}
b = {1,2,9,8,10}
c = a.symmetric_difference(b)
print(c)
```

Set comparisons

Python allows us to use the comparison operators i.e., <, >, <=, >= , == with the sets by using which we can check whether a set is a subset, superset, or equivalent to other set. The boolean true or false is returned depending upon the items present inside the sets.

Consider the following example.

```
Days1 = {"Monday",  "Tuesday", "Wednesday", "Thursday"}
Days2 = {"Monday", "Tuesday"}
```

```
Days3 = {"Monday", "Tuesday", "Friday"}
print (Days1>Days2)     #Days1 is the superset of Days2 hence it will print true.
print (Days1<Days2)    #prints false since Days1 is not the subset of Days2
print (Days2 == Days3)       #prints false since Days2 and Days3 are not equivalent
```

Set Programming Example

Ex - 1: Write a program to remove the given number from the set.

```
my_set = {1,2,3,4,5,6,12,24}
n = int(input("Enter the number you want to remove"))
my_set.discard(n)
print("After Removing:",my_set)
```

Ex - 2: Write a program to add multiple elements to the set.

```
set1 = set([1,2,4,"ABC","XYZ"])
set1.update(["Apple","Mango","Grapes"])
print(set1)
```

Ex - 3: Write a program to find the union between two set.

```
set1 = set(["Pratyush","Neer", 65,59,96])
set2  = set(["Pratyush",1,2,"Neer"])
set3 = set1.union(set2)
print(set3)
```

Ex- 4: Write a program to find the intersection between two sets.

```
set1 = {23,44,56,67,90,45,"Python"}
set2 = {13,23,56,76,"ABC"}
set3 = set1.intersection(set2)
print(set3)
```

Python Built-in set methods

SN	Method	Description
1	add(item)	It adds an item to the set. It has no effect if the item is already present in the set.

2	clear()	It deletes all the items from the set.
3	copy()	It returns a shallow copy of the set.
4	difference_update(....)	It modifies this set by removing all the items that are also present in the specified sets.
5	discard(item)	It removes the specified item from the set.
6	intersection()	It returns a new set that contains only the common elements of both the sets. (all the sets if more than two are specified).
7	intersection_update(....)	It removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).
8	Isdisjoint(....)	Return True if two sets have a null intersection.
9	Issubset(....)	Report whether another set contains this set.
10	Issuperset(....)	Report whether this set contains another set.
11	pop()	Remove and return an arbitrary set element that is the last element of the set. Raises KeyError if the set is empty.
12	remove(item)	Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.
13	symmetric_difference (....)	Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.
14	symmetric_difference_update(....)	Update a set with the symmetric difference of itself and another.
15	union(....)	Return the union of sets as a new set. (i.e. all elements that are in either set.)
16	update()	Update a set with the union of itself and others.

Python Dictionary

1) Python Dictionary is used to store the data in a key-value pair format.

2) The dictionary is the data type in Python, which can simulate the real-life data arrangement where some specific value exists for some particular key.

3) It is the mutable data-structure. The dictionary is defined into element Keys and values.

- Keys must be a single element
- Value can be any type such as list, tuple, integer, etc.

4) In other words, we can say that a dictionary is the collection of key-value pairs where the value can be any Python object.

5) the keys are the immutable Python object, i.e., Numbers, string, or tuple.

Creating the dictionary

The dictionary can be created by using multiple key-value pairs enclosed with the curly brackets { }, and each key is separated from its value by the colon (:).

Syntax:

```
Dict = {"Name": "Pratyush", "Age": 7}
```

In the above dictionary **Dict**, The keys **Name** and **Age** are the string that is an immutable object.

Let's see an example to create a dictionary and print its content.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
```

Python provides the built-in function **dict()** method which is also used to create dictionary. The empty curly braces { } is used to create empty dictionary.

```
Dict = {}                      # Creating an empty Dictionary
print("Empty Dictionary: ")
print(Dict)
Dict = dict({1: 'Java', 2: 'T', 3:'Point'})              # Creating a Dictionary with dict() method
print("\nCreate Dictionary by using  dict(): ")
print(Dict)
Dict = dict([(1, 'Devansh'), (2, 'Sharma')])    # Creating a Dictionary  with each item as a Pair
print("\nDictionary with each item as a pair: ")
print(Dict)
```

1) Accessing the dictionary values

the values can be accessed in the dictionary by using the keys as keys are unique in the dictionary.

The dictionary values can be accessed in the following way.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print("Name : %s" %Employee["Name"])
print("Age : %d" %Employee["Age"])
print("Salary : %d" %Employee["salary"])
print("Company : %s" %Employee["Company"])
```

2) Adding dictionary values

1) The dictionary is a mutable data type, and its values can be updated by using the specific keys.
2) The value can be updated along with key **Dict[key] = value**. The update() method is also used to update an existing value.
3) If the key-value already present in the dictionary, the value gets updated. Otherwise, the new keys added in the dictionary.

Ex - 1:

```
Dict = {}                                    # Creating an empty Dictionary
print("Empty Dictionary: ")
print(Dict)
Dict[0] = 'Peter'                            # Adding elements to dictionary one at a time
Dict[2] = 'Joseph'
Dict[3] = 'Ricky'
```

```
 print("\nDictionary after adding 3 elements: ")
 print(Dict)
Dict['Emp_ages'] = 20, 33, 24                    # Adding set of values   with a single Key
 print("\nDictionary after adding 3 elements: ")
 print(Dict)
 Dict[3] = 'Jay'                                  # Updating existing Key's Value
print("\nUpdated key value: ")
 print(Dict)
```

Ex - 2:

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
print("Enter the details of the new employee....");
Employee["Name"] = input("Name: ");
Employee["Age"] = int(input("Age: "));
Employee["salary"] = int(input("Salary: "));
Employee["Company"] = input("Company:");
print("printing the new data");
print(Employee)
```

Deleting elements using del keyword

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
 print(type(Employee))
 print("printing Employee data .... ")
 print(Employee)
 print("Deleting some of the employee data")
 del Employee["Name"]
 del Employee["Company"]
 print("printing the modified information ")
 print(Employee)
 print("Deleting the dictionary: Employee");
```

```
del Employee
print("Lets try to print it again ");
print(Employee)
```

.

- **Using pop() method**

The **pop()** method accepts the key as an argument and remove the associated value. Consider the following example.

```
Dict = {1: 'Python', 2: 'ETI', 3: 'MAD'}        # Creating a Dictionary
pop_ele = Dict.pop(3)                           # Deleting a key using pop() method
print(Dict)
```

Python also provides a built-in methods popitem() and clear() method for remove elements from the dictionary. The popitem() removes the arbitrary element from a dictionary, whereas the clear() method removes all elements to the whole dictionary.

Iterating Dictionary

A dictionary can be iterated using for loop as given below.

Ex 1

# for loop to print all the keys of a dictionary

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee:
    print(x)
```

Ex 2

#for loop to print all the values of the dictionary

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee:
    print(Employee[x])
```

Ex - 3

#for loop to print the values of the dictionary by using values() method.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
for x in Employee.values():
    print(x)
```

Ex 4

#for loop to print the items of the dictionary by using items() method.

```
Employee = {"Name": "John", "Age": 29, "salary":25000,"Company":"GOOGLE"}
    for x in Employee.items():
        print(x)
```

Properties of Dictionary keys

1) In the dictionary, we cannot store multiple values for the same keys. If we pass more than one value for a single key, then the value which is last assigned is considered as the value of the key.

Consider the following example.

```
Employee={"Name":"John","Age":29,"Salary":25000,"Company":"GOOGLE","Name":"John"}
  for x,y in Employee.items():
  print(x,y)
```

2) In python, the key cannot be any mutable object. We can use numbers, strings, or tuples as the key, but we cannot use any mutable object like the list as the key in the dictionary.

Consider the following example.

```
Emp = {"Name": "John", "Age": 29, "sal":25000,"Comp":"TCS",[100,201,301]:"Dept ID"}

for x,y in Emp.items():
        print(x,y)
```

Built-in Dictionary functions

The built-in python dictionary methods along with the description are given below.

SN	Function	Description
1	cmp(dict1, dict2)	It compares the items of both the dictionary and returns true if the first dictionary values are greater than the second dictionary, otherwise it returns false.
2	len(dict)	It is used to calculate the length of the dictionary.
3	str(dict)	It converts the dictionary into the printable string representation.
4	type(variable)	It is used to print the type of the passed variable.

Built-in Dictionary methods

SN	Method	Description
1	dic.clear()	It is used to delete all the items of the dictionary.
2	dict.copy()	It returns a shallow copy of the dictionary.
3	dict.fromkeys(iterable, value = None, /)	Create a new dictionary from the iterable with the values equal to value.
4	dict.get(key, default = "None")	It is used to get the value specified for the passed key.
5	dict.has_key(key)	It returns true if the dictionary contains the specified key.
6	dict.items()	It returns all the key-value pairs as a tuple.
7	dict.keys()	It returns all the keys of the dictionary.
8	dict.setdefault(key,default= "None")	It is used to set the key to the default value if the key is not specified in the dictionary
9	dict.update(dict2)	It updates the dictionary by adding the key-value pair of dict2 to this dictionary.
10	dict.values()	It returns all the values of the dictionary.
11	len()	

12	popItem()	
13	pop()	
14	count()	
15	index()	