PYTHON

LIST

List in python is implemented to store the sequence of various type of data. However, python contains six data types that are capable to store the sequences but the most common and reliable type is list.

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 defined as follows.

```
L1 = ["John", 102, "USA"]
L2 = [1, 2, 3, 4, 5, 6]
L3 = [1, "Ryan"]
```

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

Lets consider a proper example to define a list and printing its values.

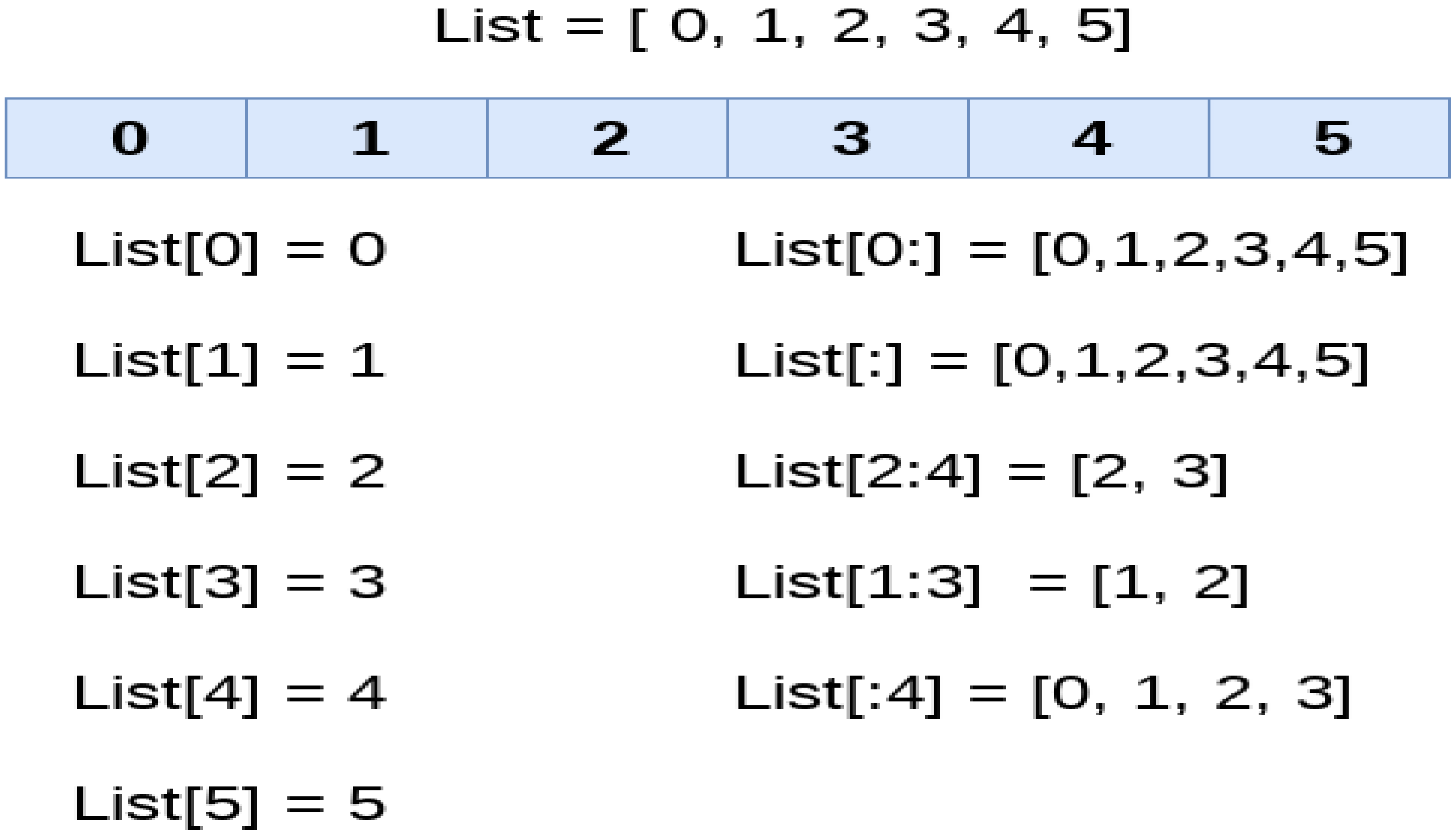
```
emp = ["John", 102, "USA"]

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],De
p2[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));
```

LIST INDEXING AND SPLITTING

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

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.

Consider the following example.

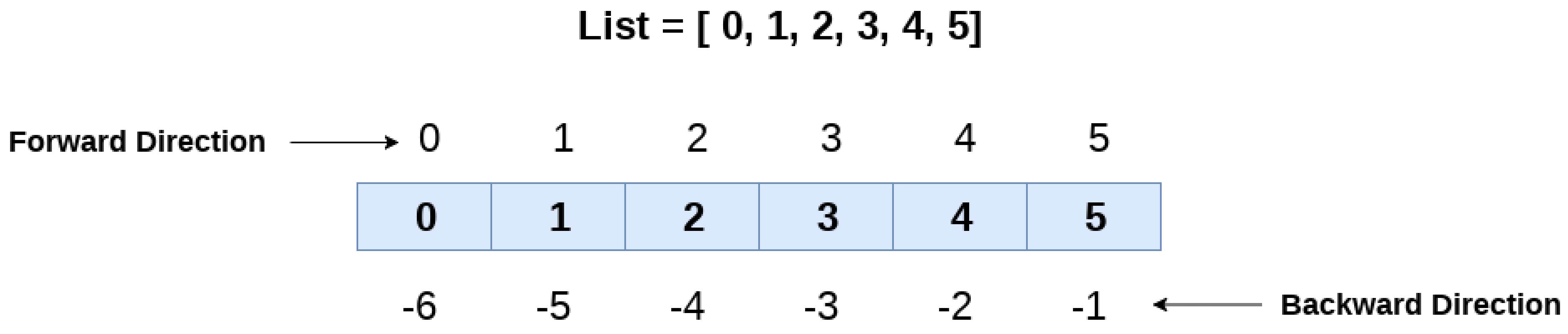


List = [0, 1, 2, 3, 4, 5]

0	1	2	3	4	5

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

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

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

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

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

List[5] = 5
```

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



UPDATING LIST VALUES

Lists are the most versatile data structures in python since they are immutable and their values can be updated by using the slice and assignment operator.

Python also provide us the append() method which can be used to add values to the string.

Consider the following example to update the values inside the list.

```
List = [1, 2, 3, 4, 5, 6]
print(List)
List[2] = 10;
print(List)
List[1:3] = [89, 78]
print(List)
```

The list elements can also be deleted by using the **del** keyword. 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 = [0,1,2,3,4]
print(List)
del List[0]
print(List)
del List[3]
print(List)
```

PYTHON LIST OPERATIONS

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

Lets see how the list responds to various operators.

Consider a List l1 = [1, 2, 3, 4], **and** l2 = [5, 6, 7, 8]

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 can be iterated as follows.

```
List = ["John", "David", "James", "Jonathan"]

for i in List:

    print(i);
```

ADDING ELEMENTS TO THE LIST

Python provides append() function by using which we can add an element to the list. However, the append() method can only add the 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 =[];
n = int(input("Enter the number of elements in the list"));
for i in range(0,n):
    l.append(input("Enter the item?"));
print("printing the list items....");
for i in l:
    print(i, end = "  ");
```

REMOVING ELEMENTS FROM THE LIST

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

PYTHON LIST BUILT-IN FUNCTIONS

1	cmp(list1, list2)	It compares the elements of both the lists.
2	len(list)	It is used to calculate the length of the list.
3	max(list)	It returns the maximum element of the list.
4	min(list)	It returns the minimum element of the list.
5	list(seq)	It converts any sequence to the list.

PYTHON LIST BUILT-IN METHODS

1	list.append(obj)	The element represented by the object obj is added to the list.
2	list.clear()	It removes all the elements from the list.
3	List.copy()	It returns a shallow copy of the list.
4	list.count(obj)	It returns the number of occurrences of the specified object in the list.
5	list.extend(seq)	The sequence represented by the object seq is extended to the list.
6	list.index(obj)	It returns the lowest index in the list that object appears.

7	list.insert(index, obj)	The object is inserted into the list at the specified index.
8	list.pop(obj=list[-1])	It removes and returns the last object of the list.
9	list.remove(obj)	It removes the specified object from the list.
10	list.reverse()	It reverses the list.
11	list.sort([func])	It sorts the list by using the specified compare function if given.

PYTHON TUPLE

Python Tuple is used to store the sequence of immutable python objects. 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 can not be changed.

A tuple can be written as the collection of comma-separated values enclosed with the small brackets. A tuple can be defined as follows.

```
T1 = (101, "Ayush", 22)
T2 = ("Apple", "Banana", "Orange")
```

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

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

However, if we try to reassign the items of a tuple, we would get an error as the tuple object doesn't support the item assignment.

An empty tuple can be written as follows.

An empty tuple can be written as follows.

T3 = ()

The tuple having a single value must include a comma as given below.

T4 = (90,)

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.

TUPLE INDEXING AND SPLITTING

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

The items in the tuple can be accessed by using the slice operator. Python also allows us to use the colon operator to access multiple items in the tuple.

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

Tuple = (0, 1, 2, 3, 4, 5)

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

Tuple[0] = 0	Tuple[0:] = (0, 1, 2, 3, 4, 5)
Tuple[1] = 1	Tuple[:] = (0, 1, 2, 3, 4, 5)
Tuple[2] = 2	Tuple[2:4] = (2, 3)
Tuple[3] = 3	Tuple[1:3] = (1, 2)
Tuple[4] = 4	Tuple[:4] = (0, 1, 2, 3)
Tuple[5] = 5	

Unlike lists, the tuple items can not be deleted by using the del keyword as tuples are immutable. To delete an entire tuple, we can use the del keyword with the tuple name.

Consider the following example.

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

Like lists, the tuple elements can be accessed in both the directions. The right most element (last) of the tuple can be accessed by using the index -1. The elements from left to right are traversed using the negative indexing.

Consider the following example.

tuple1 = (1, 2, 3, 4, 5)

print(tuple1[-1])

print(tuple1[-4])

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

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 dictionary without keys. Consider the following nested structure which can be used as a dictionary.

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

3. Tuple can be used as the key inside dictionary due to its immutable nature.

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 variable length.	The tuple has the fixed length.
4	The list provides more functionality than 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 can not be changed. It can be used as the key inside the dictionary.

NESTING LIST AND TUPLE

We can store list inside tuple or tuple inside the list up to any number of level.

Lets see an example of how can we store the tuple inside the list.

```
Employees = [(101, "Ayush", 22), (102, "john", 29), (103, "james", 45), (104, "Ben", 34)]
print("----Printing list----");
for i in Employees:
    print(i)
Employees[0] = (110, "David",22)
print();
print("----Printing list after modification----");
for i in Employees:
    print(i)
```

PYTHON SET

A Python set is the collection of the unordered items. Each element in the set must be unique, immutable, and the sets remove the duplicate elements. Sets are mutable which means we can modify it after its creation.

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. 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 items with the curly braces {}. Python also provides the set() method, which can be used to create the set by the passed sequence.

```
Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sun
day"}

print(Days)

print(type(Days))

print("looping through the set elements ... ")

for i in Days:

    print(i)
```

```
Days = set(["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "S
unday"])

print(Days)

print(type(Days))

print("looping through the set elements ... ")

for i in Days:

    print(i)
```

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

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)
```

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.

```
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)
```

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);
```

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)
```

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.

```
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.

```
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.

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.

```
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.

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

Days2 = {"Friday","Saturday","Sunday"}

print(Days1.union(Days2)) #printing the union of the sets
```

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.

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

* intersaction() use to intersect two sets

THE INTERSECTION_UPDATE() METHOD

The **intersection_update()** method removes the items from the original set that are not present in both the sets (all the sets if more than one are specified).

The **intersection_update()** method is different from the intersection() method since it modifies the original set by removing the unwanted items, on the other hand, the intersection() method returns a new set.

```
a = {"Devansh", "bob", "castle"}
b = {"castle", "dude", "emyway"}
c = {"fuson", "gaurav", "castle"}

a.intersection_update(b, c)

print(a)
```

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.

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

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

Days2 = {"Monday", "Tuesday", "Sunday"}

print(Days1.difference(Days2)) # prints the difference of the two sets Days1 and Da
ys2
```

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:

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

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

CONTINUE IN NEXT UNIT .....