There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered, unchangeable, and unindexed. No duplicate members.
- **Dictionary** is a collection which is ordered and changeable. No duplicate members

List

- Used to store **sequences** of various types of data (eg. String, int, float in)
- List with single data is referred as **singleton list**
- List is mutable, means we can modify its element after creation
- Defined as collection of values / items of different type
- Values are separated with the comma
- Enclosed with [ ] square brackets

```
List1=[“Ravi”, 22, “India”]
List2=[1,2,3,4,5,6]

print(List1)
print(List2)

Output:
['Ravi', 22, 'India']
[1, 2, 3, 4, 5, 6]
```

Type function

- type() will give the data type.

```
List1=["Ravi", 22, "India"]
List2=[1,2,3,4,5,6]
print(type(List1))
print(type(List2))

Output:
<class 'list'>
<class 'list'>
```

Characteristics of List

- Lists are Ordered
- Element of list can access by index
- List are Mutable types
- List can store various types of data

Ordered Example

```
a=[1,2,"Ravi", 4.50, "Amit", 5,6]
b=[1,2,5,"Ravi", 4.50, "Amit", 6]
print(a==b)
```

Output

```
False
```

- Both list is having **same element** but **different idex** position
- List maintains order of the element. so, called as **ordered collection of object**

Practical

3.1 WAP to create two list and display the output (print) with combination of both the list

List indexing & splitting

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

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

0	1	2	3	4	5

List indexing and Spliting (Sublist)

- The indexing is processed in the same way as string. 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.

list_variable(start:stop:step)

Start = starting index position
Stop = Last index position
Step = skip the nth element within (start:stop)

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

print(List[0:]) #[0, 1, 2, 3, 4, 5]

print(List[:]) #[0, 1, 2, 3, 4, 5]

print(List[:0]) #[]

print(List[2:4]) #[2, 3]

print(List[1:3]) #[1, 2]

print(List[:3]) #[0, 1, 2]

print(List[3:]) #[3, 4, 5]

a = ["a", "b", "c", "d", "e", "f", "g", "h"]

x = slice(3, 5)

print(a[x])   # ['d', 'e']

print(a[3:5]) # ['d', 'e']
```

Sublist - Negative indexing

- Counted from right to left
- Right most element indexed as -1 until left most element
-

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

0 Element of the List	1 Element of the List	2 Element of the List	3 Element of the List	4 Element of the List	5 Element of the List

Forward Direction index **position**

0 index position	1 index position	2 index position	3 index position	4 index position	5 index position

–

Backward Direction **position**

-6 index position	-5 index position	-4 index position	-3 index position	-2 index position	-1 index position

Example

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

print(List[-1]) #5

print(List[-3:]) #[3, 4, 5]

print(List[:-1]) #[0, 1, 2, 3, 4]

print(List[-3:-1])#[3, 4]
```

Updating List Values

append () & insert() methods used to add values to the list

Replace word not add -> it should be update
In book no example for append(), insert() & remove()

append() adds to the end.
insert() adds at a specific position. It has 2 argument (position, “value”)
remove() deletes a specific item.

```
#Creating a List
Desktop=["I5-CPU","LCD-Screen","keyboard","mouse","2-power-cable","1-VGA Cable"]

#print the Desktop list & Laptop list
print("Desktop list is",Desktop)

#print first & last items of Desktop and Laptop
print("First item in Desktop is: ",Desktop[0])
print("Last item of Desktop is: ",Desktop[-1])

#Now you realise you need to have (add) speakers for Desktop & headphone for Laptop

# Adding a new items in list
Desktop.append("Speaker")

print("Added new item in Computer list is",Desktop)

#Now you are replaceing wired keyboard mouse with wireless keyboard & mouse in desktop
Desktop[2]="Wireless-Keyboard"
Desktop[3]="Wireless-Mouse"
#print updated Desktop List
print("Updated Desktop list is",Desktop)

#Now I want to buy mouse-pad
#make sure you add mouse pad after the mouse
Desktop.insert(4,"Mouse-pad")
print("Updated Desktop list is",Desktop)

#Now I want to remove I5-CPU
Desktop.remove("I5-CPU")
print("Updated Desktop list is",Desktop)

#Now I want to Delete first element of the Desktop
del Desktop[0]
print("Updated Desktop list is",Desktop)

#Now i want to delete first two element of Desktop
del Desktop[1:3]
print("Updated Desktop list is",Desktop)
```

Differences between `append() & insert()`

1. **Position of insertion:**
   - `append()` adds the element at the **end** of the list.
   - `insert()` allows you to add the element at any **specified position** in the list.
2. **Arguments:**
   - `append()` takes only **one argument** (the element to be added).
   - `insert()` takes **two arguments** (the index and the element).

Key Differences: between `remove()` & del

1. **Element vs. Index:**
   - `remove()` is used to remove a **specific value** from the list (the first occurrence).
   - `del` is used to remove an element or elements by their **index** or a range of indices.
2. **Return Value:**
   - `remove()` returns nothing but modifies the list in place by removing the element.
   - `del` is a statement, so it also does not return anything and directly deletes elements based on index.
3. **Handling Non-Existing Elements:**
   - `remove()` raises an error if the element to be removed is not found.
   - `del` raises an `IndexError` if the specified index is out of range.

Remove Specified Index using POP()

The `pop()` method removes the specified index.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

Output:

```
['apple', 'cherry']
```

If you do not specify the index, the `pop()` method removes the last item.

```
thislist = ["apple", "banana", "cherry"]
```

```
thislist.pop()
print(thislist)
```

Output:-

```
['apple', 'banana']
```

del()

The `del` keyword can also delete the list completely.

```
thislist = ["apple", "banana", "cherry"]
print(thislist)
#del thislist
#print(thislist) #this will cause an error because you have succsesfully deleted "thislist".
```

Output:-

```
Traceback (most recent call last):
  File "./prog.py", line 4, in <module>
NameError: name 'thislist' is not defined
```

Clear the List

The `clear()` method empties the list.

The list still remains, but it has no content.

```
thislist = ["apple", "banana", "cherry"]
print("Length of List is",len(thislist))
thislist.clear()
print(thislist)
print("Length of List is",len(thislist))
```

Output:-

```
Length of List is 3
[ ]
Length of List is 0
```

Python List Operation

The Concatenation and the Repetition operators works in the same way as String.
Consider a List1=[1,2,3,4] and List2=[5,6,7,8]

Operator	Description
Repetition	- enables the list elements to be repeated multiple times List1*2 = [1,2,3,4,1,2,3,4]
Concatenation	- It concatenates the list List1+List2 = [1,2,3,4,5,6,7,8]
Membership	- it returns true if particular item exists in a particular list, if not in the list it returns false print (2 in List1) Output:- True
Iteration	- for loop is used to iterate over the list element For i in List1: print(i) Output:- 1234
Length	- get length of list len(List1)=4

```
#Python List Operator
List0=[3]
List1=[1,2,3,4]
List2=[5,6,7,8]
List3=["a","b","c","d"]

print("List1 is",List1)

print("List2 is",List2)

#Repeat
print("Repeat",List1*2)
#Repeat [1, 2, 3, 4, 1, 2, 3, 4]

#Concatenation
print("Concatenation",List1+List2)
```

```
#Concatenation [1, 2, 3, 4, 5, 6, 7, 8]

#Membership

print("Membership",2 in List1)
#Membership True

print("Membership",List0 in List1)
#Membership False

print("Membership","a" in List3)
#Membership True

#Length
print("Length of List1 is",len(List1))
#Length of List1 is 4

#Iteration
for item in List1:
        print(item)
#1
#2
#3
#4
```

Adding - Removing elements from the list using loop

- we will check how to remove multiple elements from the list

```
#adding - Removing elements in List

#declaring an empty List
My_List=[]

#take no. of elements from user

no_of_elements=int(input("How many elements, would you like to add:"))

#for loop to add element in the list
for i in range(0,no_of_elements):
    My_List.append(input("Enter Element:"))

print("Printing the List")

for i in My_List:
        print(i,end=" ")
#Removing elements from the list

remove_element=int(input("\nHow many element would you like to remove:"))

for i in range(0,remove_element):
        element_to_remove = input("Enter the element you want to remove:")
        if element_to_remove in My_List:
        My_List.remove(element_to_remove)
        else:
        print(f"{element_to_remove} not found in the list.")

print("\nUpdated List",My_List)
```

#Output-1

```
How many elements, would you like to add:5
Enter Element:1
Enter Element:2
Enter Element:3
Enter Element:4
Enter Element:5
Printing the List
1 2 3 4 5
How many element would you like to remove:2
Enter the element you want to remove:1
Enter the element you want to remove:2

Updated List ['3', '4', '5']
```

Output-2

```
How many elements, would you like to add:5
Enter Element:a
Enter Element:b
Enter Element:c
Enter Element:d
Enter Element:e
Printing the List
a b c d e
How many element would you like to remove:3
Enter the element you want to remove:a
Enter the element you want to remove:b
Enter the element you want to remove:d
```

```
Updated List ['c', 'e']
```

Python List Built-in Function

Function	Description	Example
cmp(list1,list2)	It compares the elements of both the list	This method is not used in the Python 3 and the above version
len(list)	It is used to calculate the length of the list.	x=[10,20,30,40,50,60,70] print(len(x)) #Output:7
max(list)	It returns the maximum element of the list	x=[10,20,30,40,50,60,70] print(max(x)) #Output:70
min(list)	It returns the maximum element of the list	x=[10,20,30,40,50,60,70] print(min(x)) #Output: 10

list(seq)	It converts any sequence to the list.	str="abcd"
		print(type(str))# <class 'str'> s=list(str)
		print(type(s))# <class 'list'>

```
x=[10,20,30,40,50,60,70]
y=[20,30,40,50]

print(max(x),max(y))
print(min(x),min(y))

s="SLTIET"
z=list(s)
print(z)

print(type(z))
```

Output:-

```
70 50
10 20
['S', 'L', 'T', 'I', 'E', 'T']
<class 'list'>
```

Extend List

To append elements from *another list* to the current list, use the `extend()` method.

```
List1 = ["apple", "banana", "cherry"]
List2 = ["mango", "pineapple", "papaya"]
List1.extend(List2)
print(List1)
```

Output

```
['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']
```

Sorting List

List objects have a `sort()` method that will sort the list alphanumerically, ascending, by default:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]

thislist.sort()

print(thislist)
```

Output

```
['banana', 'kiwi', 'mango', 'orange', 'pineapple']
```

Sort the list numerically:

```
thislist = [100, 50, 65, 82, 23]

thislist.sort()

print(thislist)
```

Output

```
[23, 50, 65, 82, 100]
```

Sort Descending

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]

thislist.sort(reverse = True)
```

```
print(thislist)
```

Output

```
['pineapple', 'orange', 'mango', 'kiwi', 'banana']
```

```
thislist = [100, 50, 65, 82, 23]

thislist.sort(reverse = True)

print(thislist)
```

Output

```
[100, 82, 65, 50, 23]
```

Case Insensitive Sort

By default the `sort()` method is case sensitive, resulting in all capital letters being sorted before lower case letters:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
```

Output

```
['Kiwi', 'Orange', 'banana', 'cherry']
```

we can use built-in functions as key functions when sorting a list.

So if you want a case-insensitive sort function, use str.lower as a key function:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]

thislist.sort(key = str.lower)

print(thislist)
```

Output

```
['banana', 'cherry', 'Kiwi', 'Orange']
```

Reverse Order

The `reverse()` method reverses the current sorting order of the elements regardless of the alphabet

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

Output

```
['cherry', 'Kiwi', 'Orange', 'banana']
```

Copy a List

You cannot copy a list simply by typing `list2 = list1`, because: `list2` will only be a *reference* to `list1`, and changes made in `list1` will automatically also be made in `list2`.

Built-in List method `copy()` to copy a list.

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Output;-

```
['apple', 'banana', 'cherry']
```

Another way to make a copy is to use the built-in method `list()`.

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Output;-

```
['apple', 'banana', 'cherry']
```

a copy of a list by using the `:` (slice) operator.

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist[:]
print(mylist)
```

Output;-

```
['apple', 'banana', 'cherry']
```

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.
One of the easiest ways are by using the + operator.

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]

list3 = list1 + list2
print(list3)
```

Output;-

```
['a', 'b', 'c', 1, 2, 3]
```

Another way to join two lists is by appending all the items from list2 into list1, one by one:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

Output;-

```
['a', 'b', 'c', 1, 2, 3]
```

Use the `extend()` method to add list2 at the end of list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

list1.extend(list2)
print(list1)
```

Output;-

```
['a', 'b', 'c', 1, 2, 3]
```

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Tuple

- Built-in data structure in python
- Ordered collection of objects
- Unlike list, tuple comes with the limited functionality
- Primary difference between List & Tuple is Mutability.
- Lists are mutable where as Tuple are immutable.
- It means tuple can not modified, added or deleted once it created
- List are defined by the using parentheses [ ] (square bracket) to enclose the elements, separated by comma
- Where as use of parentheses ( ) (round bracket) in tuples is optional.
- But it is recommended to use round bracket to distinguish between start point and end point of tuple

```
touple_A=(item_1,item_2,item_3,...,item_n)
```

Example

```
#touple
T1=(7,"Ravi",1.5)
T2=("Apple","Orange","Banana")
T3=10,20,30,40,50
T4=() # Creating empty tuple
T5=("Amit")#single element in tuple  will be considered as String
T6=("Amit",)# for single element in tuple, we need to write comma

print("T1",type(T1))
print("T2",type(T2))
print("T3",type(T3))
print("T4",type(T4))
print("T5",type(T5))
print("T6",type(T6))

# indxing in tuple
print(T1[0])
print(T1[1])
print(T1[2])
print()

#Negative indexing
print(T1[-1])
print(T1[-2])
print(T1[-3])
```

```
#delete speccificc element of tuple
#del T1[0] #Error - TypeError: 'tuple' object doesn't support item deletion
print("elements of tuple T1 are:",T1)


#delete complete tuple
#del T1
#print(T1)
```

Output:-

```
T1 <class 'tuple'>
T2 <class 'tuple'>
T3 <class 'tuple'>
T4 <class 'tuple'>
T5 <class 'str'>
T6 <class 'tuple'>
7
Ravi
1.5

1.5
Ravi
7
elements of tuple T1 are: (7, 'Ravi', 1.5)
```

Indexing & slicing, Negative indexing

Same as List

See example

Basic Tuple Operators

Same as List

See example

Where to use tuple

- When I have lear idea about my data, and my data is constant (not change)
- Tuple simulate as dictionary without key

Eg.

[(1,"Ravi",90),(2,"Amit",83)),(1,"Ajay",75))]

List vs Tuple

List	Tuple
The literal syntax of list is shown by the [ ]	The literal syntax of the tuple is shown by the ()
The List is mutable	The tuple is immutable.
The List has the a variable length	The tuple has the fixed length.
The list provides more functionality than a tuple.	The tuple provides less functionality than the list.
The list is used where the value of the items can be changed.	The tuple is used 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.
The lists are less memory efficient than a tuple.	The tuples are more memory efficient because of its immutability.

Examples of w3 school

Set

- A set is an unordered collection of items. Every set element is unique (no duplicates) and must be immutable (cannot be changed).
- 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.
- This is based on a data structure known as a hash table.
- We cannot access or change an element of a set using indexing or slicing. Set data type does not support it.

Characteristics of a Set

- Sets are unordered.
- Set element is unique. Duplicate elements are not allowed.
- A set itself may be modified, but the elements contained in the in the set must be of an immutable type.
- Mathematically a set is a collection of items not in any particular order.
- A python set is similar to this mathematical definition with below additional conditions.
    - The elements in the set cannot be duplicate.
    - The element in the set are immutable (can not modified) but the set as a whole is mutable.
    - There is not index attached to any element in a python set. So they do not support any indexing or slicing operation.
- The set in python are used for mathematical operations like union, intersection, difference and complement etc.
- We can create a set, access it's element and carry out these mathematical operations.

* **Note:** Set *items* are unchangeable, but you can remove items and add new items.

Creating a Set

- Set is created by using the set() function or placing all the elements within a pair of curly bracket { }
- The `set()` function takes an iterable, such as a list or a tuple,

```
#Set
Days=set(["Mon","Tue","Wed","Thu", "Fri","Sat","Sun"])
#set using list
```

```
Days2=set(("Mon","Tue","Wed","Thu", "Fri","Sat","Sun"))
#set using tuple

Months={"jan","Feb","Mar"}

Dates={24,21,22,23}

print(Days)
print(Days2)
print(Months)
print(Dates)
```

Output -1

```
{'Sat', 'Tue', 'Fri', 'Wed', 'Sun', 'Mon', 'Thu'}
{'Sat', 'Tue', 'Fri', 'Wed', 'Sun', 'Mon', 'Thu'}
{'jan', 'Feb', 'Mar'}
{24, 21, 22, 23}
```

Output -2

```
{'Fri', 'Tue', 'Sun', 'Wed', 'Thu', 'Mon', 'Sat'}
{'Fri', 'Tue', 'Sun', 'Wed', 'Thu', 'Mon', 'Sat'}
{'jan', 'Feb', 'Mar'}
{24, 21, 22, 23}
```

See the output of above program, it gives different ordered every time we print the set. It is due to following reason:

Hashing: Python uses a **hash table** to implement sets. Each element in the set is assigned a hash value, and the set stores elements based on these hash values, not the insertion order. The process of hashing ensures efficient lookup, insertion, and deletion of elements, but it does not preserve any order.

Unordered: Sets are designed to prioritize fast membership checking and other operations, rather than maintaining the order of elements. Hence, whenever you print or iterate over a set, the order of elements can appear random or different from how they were inserted.

Accessing values in a Set

We can't access individual values/ elements in a set.
We can only access all elements together.
But we can get a list of individual elements by looping through the set.

```
Days=set(["Mon","Tue","Wed","Thu", "Fri","Sat","Sun"])

for d in Days:
        print(d)
```

Output:-

```
Thu
Wed
Mon
Sun
Tue
Fri
Sat
```

Adding items to a set

- We can add elements to a set by using add() method.
- There is no specific index attached to the newly added element.

```
#Adding element in a Set
Days=set(["Mon","Tue","Wed","Thu", "Fri","Sat"])

Days.add("Sun")
print(Days)
```

Output:-

```
{'Mon', 'Tue', 'Sat', 'Thu', 'Fri', 'Sun', 'Wed'}
```

Removing items from a Set

- We can remove elements from a set by using discard() method.

```
#Removing element from a Set
Days=set(["Mon","Tue","Wed","Thu", "Fri","Sat","Sun"])

Days.discard("Sun")
print(Days)
```

Output:-

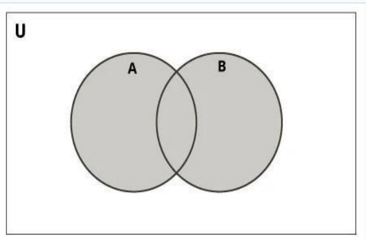
```
{'Thu', 'Sat', 'Tue', 'Mon', 'Fri', 'Wed'}
```

Set Operation

Set can be performed mathematical operation such as union, insertion, difference and symmetric difference.

Union of two sets

- The union of two set is calculated by using the pipe (|) operator.
- The union of two set containing all the elements that are present in both the set,
- Python also provides **union()** method which can also be used to calculate the union of two sets.



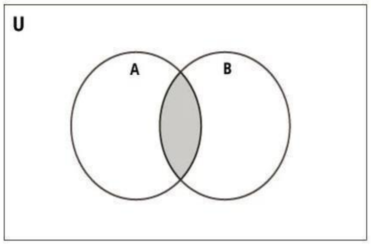
```
#Set operation
#Union
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = {8,9,10}

print("My set A, B, C are as below:")
print("Set A:",A)
print("Set B:",B)
print("Set C:",C)

#using pipe | method
print("Union pipe | method",A | B)

#using union() method
print("Union function",A.union(C))
```

Output:-

```
My set A, B, C are as below:
Set A: {1, 2, 3, 4, 5}
Set B: {4, 5, 6, 7, 8}
Set C: {8, 9, 10}
Union pipe | method {1, 2, 3, 4, 5, 6, 7, 8}
Union function {1, 2, 3, 4, 5, 8, 9, 10}
```

Intersection of two sets

- Intersection of A and B is a set of elements that are common in both the sets.
- Intersection is performed using & operator. Same can be accomplished using the intersection() method.



```
#Set operation
#intersection
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = {8,9,10}

print("My set A, B, C are as below:")
print("Set A:",A)
print("Set B:",B)
print("Set C:",C)

#using &
print("Using &",A & B)

#using intersection() method
print("using intersection() method",B.intersection(C))
```

Output:-

```
My set A, B, C are as below:
Set A: {1, 2, 3, 4, 5}
Set B: {4, 5, 6, 7, 8}
Set C: {8, 9, 10}
Using & {4, 5}
using intersection() method {8}
```

Intersection_update method

- It removes items from the item from the original set that are not present in both the set. (All the seta if more than one are specified.)
- This method is different from the intersection() method since it modifies the original set by removing the unwanted items, on other hand, the intersection() method returns a new set.

```
#Set operation
#intersection_update
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = {4,8,9,10}

print("My set A, B, C are as below:")
print("Set A:",A)
print("Set B:",B)
print("Set C:",C)

A.intersection_update(B)
print("A intersection_update with B")
print(A)

B.intersection_update(C)
print("B intersection_update with C")
print(B)

#re-declare
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = {4,8,9,10}

print("A intersection_update with B and C")
A.intersection_update(B,C)
print(A)
```

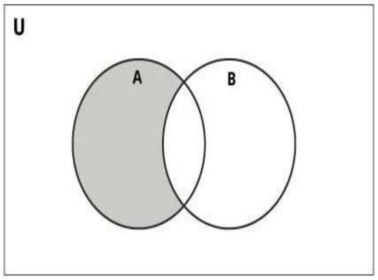
Output:-

```
My set A, B, C are as below:
Set A: {1, 2, 3, 4, 5}
Set B: {4, 5, 6, 7, 8}
Set C: {8, 9, 10, 4}
A intersection_update with B
{4, 5}
B intersection_update with C
{8, 4}
A intersection_update with B and C
{4}
```

-

Difference between two set

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



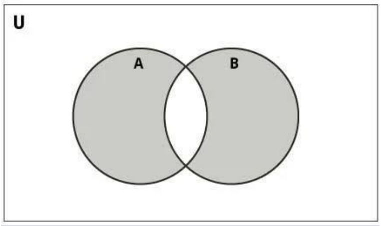
```
#Set operation
#difference
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = {8,9,10}

print("My set A, B, C are as below:")
print("Set A:",A)
print("Set B:",B)
print("Set C:",C)

#using -
print("Using A - B",A - B)

#using difference() method
print("using difference() method B - C",B.difference(C))
```

Output:-

```
My set A, B, C are as below:
Set A: {1, 2, 3, 4, 5}
Set B: {4, 5, 6, 7, 8}
Set C: {8, 9, 10}
Using A - B {1, 2, 3}
using difference() method B - C {4, 5, 6, 7}
```

Symmetric difference if two set

- It is calculated by ^ operator or symmetric_difference() method.
- It removes the elements which are present in both the sets.



```
#Set operation
#Symmetric difference
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
C = {8,9,10}

print("My set A, B, C are as below:")
print("Set A:",A)
print("Set B:",B)
print("Set C:",C)

#using ^
print("Using A ^ B",A ^ B)

#using difference() method
print("using symmetric_difference() method B ^ C",B.symmetric_difference(C))
```

Output:-

```
My set A, B, C are as below:
Set A: {1, 2, 3, 4, 5}
Set B: {4, 5, 6, 7, 8}
Set C: {8, 9, 10}
Using A ^ B {1, 2, 3, 6, 7, 8}
using symmetric_difference() method B ^ C {4, 5, 6, 7, 9, 10}
```

Set Comparisons

- Python allows to use the comparison operator like <, >, <=, >=, == 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.

```
#Set operation
#Set comparision
A = {1, 2, 3, 4, 5}
B = {4, 5}
C = {8,9,10}

print("My set A, B, C are as below:")
print("Set A:",A)
print("Set B:",B)
print("Set C:",C)

print("A > B",A > B)
print("A < B",A < B)

print("B == C", B == C)
```

Output:-

```
My set A, B, C are as below:
Set A: {1, 2, 3, 4, 5}
Set B: {4, 5}
Set C: {8, 9, 10}
A > B True
A < B False
B == C False
```

Iterating Through a Set: We can iterate through each item in a set using a for loop. Example:

```
for letter in set("apple"):
        print(letter)
```

Output:-

```
a
p
l
e
```

Dictionary