

# Python ka Chilla Witha Dr.Ammar

Musharaf Ahsan

## Indexing and Data Structure

### - Indexing

*Make a string*

```
In [1]: a="Ladu Barfi"  
a
```

```
Out[1]: 'Ladu Barfi'
```

Indices	0	1	2	3	4	5	6	7	8	9
String	L	a	d	u		B	a	r	f	i

```
In [2]: a[0]
```

```
Out[2]: 'L'
```

```
In [3]: a[6]
```

```
Out[3]: 'a'
```

```
In [4]: a[9]
```

```
Out[4]: 'i'
```

*Length of indices*

```
In [5]: len(a)
```

```
Out[5]: 10
```

```
In [6]: a[9]
```

```
Out[6]: 'i'
```

```
In [7]: a[0:3] #It mean 0 to 3
```

```
Out[7]: 'Lad'
```

```
In [8]: a[0:4]
```

```
Out[8]: 'Ladu'
```

The last index is exclusive means if we want to print `***"Ladu"` **having 4 chracteres with indeces from 0 to 3 then we have to write in code 0 to 4(0:4) because the last index is not going to print. As we see above the output of `a[0:3]**` is "Lad" not "Ladu".**

## Negative Indexing

**Negative indexing start from the end of string as shown in the image below.**

String	L	a	d	u		B	a	r	f	i
Negative Indexing	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
In [9]: a[-1]
```

```
Out[9]: 'i'
```

```
In [10]: a[-7]
```

```
Out[10]: 'u'
```

```
In [11]: a[-10]
```

```
Out[11]: 'L'
```

**Now if we want to print Barfi by using negative indexing then we we'll do that as follow.**

```
In [12]: a[-6:-1] #last index is exlusive
```

```
Out[12]: ' Barf'
```

```
In [13]: a[-5:10]
```

```
Out[13]: 'Barfi'
```

## -String Methods

```
In [14]: food="biryani"
         food
```

```
Out[14]: 'biryani'
```

### Capitalize every Element

```
In [15]: food.capitalize()
```

```
Out[15]: 'Biryani'
```

### Uppercase Letters

```
In [16]: food.upper()
```

```
Out[16]: 'BIRYANI'
```

### Lowercase Letters

```
In [17]: food.lower()
```

```
Out[17]: 'biryani'
```

### Replacement of Letters in a string

```
In [18]: food.replace('b', 'Sh')
```

```
Out[18]: 'Shiryani'
```

### Count a Spacific letter in a String

```
In [19]: name="Muhammad Musharaf Ahsan"  
name
```

```
Out[19]: 'Muhammad Musharaf Ahsan'
```

```
In [20]: name.count("M")
```

```
Out[20]: 2
```

### Finding an index number of sprific letter in a string

```
In [21]: name.find("u")
```

```
Out[21]: 1
```

```
In [22]: name.find("mm")
```

```
Out[22]: 4
```

```
In [ ]:
```

```
In [143]: name.find("m") #if there are more combinations of required alphabet then  
#it shows the index which come first in sequence
```

Out[143]: 4

```
In [24]: name.find("Ahsan")
```

Out[24]: 18

## How to Split a String

```
In [25]: khana = "I love smosa pakora raita salad and qorma"  
khana
```

Out[25]: 'I love smosa pakora raita salad and qorma'

```
In [26]: khana.split(" ") #Split string on the basis of space (" ").
```

Out[26]: ['I', 'love', 'smosa', 'pakora', 'raita', 'salad', 'and', 'qorma']

```
In [27]: khana2="I love smosa, pakora, raita, salad and qorma"  
khana2
```

Out[27]: 'I love smosa, pakora, raita, salad and qorma'

```
In [28]: khana2.split(",") #Split string on the basis of comma (",").
```

Out[28]: ['I love smosa', ' pakora', ' raita', ' salad and qorma']

# - Basic Data Structure in Python

## 1-Tuple

## 2-List

## 3-Dictionaries

## 4-Set

## 1-Tuple

- Ordered collection of Elements
- Enclosed in Parntesis()
- Diffrent kind of elements can be stored

- Once elements are stored you can not change them(immutable)

```
In [29]: tup1=(90, "python", True, 3.5)
          tup1
```

```
Out[29]: (90, 'python', True, 3.5)
```

```
In [30]: #Data type of a TUPLE
          type(tup1)
```

```
Out[30]: tuple
```

---

## -Indexing in Tuple

```
In [31]: tup1[0]
```

```
Out[31]: 90
```

```
In [32]: tup1[1]
```

```
Out[32]: 'python'
```

```
In [33]: # We know that the last element is exclusive
          tup1[0:3]
```

```
Out[33]: (90, 'python', True)
```

```
In [34]: #Length of a tuple or count total elements is a tuple
          len(tup1)
```

```
Out[34]: 4
```

```
In [35]: tup2=(4, "Ahsan", 9.4, False)
          tup2
```

```
Out[35]: (4, 'Ahsan', 9.4, False)
```

```
In [36]: #Concatenate (to add two tuple or >2)
          tup1+tup2
```

```
Out[36]: (90, 'python', True, 3.5, 4, 'Ahsan', 9.4, False)
```

```
In [37]: #Concatenate + repeat
          tup1*2+tup2
```

Out[37]: (90, 'python', True, 3.5, 90, 'python', True, 3.5, 4, 'Ahsan', 9.4, False)

```
In [38]: tup3 =(32,42,44,67,78,88,90,10)
tup3
```

Out[38]: (32, 42, 44, 67, 78, 88, 90, 10)

```
In [39]: min(tup3)
```

Out[39]: 10

```
In [40]: max(tup3)
```

Out[40]: 90

```
In [41]: tup3*2 # with this method we can repeat the values of tuple not multiply like 2*2=4.
```

Out[41]: (32, 42, 44, 67, 78, 88, 90, 10, 32, 42, 44, 67, 78, 88, 90, 10)

## Other Functions Tuple

### 1- count

```
In [42]: tup3.count(44)
```

Out[42]: 1

### 2- index

```
In [43]: tup3.index(78)
```

Out[43]: 4

---

## 2-Lists

- Ordered collection of elements
- Enclosed in [ ] squar brackets
- Mutable (you can change the values)

```
In [44]: list1=[4, "Motu Patlu", 8.4, True]
list1
```

Out[44]: [4, 'Motu Patlu', 8.4, True]

```
In [45]: type(list1)
```

```
Out[45]: list
```

```
In [46]: len(list1)
```

```
Out[46]: 4
```

```
In [47]: list1[3]
```

```
Out[47]: True
```

```
In [48]: list1*2
```

```
Out[48]: [4, 'Motu Patlu', 8.4, True, 4, 'Motu Patlu', 8.4, True]
```

```
In [49]: list2=[65, "I love Pakistan", "Musharaf Ahsan", 887, 9.5, False]
list2
```

```
Out[49]: [65, 'I love Pakistan', 'Musharaf Ahsan', 887, 9.5, False]
```

```
In [50]: list1+list2
```

```
Out[50]: [4,
'Motu Patlu',
8.4,
True,
65,
'I love Pakistan',
'Musharaf Ahsan',
887,
9.5,
False]
```

## Other Functions of Lists

```
In [51]: list1
```

```
Out[51]: [4, 'Motu Patlu', 8.4, True]
```

### - Append()

*Used to insert or add new elements in the list.*

```
In [52]: list1.append("CUVAS")
list1
```

```
Out[52]: [4, 'Motu Patlu', 8.4, True, 'CUVAS']
```

## - Clear()

*This function delete every item from the list.*

```
In [53]: list1.clear()  
list1
```

```
Out[53]: []
```

```
In [54]: list1      # Here we saw that the list is empty.
```

```
Out[54]: []
```

## - Copy()

```
In [55]: list1.copy() ##is ki samjh nhi ayeee
```

```
Out[55]: []
```

## - Count()

*Python count() function is an inbuilt function of python which is used to count the number of occurrences of an item in an array/list and the occurrences of a character or substring in the string. Count function is case-sensitive it means that 'a' & 'A' are not treated as the same both are different.*

```
In [56]: list3=[1,0,0,"APE","CUVAS",2,2,0,1,"BWP"]
```

```
In [57]: list3
```

```
Out[57]: [1, 0, 0, 'APE', 'CUVAS', 2, 2, 0, 1, 'BWP']
```

```
In [58]: list3.count(0)
```

```
Out[58]: 3
```

## - Extend()

```
In [59]: list3.extend("Ahsan")  
list3
```

```
Out[59]: [1, 0, 0, 'APE', 'CUVAS', 2, 2, 0, 1, 'BWP', 'A', 'h', 's', 'a', 'n']
```

## - Index()

*index() is an inbuilt function in Python, which searches for a given element from the start of the list and returns the lowest index where the element appears.*



```
In [60]: list3.index('APE')
```

```
Out[60]: 3
```

## - Insert()

The Python List `insert()` method is an inbuilt function in Python that inserts a given element at a given index in a list.

### Syntax:

`list_name.insert(index, element)`

```
In [61]: list3.insert(2,"Ali") #Here we insert "Ali" at index 2.
list3
```

```
Out[61]: [1, 0, 'Ali', 0, 'APE', 'CUVAS', 2, 2, 0, 1, 'BWP', 'A', 'h', 's', 'a', 'n']
```

## - pop()

Python list `pop()` is an inbuilt function in Python that removes and returns the last value from the List or the given index value.

### Syntax:

`list_name.pop(index)`

### Parameter:

**index (optional)** – The value at index is popped out and removed. If the index is not given, then the last element is popped out and removed.

**Exception:** When the index is out of range, it returns `IndexError`.

```
In [62]: list3.pop(4)
list3
```

```
Out[62]: [1, 0, 'Ali', 0, 'CUVAS', 2, 2, 0, 1, 'BWP', 'A', 'h', 's', 'a', 'n']
```

```
In [63]: list3.pop()
```

```
Out[63]: 'n'
```

```
In [64]: list3
```

```
Out[64]: [1, 0, 'Ali', 0, 'CUVAS', 2, 2, 0, 1, 'BWP', 'A', 'h', 's', 'a']
```

## - remove()

*remove()* is an inbuilt function used to remove specific element from the list.

```
In [65]: list3.remove("CUVAS")  
list3
```

```
Out[65]: [1, 0, 'Ali', 0, 2, 2, 0, 1, 'BWP', 'A', 'h', 's', 'a']
```

## - reverse()

*Used to reverse the list.*

```
In [66]: list3.reverse()  
list3
```

```
Out[66]: ['a', 's', 'h', 'A', 'BWP', 1, 0, 2, 2, 0, 'Ali', 0, 1]
```

## - Sort()

*Python list sort() function can be used to sort a List in ascending, descending, or user-defined order.*

```
In [67]: list4=[9,8,0,7,6,8,5,4,3,2,1,0]  
list4
```

```
Out[67]: [9, 8, 0, 7, 6, 8, 5, 4, 3, 2, 1, 0]
```

### Ascending Order

```
In [68]: list4.sort()  
list4
```

```
Out[68]: [0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 8, 9]
```

### Descending Order

```
In [69]: list4.sort(reverse=True)  
list4
```

```
Out[69]: [9, 8, 8, 7, 6, 5, 4, 3, 2, 1, 0, 0]
```

## 3-Dictionaries

- An unorddered collection of elements
- Contain two things Key an Value
- Enclosed in Curly Braces { }
- Mutable (you can change the values)

```
In [70]: # food and thier Prices
         food1={"samosa":30, "Pakora":100, "raita":20, "salad":50, "chiken rolls":20}
         food1
```

```
Out[70]: {'samosa': 30, 'Pakora': 100, 'raita': 20, 'salad': 50, 'chiken rolls': 20}
```

```
In [71]: type(food1)
```

```
Out[71]: dict
```

**food1 = {"samosa" : 30, "Pakora" : 100}**

## Extract Data

### dict\_keys

```
In [72]: keys=food1.keys() # also use as food1.keys
         keys
```

```
Out[72]: dict_keys(['samosa', 'Pakora', 'raita', 'salad', 'chiken rolls'])
```

### dict\_values

```
In [73]: food1.values()
```

```
Out[73]: dict_values([30, 100, 20, 50, 20])
```

## Adding New Elements

```
In [74]: food1["tikki"]=10
         food1
```

```
Out[74]: {'samosa': 30,
         'Pakora': 100,
         'raita': 20,
         'salad': 50,
         'chiken rolls': 20,
         'tikki': 10}
```

## Update Values

```
In [75]: food1["tikki"]=15  
food1
```

```
Out[75]: {'samosa': 30,  
          'Pakora': 100,  
          'raita': 20,  
          'salad': 50,  
          'chiken rolls': 20,  
          'tikki': 15}
```

```
In [76]: food2={"Dates":50, "Chocolate":200, "Sawayan":1000}  
food2
```

```
Out[76]: {'Dates': 50, 'Chocolate': 200, 'Sawayan': 1000}
```

## Concatinate

*The function `**(food1+food2)**` does not work here.*

```
In [77]: food1.update(food2)  
food1
```

```
Out[77]: {'samosa': 30,  
          'Pakora': 100,  
          'raita': 20,  
          'salad': 50,  
          'chiken rolls': 20,  
          'tikki': 15,  
          'Dates': 50,  
          'Chocolate': 200,  
          'Sawayan': 1000}
```

## Other Functions of Dictionaries

```
In [78]: clothes={"pant":1000, "shirt":500, "tie":200, "coat":2000}  
clothes
```

```
Out[78]: {'pant': 1000, 'shirt': 500, 'tie': 200, 'coat': 2000}
```

### 1- clear()

*The `clear()` method removes all the elements from a dictionary*

```
In [79]: clothes.clear()
```

```
In [80]: clothes
```

```
Out[80]: {}
```

### 2- copy()

***The dict.copy() method returns a shallow copy of the dictionary.***

***The dictionary can also be copied using the = operator, which points to the same object as the original. So if any change is made in the copied dictionary will also reflect in the original dictionary.***

```
In [81]: clothes1={"pant":100, "shirt":200, "tie":100, "coat":5000}  
clothes1
```

```
Out[81]: {'pant': 100, 'shirt': 200, 'tie': 100, 'coat': 5000}
```

```
In [82]: # By using copy() method..  
clothes2=clothes1.copy()  
clothes2
```

```
Out[82]: {'pant': 100, 'shirt': 200, 'tie': 100, 'coat': 5000}
```

```
In [83]: # By using assignment "=" operator..  
clothes3=clothes1  
clothes3
```

```
Out[83]: {'pant': 100, 'shirt': 200, 'tie': 100, 'coat': 5000}
```

### 3- fromkeys()

*The fromkeys() method returns a dictionary with the specified keys and the specified value.*

### Syntax

***dict.fromkeys(keys, value)***

### Parameter Values

Parameter	Description
<i>keys</i>	Required. An iterable specifying the keys of the new dictionary
<i>value</i>	Optional. The value for all keys. Default value is None

```
In [84]: # 1st Method  
clothes1.fromkeys(clothes1,90)
```

```
Out[84]: {'pant': 90, 'shirt': 90, 'tie': 90, 'coat': 90}
```

```
In [85]: # 2nd Method
key = ("oppo", "samsung", "iphone", "vivo")
value = (100000)
cell_price = dict.fromkeys(key, value)
cell_price
```

```
Out[85]: {'oppo': 100000, 'samsung': 100000, 'iphone': 100000, 'vivo': 100000}
```

## 4- get()

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

## Syntax

**`dictionary.get(keyname, value)`**

## Parameter Values

Parameter	Description
<i>keyname</i>	Required. The keyname of the item you want to return the value from
<i>value</i>	Optional. A value to return if the specified key does not exist. Default value None

```
In [86]: clothes1.get("pant")
```

```
Out[86]: 100
```

```
In [87]: food1.get("samosa")
```

```
Out[87]: 30
```

## 5- items()

A `items()` method is used with a dictionary to get the list with all dictionary keys with values.

## Syntax

**`dictionary.items()`**

```
In [88]: food1.items()
```

```
dict_items([('samosa', 30), ('Pakora', 100), ('raita', 20), ('salad', 50), ('chicken roll
```

```
Out[88]: s', 20), ('tikki', 15), ('Dates', 50), ('Choclade', 200), ('Sawayan', 1000)])
```

## 6- keys()

```
In [89]: food1.keys()
```

```
Out[89]: dict_keys(['samosa', 'Pakora', 'raita', 'salad', 'chicken rolls', 'tikki', 'Dates', 'Choc  
late', 'Sawayan'])
```

## 7- pop()

The `pop()` method removes the specified item from the dictionary.

## Syntax

***dictionary.pop(keyname, defaultvalue)***

## Parameter Values

Parameter	Description
<i>keyname</i>	Required. The keyname of the item you want to remove
<i>defaultvalue</i>	Optional. A value to return if the specified key do not exist.  If this parameter is not specified, and the no item with the specified key is found, an error is raised

```
In [90]: food1.pop("samosa")
```

```
Out[90]: 30
```

```
In [91]: food1      # samosa from the dictionary is removed
```

```
Out[91]: {'Pakora': 100,  
          'raita': 20,  
          'salad': 50,  
          'chicken rolls': 20,  
          'tikki': 15,  
          'Dates': 50,  
          'Choclade': 200,  
          'Sawayan': 1000}
```

## 8- popitem()

The `popitem()` method removes the item that was last inserted into the dictionary. In versions before 3.7, the `popitem()` method removes a random item.

## Syntax

### ***dictionary.popitem()***

```
In [92]: food1.popitem()
```

```
Out[92]: ('Sawayan', 1000)
```

## 9-setdefault()

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

## Syntax

### ***dictionary.setdefault(keyname, value)***

Parameter	Description
<i>keyname</i>	Required. The keyname of the item you want to return the value from
<i>value</i>	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 [93]: food1.setdefault("raita")
```

```
Out[93]: 20
```

```
In [94]: food1.setdefault("Dates",50)
```

```
Out[94]: 50
```

## 10- update()

*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
<i>iterable</i>	A dictionary or an iterable object with key value pairs, that will be inserted to the dictionary

```
In [95]: food1.update({"burger":150})
```

```
In [96]: food1
```

```
Out[96]: {'Pakora': 100,
          'raita': 20,
          'salad': 50,
          'chicken rolls': 20,
          'tikki': 15,
          'Dates': 50,
          'Chocolate': 200,
          'burger': 150}
```

## 4-Set

- Unordered and unindexed
- Curly braces are used { }
- No duplicates are allowed

```
In [97]: s1={1, 2.5, 5.6, "Musharaf", "CUVAS", True} # bolian operator add nhi kr sakte wo print s1
```

```
Out[97]: {1, 2.5, 5.6, 'CUVAS', 'Musharaf'}
```

### 1- add()

The `add()` method adds an element to the set.

If the element already exists, the `add()` method does not add the element.

## Syntax

**`set.add(elmnt)`**

```
In [98]: s1.add("Musharaf") # Duplicate items add nhi ho gy.
          s1
```

```
Out[98]: {1, 2.5, 5.6, 'CUVAS', 'Musharaf'}
```

### 2- clear()

The `clear()` method removes all elements in a set.

## Syntax

### ***set.clear()***

```
In [99]: s1.clear()
```

```
In [100... s1
```

```
Out[100... set()
```

```
In [101... s={88, 8.9, 0.45, "Akif", "boss"}
```

## 3- copy()

*The copy() method copies the set.*

## Syntax

### ***set.copy()***

```
In [102... copy_set=s.copy()
```

```
In [103... copy_set
```

```
Out[103... {0.45, 8.9, 88, 'Akif', 'boss'}
```

```
In [104... s2={0.45, 8.9, "Ahsan", "boss", "asfand"}
```

## 4- difference()

*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)***

```
In [105... x=s.difference(s2)
```

```
In [106... x
```

```
Out[106... {88, 'Akif'}
```

```
In [107... y=s2.difference(s)
y
```

```
Out[107... {'Ahsan', 'asfand'}
```

## 5- difference\_update()

The `difference_update()` method removes the items that exist in both sets.

The `difference_update()` method is different from the `difference()` method, because the `difference()` method returns a new set, without the unwanted items, and the `difference_update()` method removes the unwanted items from the original set.

## Syntax

**`set.difference_update(set)`**

```
In [108... a=s.difference_update(s2)
```

```
In [109... s
```

```
Out[109... {88, 'Akif'}
```

```
In [110... b=s2.difference_update(s) #from s={88, "Akif"} & s2={0.45, 8.9, 'Ahsan', 'asfand', 'bos
```

```
In [111... s2
```

```
Out[111... {0.45, 8.9, 'Ahsan', 'asfand', 'boss'}
```

## 6- discard()

The `discard()` method removes the specified item from the set.

This method is different from the `remove()` method, because the `remove()` method will raise an error if the specified item does not exist, and the `discard()` method will not.

## Syntax

**`set.discard(value)`**

```
In [112... s3={"python", "c++", "c", 99, 99.09}
```

```
In [113... s3
```

```
Out[113... {99, 99.09, 'c', 'c++', 'python'}
```

```
In [114... s3.discard("c++")
```

```
In [115... s3
```

```
Out[115... {99, 99.09, 'c', 'python'}
```

## 7- intersection()

*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)***

```
In [116... s4={99.09, 99, "dell", "hp", "Apple", "python"}
```

```
In [117... s5={"dell", "python", 99 }
```

```
In [118... s3.intersection(s4,s5)
```

```
Out[118... {99, 'python'}
```

## 8- intersection\_update()

*The intersection\_update() method removes the items that is not present in both sets (or in all sets if the comparison is done between more than two sets).*

*The intersection\_update() method is different from the intersection() method, because the intersection() method returns a new set, without the unwanted items, and the intersection\_update() method removes the unwanted items from the original set.*

## Syntax

***set.intersection\_update(set1, set2 ... etc)***

```
In [119... s4.intersection_update(s5)
```

```
In [120... s4
```

```
Out[120... {99, 'dell', 'python'}
```

## 9- isdisjoint()

The `isdisjoint()` method returns `True` if none of the items are present in both sets, otherwise it returns `False`.

### Syntax

**`set.isdisjoint(set)`**

```
In [121... s4.isdisjoint(s5)
```

```
Out[121... False
```

## 10- issubset()

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

### Syntax

**`set.issubset(set)`**

```
In [122... s6={"a", "b", "d", 9, 8, 7, 6, 3}
```

```
In [123... s7={"a", "b", "c", "d", "e", "f", "g", 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

```
In [124... s6.issubset(s7)
```

```
Out[124... True
```

```
In [125... s4.issubset(s6)
```

```
Out[125... False
```

## 11- issuperset()

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

### Syntax

**`set.issuperset(set)`**

```
In [126... s6.issuperset(s7)
```

Out[126... False

In [127... `s7.issuperset(s6)`

Out[127... True

## 12- pop()

*The pop() method removes a random item from the set.*

*This method returns the removed item.*

## Syntax

***set.pop()***

In [128... `s6.pop()`

Out[128... 3

## 13- remove()

*The remove() method removes the specified element from the set.*

*This method is different from the discard() method, because the remove() method will raise an error if the specified item does not exist, and the discard() method will not.*

## Syntax

***set.remove(item)***

In [129... `s6.remove(9)`

In [130... `s6`

Out[130... {6, 7, 8, 'a', 'b', 'd'}

## 14- symmetric\_difference()

*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)***

```
In [131... s6.symmetric_difference(s7)

Out[131... {1, 10, 11, 12, 2, 3, 4, 5, 9, 'c', 'e', 'f', 'g'}
```

## 15- symmetric\_difference\_update()

The `symmetric_difference_update()` method updates the original set by removing items that are present in both sets, and inserting the other items.

### Syntax

**`set.symmetric_difference_update(set)`**

```
In [134... s8=s6.symmetric_difference_update(s7)
```

```
In [135... s6
```

```
Out[135... {6, 7, 8, 'a', 'b', 'd'}
```

## 16- union()

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...)`**

```
In [136... s8={1,2,3,4,5,6,7,7,8,9,10}
```

```
In [137... s9={"a","b","c","d",1,2,4,4,5}
```

```
In [138... s8.union(s9)
```

```
Out[138... {1, 10, 2, 3, 4, 5, 6, 7, 8, 9, 'a', 'b', 'c', 'd'}
```

## 17- update()

The `update()` method updates the current set, by adding items from another set (or any other iterable).

*If an item is present in both sets, only one appearance of this item will be present in the updated set.*

## Syntax

***set.update(set)***

In [141...

```
s8.update(s9)
```

In [142...

Out[142... {1, 10, 2, 3, 4, 5, 6, 7, 8, 9, 'a', 'b', 'c', 'd'}