Tuple

- Tuples are used to hold together multiple objects. Think of them as similar to lists, but without the extensive functionality that the list class gives you.

- One major feature of tuples is that they are immutable i.e. you cannot modify tuples.

- Tuples are usually used in cases where a statement or a user-defined function can safely assume that the collection of values i.e. the tuple of values used will not change.

Tuple Creation

- Tuples are defined by specifying items separated by commas within an optional pair of parentheses.

```
T1 = (5,4,2,6,7)
print(T1)
print(type(T1))
```

```
(5, 4, 2, 6, 7)
<class 'tuple'>
```

```
T2= 5,4,2,6,7
print(T2)
print(type(T2))
```

```
(5, 4, 2, 6, 7)
<class 'tuple'>
```

- Tuple with only single element is defined by specifying item with comma within an optional pair of parentheses.

```
x1 = (5)
print(x1)
print(type(x1))
```

```
5
<class 'int'>
```

```
T2= (5,)
print(T2)
print(type(T2))
```

```
(5,)
<class 'tuple'>
```

Tuple Manipulation

- Tuple are immutable.

```
T1 = (5,4,2,6,7)
T1
```

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

```
T1[2] = 4
```

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

- Tuple elements can be accessed by indexing and slicing methods.

```
T2 = (5,4,2,6,7,8,9)
print(T2[2])
print(T2[-1])
print(T2[2:5])
print(T2[2:5:2])
print(T2[:2])
print(T2[2:])
```

```
2
9
(2, 6, 7)
(2, 7)
(5, 4)
(2, 6, 7, 8, 9)
```

Tuple Unpacking

- Unpacking allows assigning multiple values at a time.

```
T1= (2,'b',3.2,10,(4,2))
print(T1)
```

```
(2, 'b', 3.2, 10, (4, 2))
```

```
a, b, c, d, e = T1
print(a, b, c, d, e)
```

```
2 b 3.2 10 (4, 2)
```

```
x, y = e
print(x, y)
```

```
4 2
```

- Tuples unpacking is also applicable to loops.

```
T1 = [(2,3),(4,5),(6,7),(8,9)]
T1
```

```
[(2, 3), (4, 5), (6, 7), (8, 9)]
```

```
for x,y in T1:
    print(x, y)
```

```
2 3
4 5
6 7
8 9
```

Traversal and Membership in Tuples

Traversal :

```
T1= (2, 'b', 3.2, 10, (4,2))
print(T1)
```

```
(2, 'b', 3.2, 10, (4, 2))
```

```
for i in T1:
    print(i, type(i))
```

```
2 <class 'int'>
b <class 'str'>
3.2 <class 'float'>
10 <class 'int'>
(4, 2) <class 'tuple'>
```

Membership :

```
x1 = 'b'
```

```
x1 in T1
```

```
True
```

```
x2 = 5
```

```
x2 in T1
```

```
False
```

Tuple Examples

- WAP for swapping two numbers.

```
a = 10
b = 20
print(a, b)
```

```
10 20
```

```
a, b = b, a
```

```
print(a, b)
```

```
20 10
```

- WAP to concatenate two tuples.

```
a = (10, 20, 30)
a
```

```
(10, 20, 30)
```

```
b = (20, 40, 50)
b
```

```
(20, 40, 50)
```

```
c = a + b
c
```

```
(10, 20, 30, 20, 40, 50)
```

Sets

- A set is unordered list of elements indentified by curly braces.
- It is mutable like List.
- It can only contain unique elements
- Duplicates are eliminated in set.
- Set do not support indexing.

```
x = {'Swapnil', 'Vishal', 'Pranay'}
x
```

```
{'Pranay', 'Swapnil', 'Vishal'}
```

```
print(type(x))
```

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

```
cset = {11, 22, 11}
print(cset)
```

```
{11, 22}
```

```
cset[1]
```

```
TypeError: 'set' object does not support
           indexing
```

Set class Methods

- Creation methods

```
s1 = set()
s1
```

```
set()
```

```
s2 = set((2,3,4))
s2
```

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

```
print(type(s2))
```

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

```
s2.clear()
s2
```

```
set()
```

- New elements addition

```
s1 = {2,3,4}
s1
```

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

```
s1.add(20)
s1
```

```
{2, 3, 4, 20}
```

```
s1.add(30,40)
s1
```

```
TypeError: add() takes exactly
        one argument (2 given)
```

```
s1.update((20,30))
s1
```

```
{2, 3, 4, 20, 30}
```

Set class Methods

- Deletion Methods – discard (without error), remove , pop

```
s1 = {2,3,4}
print(s1)
s1.discard (2)
print(s1)
s1.discard(20)
```

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

```
s1 = {2,3,4}
print(s1)
s1.remove (2)
print(s1)
s1.remove(20)
```

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

KeyError: 20
```

```
s1 = {2,3,4}
print(s1)

s1.pop ()
print(s1)
```

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

- **WAP to remove the duplicates from given list L = [23,33,33,23,45,67,56,67,45]**

Boolean Operations on Sets

```
aset = {11, 22, 33}
bset = {12, 23, 33}
```

```
# Union of two sets:
print(aset | bset)
```

```
{33, 22, 23, 11, 12}
```

```
print(aset.union(bset))
```

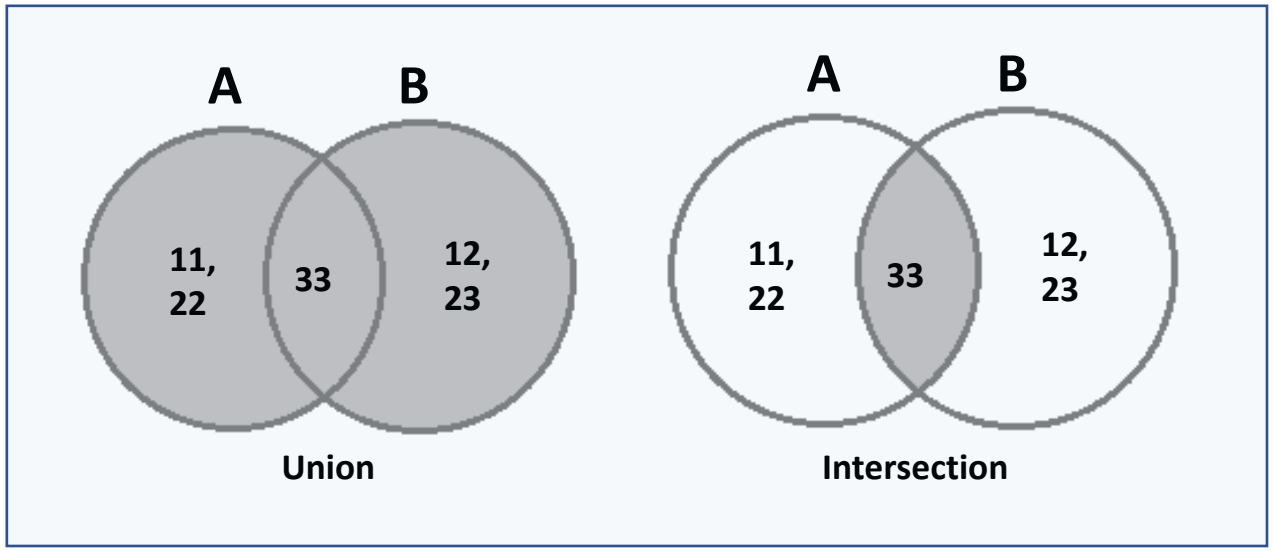
```
{33, 22, 23, 11, 12}
```

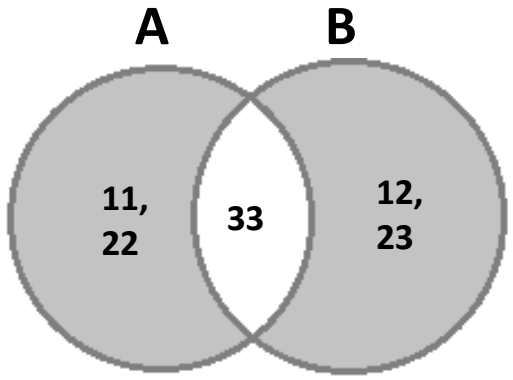
```
# Intersection of two sets:
print(aset & bset)
```

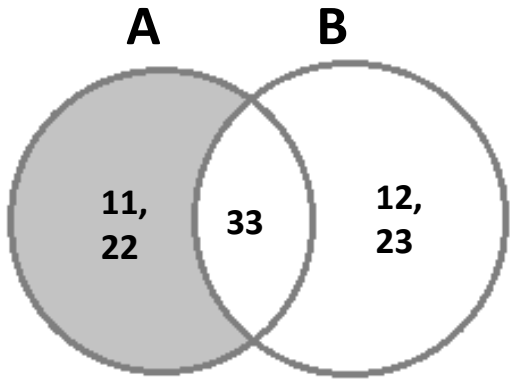
```
{33}
```

```
print(aset.intersection(bset))
```

```
{33}
```







Boolean Operations on Sets

```
# Difference
print(aset - bset)
```

```
{11, 22}
```

```
print(aset.difference(bset))
```

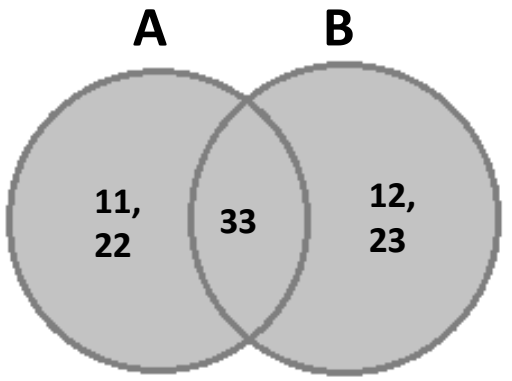
```
{11, 22}
```

```
# Symmetric Difference
print(aset ^ bset)
```

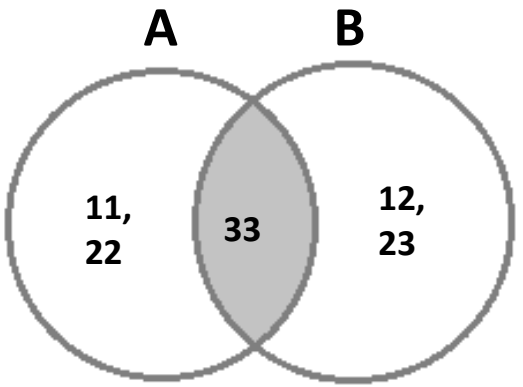
```
{11, 12, 22, 23}
```

```
print(aset.symmetric_difference(bset))
```

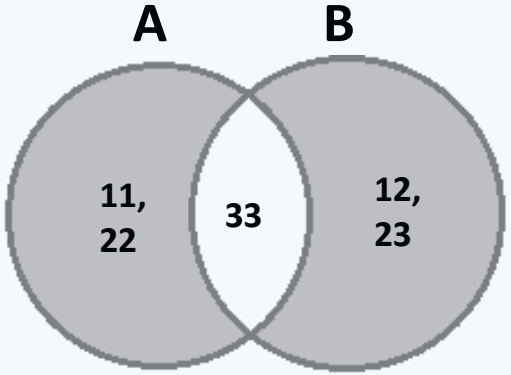
```
{11, 12, 22, 23}
```



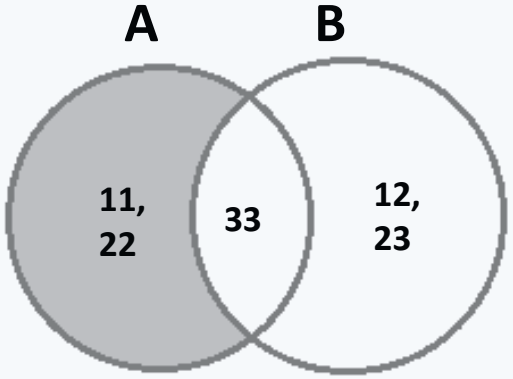
Union



Intersection



Symmetric Difference



Difference

Dictionary

- A dictionary is like an address-book where you can find the address or contact details of a person by knowing only his/her name i.e. we associate keys (name) with values (details).

- Note that the key must be unique just like you cannot find out the correct information if you have two persons with the exact same name.

- One can use only immutable objects (like strings) for the keys of a dictionary but can use either immutable or mutable objects for the values of the dictionary.

- This basically translates to say that you should use only simple objects for keys.

Dictionary

- Pairs of keys and values are specified in a dictionary by using the notation

```
d = { key1 : value1, key2 : value2 }
```

- Notice that the key-value pairs are separated by a colon and the pairs are separated themselves by commas and all this is enclosed in a pair of curly braces.

- Remember that key-value pairs in a dictionary are not ordered in any manner. If you want a particular order, then you will have to sort them yourself before using it.

- The dictionaries that we will be using are instances/objects of the dict class.

Dictionary Creation Methods

- Creation methods –

```
d1 = {}
print (d1)
```

```
{}
```

```
d2 = dict()
print (d2)
```

```
{}
```

```
d1 = {'a' : 3 , 'b':4}
print (d1)
print (type(d1))
```

```
{'a': 3, 'b': 4}
<class 'dict'>
```

```
d1 = {3}
print (d1)
print (type(d1))
```

```
{3}
<class 'set'>
```

- List of tuple and list of list.

```
d1 = dict([('a',3), ('b',5)])
d1
```

```
{'a': 3, 'b': 5}
```

```
d2= dict([['a',3], ['b',5]])
d2
```

```
{'a': 3, 'b': 5}
```

Dictionary Basic Methods

- New **key:value** pair addition and update of values

```
d = {1:'a' , 2:'b'}
d
```

```
{1: 'a', 2: 'b'}
```

```
d[5] = 'g'
d
```

```
{1: 'a', 2: 'b', 5: 'g'}
```

```
d[1] = 'g'
d
```

```
{1: 'g', 2: 'b', 5: 'g'}
```

```
# Only Unique Keys
d = {1:'a', 1:'b'}
d
```

```
{1: 'b'}
```

- Keys has to be immutable data types only.

```
d = {1:'a', 2:'b'}
d
```

```
{1: 'a', 2: 'b'}
```

```
d = {'a':2, 'b':3}
d
```

```
{'a': 2, 'b': 3}
```

```
d = {(1,3):'a', 1:'b'}
d
```

```
{(1, 3): 'a', 1: 'b'}
```

```
d1 = {[1,3]:'a', 1:'b'}
d
```

```
TypeError: unhashable type: 'list'
```

Dictionary Basic Methods

- len() and del()

```
d = {1:'a', 2:'b'}
d
```

```
{1: 'a', 2: 'b'}
```

```
x = len(d)
x
```

```
2
```

```
del d[1]
d
```

```
{2: 'b'}
```

```
del d
d
```

```
NameError: name 'd' is
           not defined
```

- Other operations

```
d = {1:'a', 2:'b'}
d
```

```
{1: 'a', 2: 'b'}
```

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

```
1
2
```

```
d.keys()
```

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

```
for i in d.keys():
    print (i)
```

```
1
2
```

```
d.values()
```

```
dict_values(['a', 'b'])
```

```
for i in d.values():
    print (i)
```

```
a
b
```

```
d.items()
```

```
dict_items([(1, 'a'), (2, 'b')])
```

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

```
1 a
2 b
```

Dictionary Basic Methods

- Shallow copy

```
d = {1:'a', 2:'b'}
print(d)
```

```
{1: 'a', 2: 'b'}
```

```
d1 = d
print(d1)
```

```
{1: 'a', 2: 'b'}
```

```
d1[1] = 'c'
print(d1)
```

```
{1: 'c', 2: 'b'}
```

```
print(d)
```

```
{1: 'c', 2: 'b'}
```

- Deep copy

```
d = {1:'a', 2:'b'}
print(d)
```

```
{1: 'a', 2: 'b'}
```

```
d1 = d.copy()
print(d1)
```

```
{1: 'a', 2: 'b'}
```

```
d1[1] = 'c'
print(d1)
```

```
{1: 'c', 2: 'b'}
```

```
print(d)
```

```
{1: 'a', 2: 'b'}
```

Dictionary Basic Methods

- get()

```
d = {1:'a', 2:'b'}
print (d)
```

```
{1: 'a', 2: 'b'}
```

```
print (d.get(1))
```

```
a
```

```
print (d.get(5))
```

```
None
```

```
print (d.get(5 , 'No Key'))
d
```

```
No Key
{1: 'a', 2: 'b'}
```

- update()

```
d1 = {1:'a', 2:'b'}
d1
```

```
{1: 'a', 2: 'b'}
```

```
d2 = {3:'c', 4:'d'}
d2
```

```
{3: 'c', 4: 'd'}
```

```
d1.update(d2)
```

```
print(d1)
print(d2)
```

```
{1: 'a', 2: 'b', 3: 'c', 4: 'd'}
{3: 'c', 4: 'd'}
```

Dictionary Basic Methods

- pop() operation

```
d = {1:'a', 2:'b'}
d
```

```
{1: 'a', 2: 'b'}
```

```
k = d.pop(1)
k
```

```
'a'
```

```
print(d)
```

```
{2: 'b'}
```

```
d = {1:'a', 2:'b'}
d
```

```
{1: 'a', 2: 'b'}
```

```
k = d.pop(5,'No Key')
k
```

```
'No Key'
```

```
print(d)
```

```
{1: 'a', 2: 'b'}
```

- Dict Comprehension

```
d = {y : y*2 for y in range(5)}
print(d)
```

```
{0: 0, 1: 2, 2: 4, 3: 6, 4: 8}
```

Update element in nested data type

```
x = { 'a' : [4, 5, 6], 'b' : (11, 22, {'c' : 9})}
x
```

```
{'a': [4, 5, 6], 'b': (11, 22, {'c': 9})}
```

To replace element *5* by *55* :

```
x['a'][1] = 55
x
```

```
{'a': [4, 55, 6], 'b': (11, 22, {'c': 9})}
```

To replace element *9* by *99* :

```
x['b'][2]['c'] = 99
x
```

```
{'a': [4, 55, 6], 'b': (11, 22, {'c': 99})}
```