Tuples

1. Tuple is similar to List except that the objects in tuple are immutable which means we cannot change the elements of a tuple once assigned.
2. When we do not want to change the data over time, tuple is a preferred data type.
3. Iterating over the elements of a tuple is faster compared to iterating over a list.

Tuple Creation

```
In [533]: tup1 = ()      # Empty tuple

In [534]: tup2 = (10,30,60)      # tuple of integers numbers

In [535]: tup3 = (10.77,30.66,60.89)      # tuple of float numbers

In [536]: tup4 = ('one','two' , "three")    # tuple of strings

In [537]: tup5 = ('Asif', 25 ,(50, 100),(150, 90))    # Nested tuples

In [538]: tup6 = (100, 'Asif', 17.765)    # Tuple of mixed data types

In [539]: tup7 = ('Asif', 25 ,[50, 100],[150, 90] , {'John' , 'David'} , (99,22,33))

In [540]: len(tup7) #Length of list

Out[540]: 6
```

Tuple Indexing

```
In [541]: tup2[0] # Retreive first element of the tuple

Out[541]: 10

In [542]: tup4[0] # Retreive first element of the tuple

Out[542]: 'one'

In [543]: tup4[0][0] # Nested indexing - Access the first character of the first tuple ele

Out[543]: 'o'

In [544]: tup4[-1] # Last item of the tuple

Out[544]: 'three'
```

```
In [545]: tup5[-1]  # Last item of the tuple
Out[545]: (150, 90)
```

Tuple Slicing

```
In [560]: mytuple = ('one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight'

In [547]: mytuple[0:3] # Return all items from 0th to 3rd index location excluding the ite
Out[547]: ('one', 'two', 'three')

In [548]: mytuple[2:5] # List all items from 2nd to 5th index location excluding the item
Out[548]: ('three', 'four', 'five')

In [549]: mytuple[:3] # Return first three items
Out[549]: ('one', 'two', 'three')

In [550]: mytuple[:2]  # Return first two items
Out[550]: ('one', 'two')

In [551]: mytuple[-3:] # Return last three items
Out[551]: ('six', 'seven', 'eight')

In [552]: mytuple[-2:] # Return last two items
Out[552]: ('seven', 'eight')

In [553]: mytuple[-1] # Return last item of the tuple
Out[553]: 'eight'

In [554]: mytuple[:] # Return whole tuple
Out[554]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

Remove & Change Items

```
In [555]: mytuple
Out[555]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

```
In [556]: del mytuple[0] # Tuples are immutable which means we can't DELETE tuple items
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-556-667a276aa503> in <module>
----> 1 del mytuple[0]

TypeError: 'tuple' object doesn't support item deletion
```

```
In [557]: mytuple[0] = 1 # Tuples are immutable which means we can't CHANGE tuple items
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-557-4cf492702bfd> in <module>
----> 1 mytuple[0] = 1

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

```
In [561]: del mytuple # Deleting entire tuple object is possible
```

Loop through a tuple

```
In [570]: mytuple
```

```
Out[570]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

```
In [571]: for i in mytuple:
              print(i)
```

```
one
two
three
four
five
six
seven
eight
```

```
In [572]: for i in enumerate(mytuple):
              print(i)
```

```
(0, 'one')
(1, 'two')
(2, 'three')
(3, 'four')
(4, 'five')
(5, 'six')
(6, 'seven')
(7, 'eight')
```

Count

```
In [573]: mytuple1 =('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')
```

```
In [574]: mytuple1.count('one') # Number of times item "one" occurred in the tuple.

Out[574]: 3
```

```
In [575]: mytuple1.count('two') # Occurence of item 'two' in the tuple

Out[575]: 2
```

```
In [576]: mytuple1.count('four') #Occurence of item 'four' in the tuple

Out[576]: 1
```

Tuple Membership

```
In [577]: mytuple

Out[577]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

```
In [578]: 'one' in mytuple # Check if 'one' exist in the list

Out[578]: True
```

```
In [579]: 'ten' in mytuple # Check if 'ten' exist in the list

Out[579]: False
```

```
In [581]: if 'three' in mytuple: # Check if 'three' exist in the list
              print('Three is present in the tuple')
          else:
              print('Three is not present in the tuple')

          Three is present in the tuple
```

```
In [583]: if 'eleven' in mytuple:  # Check if 'eleven' exist in the list
              print('eleven is present in the tuple')
          else:
              print('eleven is not present in the tuple')

          eleven is not present in the tuple
```

Index Position

```
In [586]: mytuple

Out[586]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

```
In [587]: mytuple.index('one') # Index of first element equal to 'one'
Out[587]: 0
```

```
In [590]: mytuple.index('five') # Index of first element equal to 'five'
Out[590]: 4
```

```
In [591]: mytuple1
Out[591]: ('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')
```

```
In [593]: mytuple1.index('one') # Index of first element equal to 'one'
Out[593]: 0
```

Sorting

```
In [594]: mytuple2 = (43,67,99,12,6,90,67)
```

```
In [595]: sorted(mytuple2)  # Returns a new sorted list and doesn't change original tuple
Out[595]: [6, 12, 43, 67, 67, 90, 99]
```

```
In [596]: sorted(mytuple2, reverse=True) # Sort in descending order
Out[596]: [99, 90, 67, 67, 43, 12, 6]
```

Sets

1) Unordered & Unindexed collection of items.

2) Set elements are unique. Duplicate elements are not allowed.

3) Set elements are immutable (cannot be changed).

4) Set itself is mutable. We can add or remove items from it.

Set Creation

```
In [634]: myset = {1,2,3,4,5} # Set of numbers
          myset
Out[634]: {1, 2, 3, 4, 5}
```

```
In [635]: len(myset) #Length of the set
Out[635]: 5
```

```
In [636]: my_set = {1,1,2,2,3,4,5,5}
          my_set                          # Duplicate elements are not allowed.

Out[636]: {1, 2, 3, 4, 5}

In [637]: myset1 = {1.79,2.08,3.99,4.56,5.45} # Set of float numbers
          myset1

Out[637]: {1.79, 2.08, 3.99, 4.56, 5.45}

In [638]: myset2 = {'Asif' , 'John' , 'Tyrion'} # Set of Strings
          myset2

Out[638]: {'Asif', 'John', 'Tyrion'}

In [639]: myset3 = {10,20, "Hola", (11, 22, 32)} # Mixed datatypes
          myset3

Out[639]: {(11, 22, 32), 10, 20, 'Hola'}

In [640]: myset3 = {10,20, "Hola", [11, 22, 32]} # set doesn't allow mutable items like li
          myset3

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-640-d23fdc3a319e> in <module>
          ----> 1 myset3 = {10,20, "Hola", [11, 22, 32]} # set doesn't allow mutable item
          s like lists
                2 myset3

          TypeError: unhashable type: 'list'


In [641]: myset4 = set() # Create an empty set
          print(type(myset4))

          <class 'set'>

In [673]: my_set1 = set(('one' , 'two' , 'three' , 'four'))
          my_set1

Out[673]: {'four', 'one', 'three', 'two'}
```

Loop through a Set

In [776]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}

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

```
eight
one
seven
three
five
two
six
four
```

In [777]:
```
for i in enumerate(myset):
    print(i)
```

```
(0, 'eight')
(1, 'one')
(2, 'seven')
(3, 'three')
(4, 'five')
(5, 'two')
(6, 'six')
(7, 'four')
```

Set Membership

In [675]:
```
myset
```

Out[675]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [676]:
```
'one' in myset # Check if 'one' exist in the set
```

Out[676]:
```
True
```

In [677]:
```
'ten' in myset # Check if 'ten' exist in the set
```

Out[677]:
```
False
```

In [678]:
```
if 'three' in myset: # Check if 'three' exist in the set
    print('Three is present in the set')
else:
    print('Three is not present in the set')
```

```
Three is present in the set
```

In [679]:
```
if 'eleven' in myset:  # Check if 'eleven' exist in the list
    print('eleven is present in the set')
else:
    print('eleven is not present in the set')
```

```
eleven is not present in the set
```

Add & Remove Items

In [680]:
```
myset
```
Out[680]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [681]:
```
myset.add('NINE') # Add item to a set using add() method
myset
```
Out[681]:
```
{'NINE', 'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [683]:
```
myset.update(['TEN' , 'ELEVEN' , 'TWELVE'])  # Add multiple item to a set using
myset
```
Out[683]:
```
{'ELEVEN',
 'NINE',
 'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [684]:
```
myset.remove('NINE') # remove item in a set using remove() method
myset
```
Out[684]:
```
{'ELEVEN',
 'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [685]:
```
myset.discard('TEN') # remove item from a set using discard() method
myset
```
Out[685]:
```
{'ELEVEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

```
In [688]: myset.clear() # Delete all items in a set
          myset

Out[688]: set()

In [689]: del myset # Delete the set object
          myset

          ---------------------------------------------------------------------------
          NameError                                 Traceback (most recent call last)
          <ipython-input-689-0912ea1b8932> in <module>
                1 del myset
          ----> 2 myset

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

Copy Set

```
In [705]: myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
          myset

Out[705]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}

In [706]: myset1 = myset  # Create a new reference "myset1"
          myset1

Out[706]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}

In [707]: id(myset) , id(myset1)  # The address of both myset & myset1 will be the same as

Out[707]: (1537349033320, 1537349033320)

In [708]: my_set = myset.copy() # Create a copy of the list
          my_set

Out[708]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}

In [710]: id(my_set) # The address of my_set will be different from myset because my_set i

Out[710]: 1537352902024

In [711]: myset.add('nine')
          myset

Out[711]: {'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}

In [712]: myset1 # myset1 will be also impacted as it is pointing to the same Set

Out[712]: {'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}

In [713]: my_set # Copy of the set won't be impacted due to changes made on the original S

Out[713]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

Set Operation

Union

```
In [757]: A = {1,2,3,4,5}
          B = {4,5,6,7,8}
          C = {8,9,10}
```

```
In [758]: A | B  # Union of A and B (All elements from both sets. NO DUPLICATES)
```

```
Out[758]: {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [759]: A.union(B) # Union of A and B
```

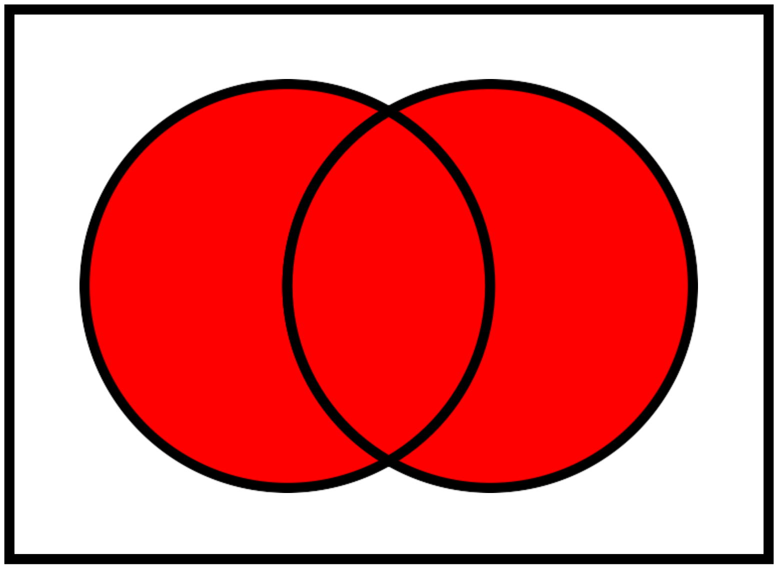
```
Out[759]: {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [760]: A.union(B, C)  # Union of A, B and C.
```

```
Out[760]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
In [761]: """
          Updates the set calling the update() method with union of A , B & C.

          For below example Set A will be updated with union of A,B & C.
          """
          A.update(B,C)
          A
```

```
Out[761]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```



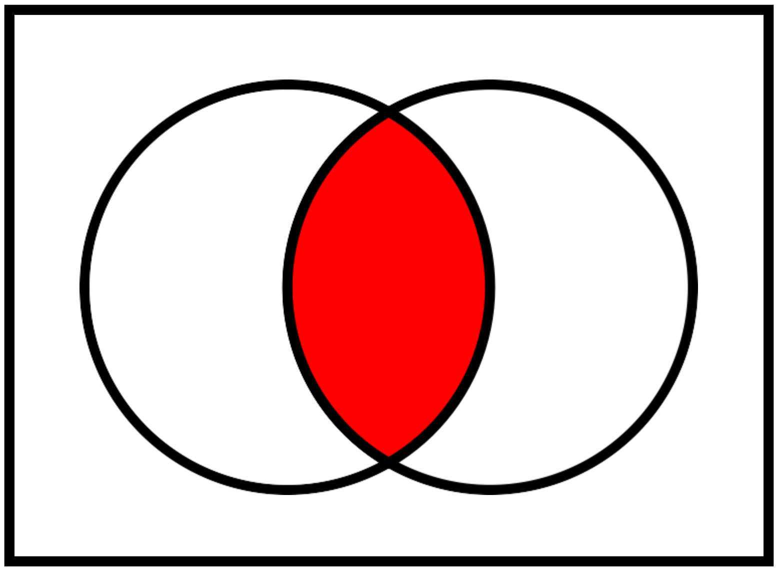
Intersection

```
In [762]: A = {1,2,3,4,5}
          B = {4,5,6,7,8}
```

```
In [763]: A & B  # Intersection of A and B (Common items in both sets)

Out[763]: {4, 5}
```

```
In [764]: A.intersection(B)  Intersection of A and B

            File "<ipython-input-764-f01b60f4d31d>", line 1
              A.intersection(B)  Intersection of A and B
                                             ^
          SyntaxError: invalid syntax
```

```
In [765]: """
          Updates the set calling the intersection_update() method with the intersection o

          For below example Set A will be updated  with the intersection of A & B.
          """
          A.intersection_update(B)
          A

Out[765]: {4, 5}
```



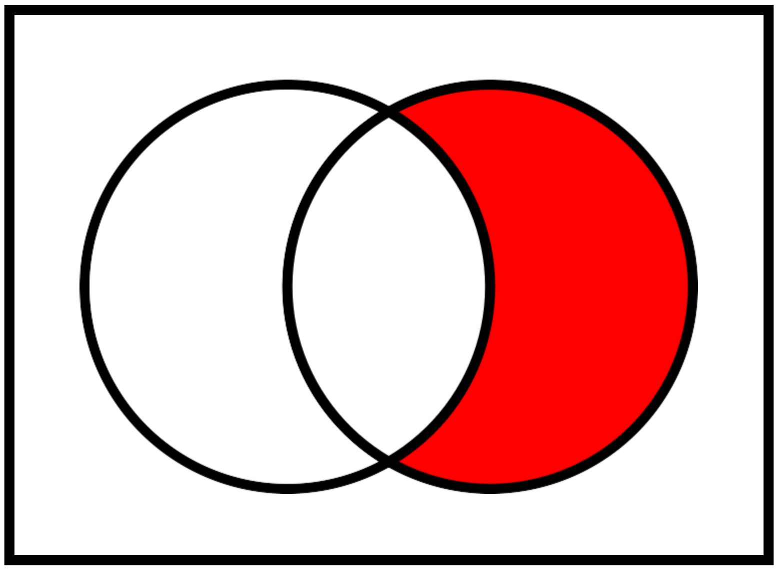
Difference

```
In [766]: A = {1,2,3,4,5}
          B = {4,5,6,7,8}
```

```
In [767]: A - B  # set of elements that are only in A but not in B

Out[767]: {1, 2, 3}
```

```
In [768]: A.difference(B) # Difference of sets

Out[768]: {1, 2, 3}
```

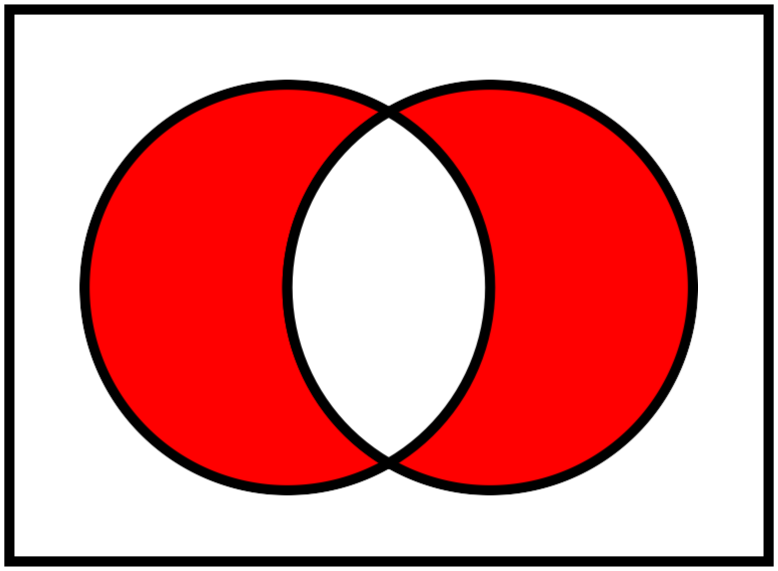
```
In [769]: B- A  # set of elements that are only in B but not in A

Out[769]: {6, 7, 8}

In [770]: B.difference(A)

Out[770]: {6, 7, 8}

In [771]: """
          Updates the set calling the difference_update() method with the difference of se

          For below example Set B will be updated  with the difference of B & A.
          """
          B.difference_update(A)
          B

Out[771]: {6, 7, 8}
```



Symmetric Difference

```
In [772]: A = {1,2,3,4,5}
          B = {4,5,6,7,8}

In [773]: A ^ B # Symmetric difference (Set of elements in A and B but not in both. "EXCLU

Out[773]: {1, 2, 3, 6, 7, 8}

In [774]: A.symmetric_difference(B)  # Symmetric difference of sets

Out[774]: {1, 2, 3, 6, 7, 8}
```

In [775]:
```
"""
Updates the set calling the symmetric_difference_update() method with the symmet

For below example Set A will be updated  with the symmetric difference of A & B.
"""

A.symmetric_difference_update(B)
A
```
Out[775]:
```
{1, 2, 3, 6, 7, 8}
```



Subset , Superset & Disjoint

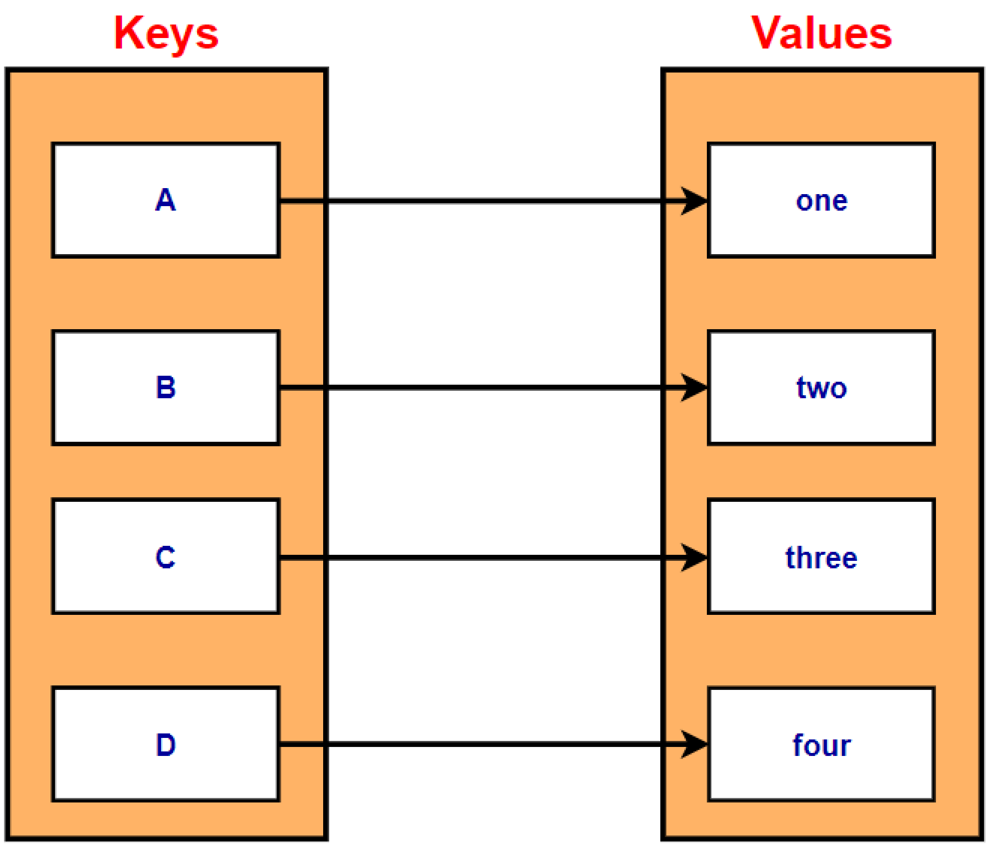
In [784]:
```
A = {1,2,3,4,5,6,7,8,9}
B = {3,4,5,6,7,8}
C = {10,20,30,40}
```
In [785]:
```
B.issubset(A) # Set B is said to be the subset of set A if all elements of B are
```
Out[785]:
```
True
```
In [786]:
```
A.issuperset(B) # Set A is said to be the superset of set B if all elements of B
```
Out[786]:
```
True
```
In [787]:
```
C.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common e
```
Out[787]:
```
True
```
In [788]:
```
B.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common e
```
Out[788]:
```
False
```

Other Builtin functions

```
In [789]: A
Out[789]: {1, 2, 3, 4, 5, 6, 7, 8, 9}

In [790]: sum(A)
Out[790]: 45

In [791]: max(A)
Out[791]: 9

In [792]: min(A)
Out[792]: 1

In [793]: len(A)
Out[793]: 9

In [795]: list(enumerate(A))
Out[795]: [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]

In [798]: D= sorted(A,reverse=True)
          D
Out[798]: [9, 8, 7, 6, 5, 4, 3, 2, 1]

In [799]: sorted(D)
Out[799]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Dictionary

- Dictionary is a mutable data type in Python.
- A python dictionary is a collection of key and value pairs separated by a colon (:) & enclosed in curly braces {}.
- Keys must be unique in a dictionary, duplicate values are allowed.



mydict = {'A':'one' , 'B':'two' , 'C':'three' , 'D' :'four'}

Create Dictionary

```
In [947]: mydict = dict() # empty dictionary
          mydict
Out[947]: {}
```

```
In [948]: mydict = {}   # empty dictionary
          mydict
Out[948]: {}
```

```
In [949]: mydict = {1:'one' , 2:'two' , 3:'three'} # dictionary with integer keys
          mydict
Out[949]: {1: 'one', 2: 'two', 3: 'three'}
```

```
In [950]: mydict = dict({1:'one' , 2:'two' , 3:'three'}) # Create dictionary using dict()
          mydict
Out[950]: {1: 'one', 2: 'two', 3: 'three'}
```

```
In [951]: mydict = {'A':'one' , 'B':'two' , 'C':'three'} # dictionary with character keys
          mydict

Out[951]: {'A': 'one', 'B': 'two', 'C': 'three'}

In [318]: mydict = {1:'one' , 'A':'two' , 3:'three'} # dictionary with mixed keys
          mydict

Out[318]: {1: 'one', 'A': 'two', 3: 'three'}

In [319]: mydict.keys() # Return Dictionary Keys using keys() method

Out[319]: dict_keys([1, 'A', 3])

In [320]: mydict.values() # Return Dictionary Values using values() method

Out[320]: dict_values(['one', 'two', 'three'])

In [321]: mydict.items() # Access each key-value pair within a dictionary

Out[321]: dict_items([(1, 'one'), ('A', 'two'), (3, 'three')])

In [955]: mydict = {1:'one' , 2:'two' , 'A':['asif' , 'john' , 'Maria']} # dictionary with
          mydict

Out[955]: {1: 'one', 2: 'two', 'A': ['asif', 'john', 'Maria']}

In [956]: mydict = {1:'one' , 2:'two' , 'A':['asif' , 'john' , 'Maria'],  'B':('Bat' , 'ca
          mydict

Out[956]: {1: 'one',
           2: 'two',
           'A': ['asif', 'john', 'Maria'],
           'B': ('Bat', 'cat', 'hat')}

  In [1]: mydict = {1:'one' , 2:'two' , 'A':{'Name':'asif' , 'Age' :20},  'B':('Bat' , 'ca
          mydict

  Out[1]: {1: 'one',
           2: 'two',
           'A': {'Name': 'asif', 'Age': 20},
           'B': ('Bat', 'cat', 'hat')}

In [957]: keys = {'a' , 'b' , 'c' , 'd'}
          mydict3 = dict.fromkeys(keys)   # Create a dictionary from a sequence of keys
          mydict3

Out[957]: {'c': None, 'd': None, 'a': None, 'b': None}

In [958]: keys = {'a' , 'b' , 'c' , 'd'}
          value = 10
          mydict3 = dict.fromkeys(keys , value)  # Create a dictionary from a sequence of
          mydict3

Out[958]: {'c': 10, 'd': 10, 'a': 10, 'b': 10}
```

```
In [959]: keys = {'a' , 'b' , 'c' , 'd'}
          value = [10,20,30]
          mydict3 = dict.fromkeys(keys , value)  # Create a dictionary from a sequence of
          mydict3

Out[959]: {'c': [10, 20, 30], 'd': [10, 20, 30], 'a': [10, 20, 30], 'b': [10, 20, 30]}
```

```
In [960]: value.append(40)
          mydict3

Out[960]: {'c': [10, 20, 30, 40],
           'd': [10, 20, 30, 40],
           'a': [10, 20, 30, 40],
           'b': [10, 20, 30, 40]}
```

Accessing Items

```
In [961]: mydict = {1:'one' , 2:'two' , 3:'three' , 4:'four'}
          mydict

Out[961]: {1: 'one', 2: 'two', 3: 'three', 4: 'four'}

In [962]: mydict[1] # Access item using key

Out[962]: 'one'

In [963]: mydict.get(1) # Access item using get() method

Out[963]: 'one'

In [964]: mydict1 = {'Name':'Asif' , 'ID': 74123 , 'DOB': 1991 , 'job' :'Analyst'}
          mydict1

Out[964]: {'Name': 'Asif', 'ID': 74123, 'DOB': 1991, 'job': 'Analyst'}

In [965]: mydict1['Name']  # Access item using key

Out[965]: 'Asif'

In [966]: mydict1.get('job')  # Access item using get() method

Out[966]: 'Analyst'
```

Add, Remove & Change Items

```
In [967]: mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
          mydict1

Out[967]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

```
In [968]: mydict1['DOB'] = 1992  # Changing Dictionary Items
          mydict1['Address'] = 'Delhi'
          mydict1

Out[968]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1992, 'Address': 'Delhi'}

In [969]: dict1 = {'DOB':1995}
          mydict1.update(dict1)
          mydict1

Out[969]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}

In [970]: mydict1['Job'] = 'Analyst' # Adding items in the dictionary
          mydict1

Out[970]: {'Name': 'Asif',
           'ID': 12345,
           'DOB': 1995,
           'Address': 'Delhi',
           'Job': 'Analyst'}

In [971]: mydict1.pop('Job') # Removing items in the dictionary using Pop method
          mydict1

Out[971]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}

In [972]: mydict1.popitem() # A random item is removed

Out[972]: ('Address', 'Delhi')

In [973]: mydict1

Out[973]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1995}

In [974]: del[mydict1['ID']] # Removing item using del method
          mydict1

Out[974]: {'Name': 'Asif', 'DOB': 1995}

In [975]: mydict1.clear() # Delete all items of the dictionary using clear method
          mydict1

Out[975]: {}

In [976]: del mydict1 # Delete the dictionary object
          mydict1

          ---------------------------------------------------------------------------
          NameError                                 Traceback (most recent call last)
          <ipython-input-976-da2fba4eca0f> in <module>
                1 del mydict1 # Delete the dictionary object
          ----> 2 mydict1

          NameError: name 'mydict1' is not defined
```

Copy Dictionary

```
In [977]: mydict = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
          mydict

Out[977]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}

In [978]: mydict1 = mydict # Create a new reference "mydict1"

In [979]: id(mydict) , id(mydict1) # The address of both mydict & mydict1 will be the same

Out[979]: (1537346312776, 1537346312776)

In [980]: mydict2 = mydict.copy() # Create a copy of the dictionary

In [981]: id(mydict2) # The address of mydict2 will be different from mydict because mydic

Out[981]: 1537345875784

In [982]: mydict['Address'] = 'Mumbai'

In [983]: mydict

Out[983]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}

In [984]: mydict1 # mydict1 will be also impacted as it is pointing to the same dictionary

Out[984]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}

In [985]: mydict2 # Copy of list won't be impacted due to the changes made in the original

Out[985]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

Loop through a Dictionary

```
In [986]: mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki' ,
          mydict1

Out[986]: {'Name': 'Asif',
           'ID': 12345,
           'DOB': 1991,
           'Address': 'Hilsinki',
           'Job': 'Analyst'}

In [987]: for i in mydict1:
              print(i , ':' , mydict1[i]) # Key & value pair

          Name : Asif
          ID : 12345
          DOB : 1991
          Address : Hilsinki
          Job : Analyst
```

```
In [988]: for i in mydict1:
              print(mydict1[i]) # Dictionary items
```

```
Asif
12345
1991
Hilsinki
Analyst
```

Dictionary Membership

```
In [989]: mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
          mydict1

Out[989]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

```
In [990]: 'Name' in mydict1 # Test if a key is in a dictionary or not.

Out[990]: True
```

```
In [991]: 'Asif' in mydict1  # Membership test can be only done for keys.

Out[991]: False
```

```
In [992]: 'ID' in mydict1

Out[992]: True
```

```
In [993]: 'Address' in mydict1

Out[993]: False
```

All / Any

The **all()** method returns:

- **True** - If all all keys of the dictionary are true
- **False** - If any key of the dictionary is false

The **any()** function returns True if any key of the dictionary is True. If not, any() returns False.

```
In [995]: mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
          mydict1

Out[995]: {'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

```
In [996]: all(mydict1) # Will Return false as one value is false (Value 0)

Out[996]: True
```