# 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 [3]:
```
tup1 = () # Empty tuple
```

In [4]:
```
tup2 = (10,30,60) # tuple of integers numbers
```

In [5]:
```
tup3 = (10.77,30.66,60.89) # tuple of float numbers
```

In [6]:
```
tup4 = ('one', 'two', "three") # tuple of strings
```

In [7]:
```
tup5 = ('Asif', 25 ,(50, 100),(150, 90)) # Nested tuples
```

In [8]:
```
tup6 = (100, 'Asif', 17.765) # Tuple of mixed data types
```

In [9]:
```
tup7 = ('Asif', 25 ,[50, 100],[150, 90] , {'John' , 'David'} , (99,22,33))
```

In [10]:
```
len(tup7) #Length of list
```

Out[10]:
```
6
```

# Tuple Indexing

In [12]:
```
tup2[0] # Retreive first element of the tuple
```

Out[12]:
```
10
```

In [13]:
```
tup4[0] # Retreive first element of the tuple
```

Out[13]:
```
'one'
```

In [14]:
```
tup4[0][0] # Nested indexing - Access the first character of the first tuple ele
```

Out[14]:
```
'o'
```

In [15]:
```
tup4[-1] # Last item of the tuple
```

Out[15]:
```
'three'
```

In [16]:
```
tup5[-1] # Last item of the tuple
```

Out[16]: (150, 90)

# Tuple Slicing

In [18]:
```
mytuple = ('one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight'
```

In [19]:
```
mytuple[0:3] # Return all items from 0th to 3rd index location excluding the ite
```

Out[19]: ('one', 'two', 'three')

In [20]:
```
mytuple[2:5] # List all items from 2nd to 5th index location excluding the item
```

Out[20]: ('three', 'four', 'five')

In [21]:
```
mytuple[:3] # Return first three items
```

Out[21]: ('one', 'two', 'three')

In [22]:
```
mytuple[:2] # Return first two items
```

Out[22]: ('one', 'two')

In [23]:
```
mytuple[-3:] # Return last three items
```

Out[23]: ('six', 'seven', 'eight')

In [24]:
```
mytuple[-2:] # Return last two items
```

Out[24]: ('seven', 'eight')

In [25]:
```
mytuple[-1] # Return last item of the tuple
```

Out[25]: 'eight'

In [26]:
```
mytuple[:] # Return whole tuple
```

Out[26]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

# Remove & Change Items

In [28]:
```
mytuple
```

Out[28]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

In [29]:
```
del mytuple[0] # Tuples are immutable which means we can't DELETE tuple items
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[29], line 1
----> 1 del mytuple[0]

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

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

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

In [ ]:
```
mytuple
```

# Loop through a tuple

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

In [ ]:
```
mytuple
```

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

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

# Count

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

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

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

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

In [ ]:
```
mytuple
```

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

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

In [ ]:
```
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')
```

In [ ]:
```
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')
```

# Index Position

In [ ]:
```
mytuple
```

In [ ]:
```
mytuple.index('one') # Index of 'one' is printed
```

In [ ]:
```
mytuple.index('five') # Index of 'five' is printed
```

In [ ]:
```
mytuple1
```

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

# Sorting

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

In [ ]:
```
sorted(mytuple2) # Returns a new sorted list and doesn't change original tuple
```

In [ ]:
```
sorted(mytuple2, reverse=True) # Sort in descending order
```

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

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

In [ ]:
```
my_set = {1,1,2,2,3,4,5,5}
my_set
```

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

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

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

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

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

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

# Loop through a Set

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

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

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

# Set Membership

In [ ]:
```
myset
```

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

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

In [ ]:
```
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')
```

In [ ]:
```
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')
```

# Add & Remove Items

In [ ]:
```
myset
```

In [ ]:
```
myset.add('NINE') # Add item to a set using add() method
myset
```

In [ ]:
```
myset.update(['TEN' , 'ELEVEN' , 'TWELVE']) # Add multiple item to a set using u
myset
```

In [ ]:
```
myset.remove('NINE') # remove item in a set using remove() method
myset
```

In [ ]:
```
myset.discard('TEN') # remove item from a set using discard() method
myset
```

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

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

# Copy Set

In [52]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
myset
```

Out[52]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [54]:
```
myset1 = myset # Create a new reference "myset1"
myset1
```

Out[54]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [72]:
```
id(myset) , id(myset1) # The address of both myset & myset1 will be the same as
```

Out[72]:
```
(2477347972576, 2477347972576)
```

In [58]:
```
my_set = myset.copy() # Create a copy of the list
my_set
```

Out[58]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [64]:
```
id(my_set) # The address of my_set will be different from myset because my_set i
```

Out[64]:
```
2477347973696
```

In [66]:
```
myset.add('nine')
myset
```

Out[66]:
```
{'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

In [68]:
```
myset1 # myset1 will be also impacted as it is pointing to the same Set
```

Out[68]:
```
{'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

In [70]:
```
my_set # Copy of the set won't be impacted due to changes made on the original S
```

Out[70]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

# Set Operation

# Union

In [76]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

```
C = {8,9,10}
```

In [78]:
```
A | B # Union of A and B (All elements from both sets. NO DUPLICATES)
```

Out[78]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [80]:
```
A.union(B) # Union of A and B
```

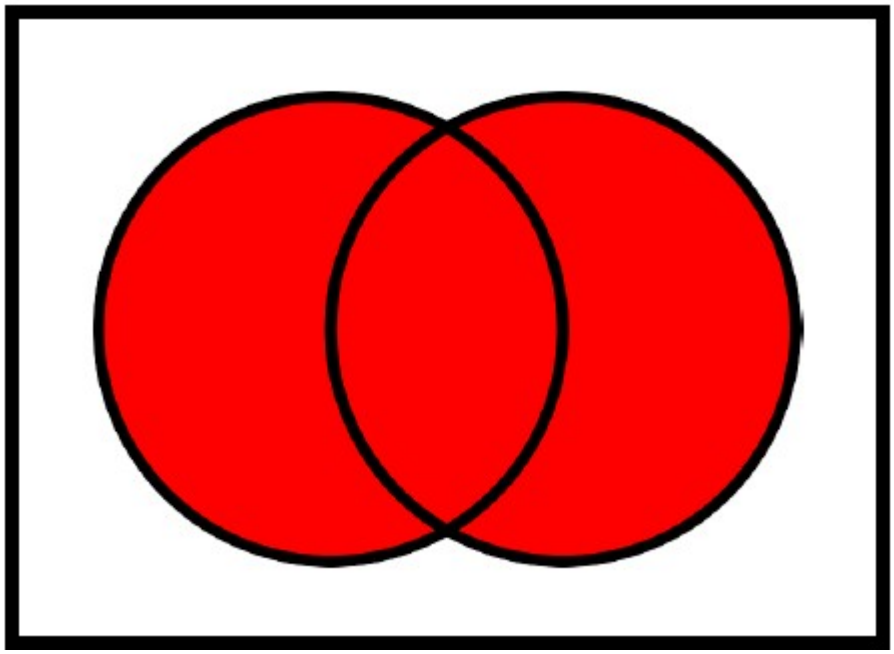
Out[80]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [82]:
```
A.union(B, C) # Union of A, B and C.
```

Out[82]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [88]:
```
"""
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[88]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```



# Intersection

In [92]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [94]:
```
A & B # Intersection of A and B (Common items in both sets)
```

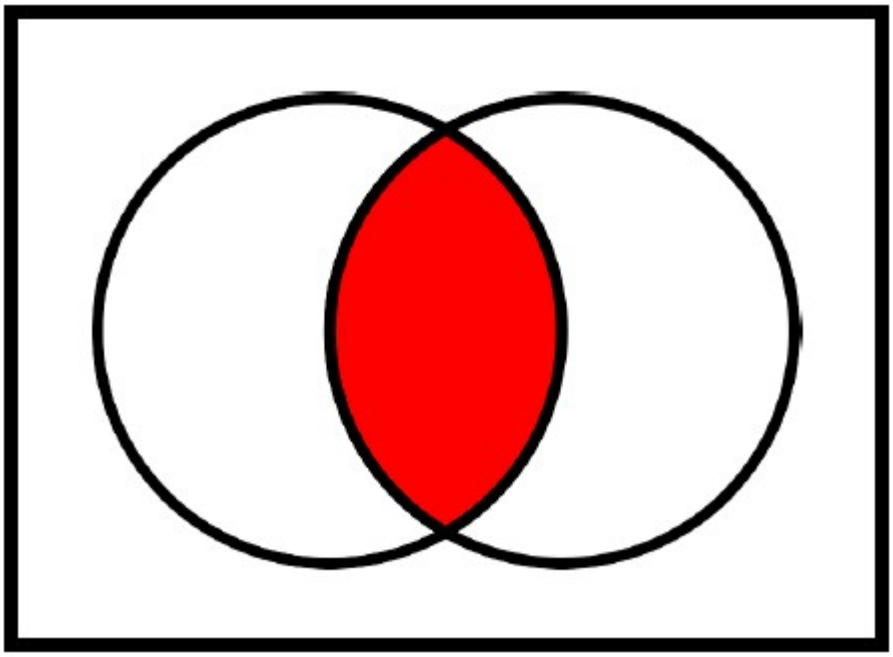
Out[94]:
```
{4, 5}
```

In [98]:
```
A.intersection(B) #Intersection of A and B
```

Out[98]:
```
{4, 5}
```

In [100…
```
"""
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[100…
```
{4, 5}
```



# Difference

In [104…
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [106…
```
A - B # set of elements that are only in A but not in B
```

Out[106…
```
{1, 2, 3}
```

In [108…
```
A.difference(B) # Difference of sets
```

Out[108…
```
{1, 2, 3}
```

In [110…
```
B - A # set of elements that are only in B but not in A
```

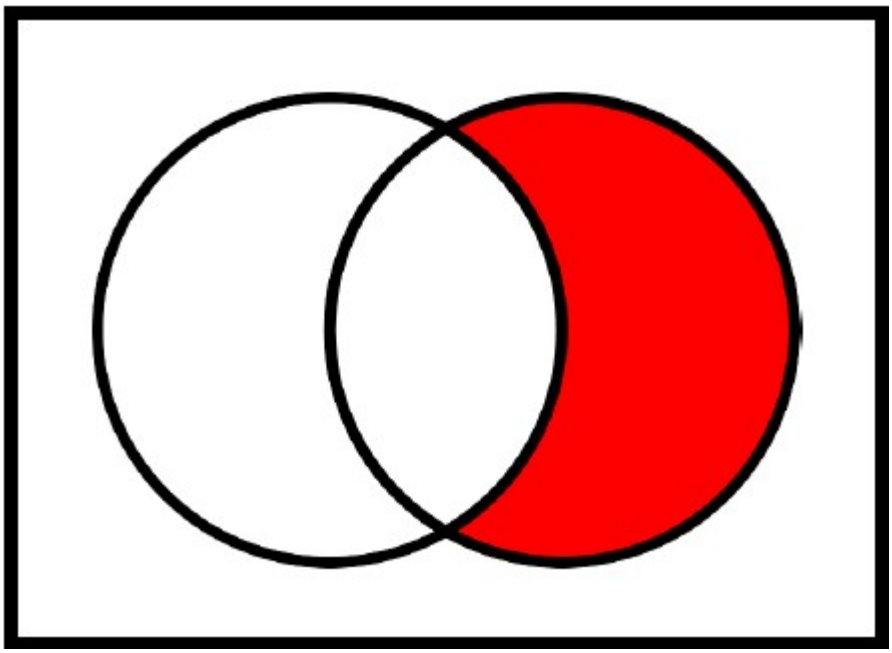
Out[110…
```
{6, 7, 8}
```

In [112…
```
B.difference(A)
```

Out[112…
```
{6, 7, 8}
```

In [114…
```
"""
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[114… {6, 7, 8}



# Symmetric Difference

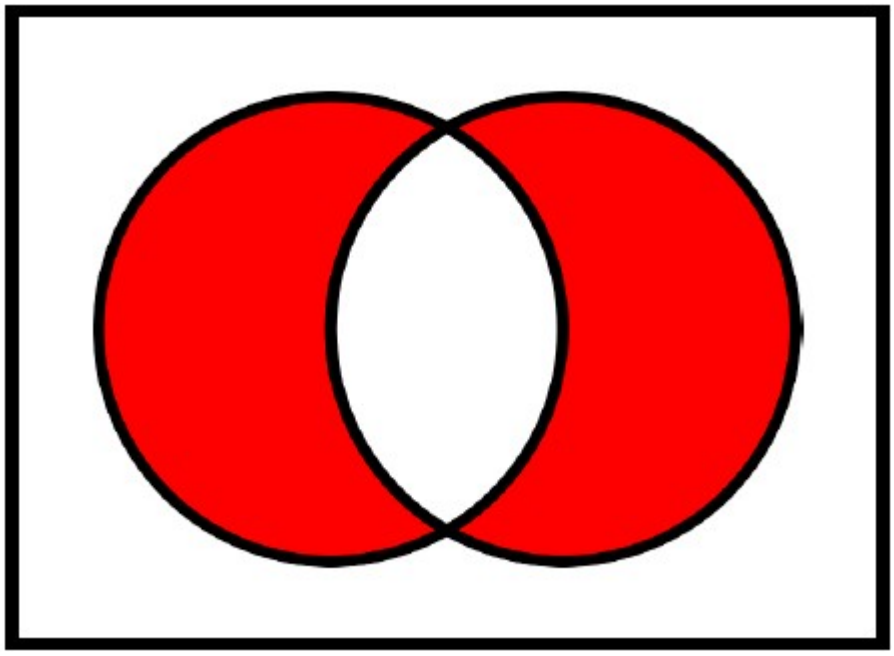
In [118…
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [122…
```
A ^ B # Symmetric difference (Set of elements in A and B but not in both. "EXCLU
```

Out[122… {1, 2, 3, 6, 7, 8}

In [124…
```
"""
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[124… {1, 2, 3, 6, 7, 8}



# Subset , Superset & Disjoint

In [128…
```
A = {1,2,3,4,5,6,7,8,9}
B = {3,4,5,6,7,8}
C = {10,20,30,40}
```

In [130…
```
B.issubset(A) # Set B is said to be the subset of set A if all elements of B are
```

Out[130…
```
True
```

In [132…
```
A.issuperset(B) # Set A is said to be the superset of set B if all elements of B
```

Out[132…
```
True
```

In [134…
```
C.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common e
```

Out[134…
```
True
```

In [136…
```
B.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common e
```

Out[136…
```
False
```

In [138…
```
B.isdisjoint(C)
```

Out[138…
```
True
```

# Other Built-in Functions

In [142…
```
A
```

Out[142…
```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [144…
```
sum(A)
```
Out[144… 45

In [146…
```
max(A)
```
Out[146… 9

In [148…
```
min(A)
```
Out[148… 1

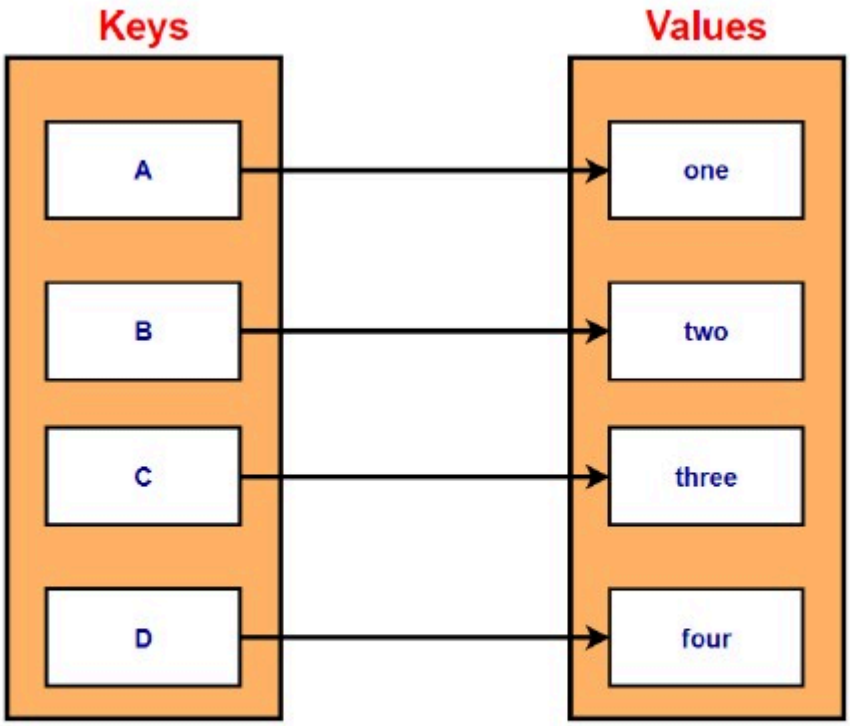
In [150…
```
len(A)
```
Out[150… 9

In [152…
```
list(enumerate(A))
```
Out[152… [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]

In [154…
```
D= sorted(A,reverse=True)
D
```
Out[154… [9, 8, 7, 6, 5, 4, 3, 2, 1]

# 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 [160…
```
mydict = dict() # empty dictionary
mydict
```

Out[160…
```
{}
```

In [162…
```
mydict = {} # empty dictionary
mydict
```

Out[162…
```
{}
```

In [164…
```
mydict = {1:'one' , 2:'two' , 3:'three'} # dictionary with integer keys
mydict
```

Out[164…
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [166…
```
mydict = dict({1:'one' , 2:'two' , 3:'three'}) # Create dictionary using dict()
mydict
```

Out[166…
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [168…
```
mydict = {'A':'one' , 'B':'two' , 'C':'three'} # dictionary with character keys
mydict
```

Out[168…
```
{'A': 'one', 'B': 'two', 'C': 'three'}
```

In [170…
```
mydict = {1:'one' , 'A':'two' , 3:'three'} # dictionary with mixed keys
mydict
```

Out[170…
```
{1: 'one', 'A': 'two', 3: 'three'}
```

In [172…
```
mydict.keys() # Return Dictionary Keys using keys() method
```

Out[172…
```
dict_keys([1, 'A', 3])
```

In [174…
```
mydict.values() # Return Dictionary Values using values() method
```

Out[174…
```
dict_values(['one', 'two', 'three'])
```

In [176…
```
mydict.items() # Access each key-value pair within a dictionary
```

Out[176…
```
dict_items([(1, 'one'), ('A', 'two'), (3, 'three')])
```

In [180…
```
mydict = {1:'one' , 2:'two' , 'A':['asif' , 'john' , 'Maria']} # dictionary with
mydict
```

Out[180…
```
{1: 'one', 2: 'two', 'A': ['asif', 'john', 'Maria']}
```

In [182…
```
mydict = {1:'one' , 2:'two' , 'A':['asif' , 'john' , 'Maria'], 'B':('Bat' , 'cat
mydict
```

Out[182…
```
{1: 'one',
 2: 'two',
 'A': ['asif', 'john', 'Maria'],
 'B': ('Bat', 'cat', 'hat')}
```

In [184…
```
mydict = {1:'one' , 2:'two' , 'A':{'Name':'asif' , 'Age' :20}, 'B':('Bat' , 'cat
mydict
```

Out[184…
```
{1: 'one',
 2: 'two',
 'A': {'Name': 'asif', 'Age': 20},
 'B': ('Bat', 'cat', 'hat')}
```

In [186…
```
keys = {'a' , 'b' , 'c' , 'd'}
mydict3 = dict.fromkeys(keys) # Create a dictionary from a sequence of keys
mydict3
```

Out[186…
```
{'a': None, 'd': None, 'b': None, 'c': None}
```

In [188…
```
keys = {'a' , 'b' , 'c' , 'd'}
value = 10
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[188…
```
{'a': 10, 'd': 10, 'b': 10, 'c': 10}
```

In [190…
```
keys = {'a' , 'b' , 'c' , 'd'}
value = [10,20,30]
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[190…
```
{'a': [10, 20, 30], 'd': [10, 20, 30], 'b': [10, 20, 30], 'c': [10, 20, 30]}
```

In [192…
```
value.append(40)
mydict3
```

Out[192…
```
{'a': [10, 20, 30, 40],
 'd': [10, 20, 30, 40],
 'b': [10, 20, 30, 40],
 'c': [10, 20, 30, 40]}
```

# Accessing Items

In [197…
```
mydict = {1:'one' , 2:'two' , 3:'three' , 4:'four'}
mydict
```

Out[197…
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [199…
```
mydict[1] # Access item using key
```

Out[199…
```
'one'
```

In [201…
```
mydict.get(1) # Access item using get() method
```

Out[201…
```
'one'
```

In [205…
```
mydict1 = {'Name':'Asif' , 'ID': 74123 , 'DOB': 1991 , 'job' :'Analyst'}
mydict1
```

Out[205…
```
{'Name': 'Asif', 'ID': 74123, 'DOB': 1991, 'job': 'Analyst'}
```

In [207…
```
mydict1['Name'] # Access item using key
```

Out[207…
```
'Asif'
```

In [209…
```
mydict1.get('job') # Access item using get() method
```

Out[209…
```
'Analyst'
```

# Add, Remove & Change Items

In [212…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
mydict1
```

Out[212…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [214…
```
mydict1['DOB'] = 1992 # Changing Dictionary Items
mydict1['Address'] = 'Delhi'
mydict1
```

Out[214…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1992, 'Address': 'Delhi'}
```

In [216…
```
dict1 = {'DOB':1995}
mydict1.update(dict1)
mydict1
```

Out[216…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}
```

In [218…
```
mydict1['Job'] = 'Analyst' # Adding items in the dictionary
mydict1
```

Out[218…
```
{'Name': 'Asif',
 'ID': 12345,
 'DOB': 1995,
 'Address': 'Delhi',
 'Job': 'Analyst'}
```

In [220…
```
mydict1.pop('Job') # Removing items in the dictionary using Pop method
mydict1
```

Out[220…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}
```

In [222…
```
mydict1.popitem() # A random item is removed
mydict1
```

Out[222…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995}
```

In [232…
```
del[mydict1['ID']] # Removing item using del method .(Use del only once, else th
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[232], line 1
----> 1 del[mydict1['ID']]

KeyError: 'ID'
```

In [230…
```
mydict1
```

Out[230…
```
{'Name': 'Asif', 'DOB': 1995}
```

In [234…
```
mydict1.clear() # Delete all items of the dictionary using clear method
mydict1
```

Out[234…
```
{}
```

In [239…
```
del mydict1 # Delete the dictionary object
mydict1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[239], line 1
----> 1 del mydict1 # Delete the dictionary object
      2 mydict1

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

# Copy Dictionary

In [242…
```
mydict = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
mydict
```

Out[242…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [244…
```
mydict1 = mydict # Create a new reference "mydict1"
```

In [246…
```
id(mydict) , id(mydict1) # The address of both mydict & mydict1 will be the same
```

Out[246…
```
(2477306145856, 2477306145856)
```

In [248…
```
mydict2 = mydict.copy() # Create a copy of the dictionary
```

In [252…
```
id(mydict2) # The address of mydict2 will be different from mydict because mydic
```

Out[252…
```
2477360564672
```

In [254…
```
mydict['Address'] = 'Mumbai'
mydict
```

Out[254…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}
```

In [258…
```
mydict1 # mydict1 will be also impacted as it is pointing to the same dictionary
```

Out[258…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}
```

In [256…
```
mydict2 # Copy of list won't be impacted due to the changes made in the original
```

Out[256…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

# Loop through a Dictionary

In [261…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki' ,
mydict1
```

Out[261…
```
{'Name': 'Asif',
 'ID': 12345,
 'DOB': 1991,
 'Address': 'Hilsinki',
 'Job': 'Analyst'}
```

In [265…
```
for i in mydict1:
    print(i , ':' , mydict1[i]) # Key & value pair
```

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

In [267…
```
for i in mydict1:
    print(mydict1[i]) # Dictionary items
```

```
Asif
12345
1991
Hilsinki
Analyst
```

# Dictionary Membership

In [270…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[270…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [272…
```
'Name' in mydict1 # Test if a key is in a dictionary or not.
```

Out[272…
```
True
```

In [274…
```
'Asif' in mydict1 # Membership test can be only done for keys.
```

Out[274…
```
False
```

In [276…
```
'ID' in mydict1
```

Out[276…
```
True
```

In [278…
```
'Address' in mydict1
```

Out[278…
```
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

# For dictionaries the all() function checks the keys, not the values.

The any() function returns:

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

In [281…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[281…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [283…
```
all(mydict1) # Will Return false if one key is false (Value 0)
```

Out[283…
```
True
```

In [306…
```
mytest1 = {0: 'Sam' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mytest1
```

Out[306…
```
{0: 'Sam', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [308…
```
all(mytest1) # Will Return false as one key is false (Value 0)
```
Out[308…
```
False
```

In [310…
```
any(mydict1)
```
Out[310…
```
True
```

In [312…
```
any(mytest1)
```
Out[312…
```
True
```

In [314…
```
mytest2 = { 0 : 0 }
mytest2
```
Out[314…
```
{0: 0}
```

In [316…
```
any(mytest2)
```
Out[316…
```
False
```

In [ ]: