List

1. List is an ordered sequence of items.
2. We can have different data types under a list. E.g we can have integer, float and string items in a same list

List Creation

In [1]:
```
list1 = [] # Empty List
```

In [2]:
```
print(type(list1))
```

```
<class 'list'>
```

In [3]:
```
list2 = [10,30,60] # List of integers numbers
```

In [4]:
```
list3 = [10.77,30.66,60.89] # List of float numbers
```

In [5]:
```
list4 = ['one','two' , "three"] # List of strings
```

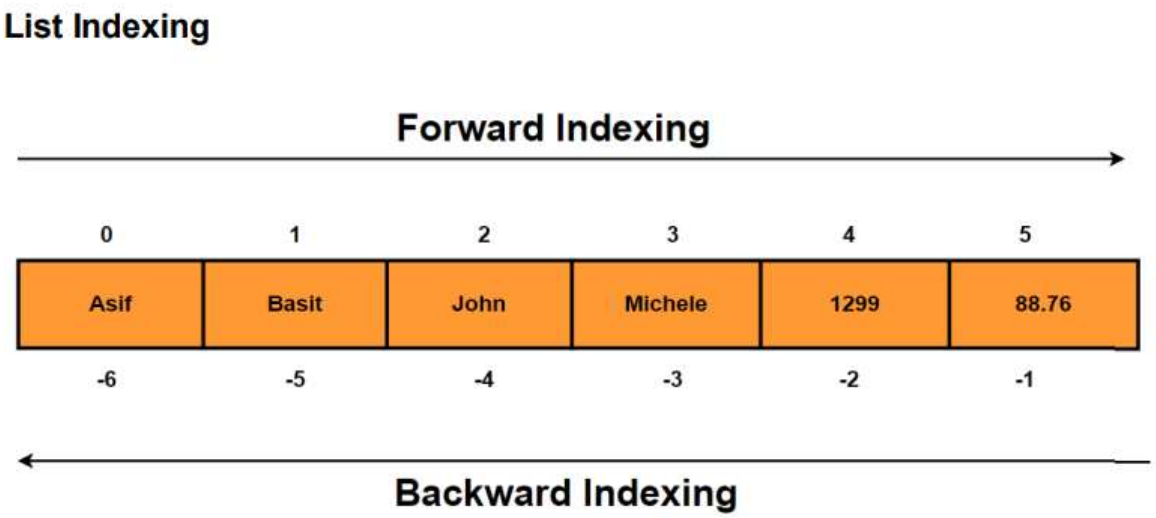
In [6]:
```
list5 = ['Asif', 25 ,[50, 100],[150, 90]] # Nested Lists
```

In [7]:
```
list6 = [100, 'Asif', 17.765] # List of mixed data types
```

In [8]:
```
list7 = ['Asif', 25 ,[50, 100],[150, 90] , {'John' , 'David'}]
```

In [9]:
```
len(list6) #Length of list
```

Out[9]:
```
3
```



In [10]:
```
list2[0] # Retreive first element of the list
```

Out[10]: 10

In [11]:
```
list4[0] # Retreive first element of the list
```

Out[11]: 'one'

In [12]:
```
list4[0][0] # Nested indexing - Access the first character of the first list elem
```

Out[12]: 'o'

In [13]:
```
list4[-1] # Last item of the list
```

Out[13]: 'three'

In [14]:
```
list5[-1] # Last item of the list
```

Out[14]: [150, 90]

List Slicing

In [15]:
```
mylist = ['one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight']
```

In [16]:
```
mylist[0:3] # Return all items from 0th to 3rd index location excluding the item
```

Out[16]: ['one', 'two', 'three']

In [17]:
```
mylist[2:5] # List all items from 2nd to 5th index location excluding the item at
```

Out[17]:
```
['three', 'four', 'five']
```

In [18]:
```
mylist[:3] # Return first three items
```

Out[18]:
```
['one', 'two', 'three']
```

In [19]:
```
mylist[:2] # Return first two items
```

Out[19]:
```
['one', 'two']
```

In [20]:
```
mylist[-3:] # Return last three items
```

Out[20]:
```
['six', 'seven', 'eight']
```

In [21]:
```
mylist[-2:] # Return last two items
```

Out[21]:
```
['seven', 'eight']
```

In [22]:
```
mylist[-1] # Return last item of the list
```

Out[22]:
```
'eight'
```

In [23]:
```
mylist[:] # Return whole list
```

Out[23]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

Add , Remove & Change Items

In [24]:
```
mylist
```

Out[24]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

In [25]:
```
mylist.append('nine') # Add an item to the end of the list
mylist
```

Out[25]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [26]:
```
mylist.insert(9,'ten') # Add item at index location 9
mylist
```

Out[26]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

In [27]:
```
mylist.insert(1,'ONE') # Add item at index location 1
mylist
```

Out[27]:
```
['one',
 'ONE',
 'two',
 'three',
 'four',
 'five',
 'six',
 'seven',
 'eight',
 'nine',
 'ten']
```

In [28]:
```
mylist.remove('ONE') # Remove item "ONE"
mylist
```

Out[28]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten']
```

In [29]:
```
mylist.pop() # Remove last item of the list
mylist
```

Out[29]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [30]:
```
mylist.pop(8) # Remove item at index location 8
mylist
```

Out[30]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

In [31]:
```
del mylist[7] # Remove item at index location 7
mylist
```

Out[31]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven']
```

In [32]:
```
# Change value of the string
mylist[0] = 1
mylist[1] = 2
mylist[2] = 3
mylist
```

Out[32]:
```
[1, 2, 3, 'four', 'five', 'six', 'seven']
```

In [33]:
```
mylist.clear() # Empty List / Delete all items in the list
mylist
```

Out[33]:
```
[]
```

In [34]:
```
del mylist # Delete the whole list
mylist
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[34], line 2
      1 del mylist # Delete the whole list
----> 2 mylist

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

Copy List

In [82]:
```
mylist = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [83]:
```
mylist1 = mylist # Create a new reference "mylist1"
```

In [84]:
```
id(mylist) , id(mylist1) # The address of both mylist & mylist1 will be the same
```

Out[84]:
```
(2938882911360, 2938882911360)
```

In [85]:
```
mylist2 = mylist.copy() # Create a copy of the list
```

In [86]:
```
id(mylist2) # The address of mylist2 will be different from mylist because mylist
```

Out[86]:
```
2938882940352
```

In [87]:
```
mylist[0] = 1
```

In [88]:
```
mylist
```

Out[88]:
```
[1, 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [89]:
```
mylist1 # mylist1 will be also impacted as it is pointing to the same list
```

Out[89]:
```
[1, 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

In [90]:
```
mylist2 # Copy of list won't be impacted due to changes made on the original list
```

Out[90]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']
```

Join Lists

In [42]:
```
list1 = ['one', 'two', 'three', 'four']
list2 = ['five', 'six', 'seven', 'eight']
```

In [43]:
```
list3 = list1 + list2 # Join two lists by '+' operator
list3
```

Out[43]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

In [44]:
```
list1.extend(list2) #Append list2 with list1
list1
```

Out[44]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

In [45]:
```
list1
```

Out[45]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

In [46]:
```
'one' in list1 # Check if 'one' exist in the list
```

Out[46]:
```
True
```

In [47]:
```
'ten' in list1 # Check if 'ten' exist in the list
```

Out[47]:
```
False
```

In [48]:
```
if 'three' in list1: # Check if 'three' exist in the list
    print('Three is present in the list')
else:
    print('Three is not present in the list')
```

```
Three is present in the list
```

In [49]:
```
if 'eleven' in list1: # Check if 'eleven' exist in the list
    print('eleven is present in the list')
else:
    print('eleven is not present in the list')
```

```
eleven is not present in the list
```

Reverse & Sort List

In [50]:
```
list1
```

Out[50]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

In [51]:
```
list1.reverse() # Reverse the list
list1
```

Out[51]:
```
['eight', 'seven', 'six', 'five', 'four', 'three', 'two', 'one']
```

In [52]:
```
list1 = list1[::-1] # Reverse the list
list1
```

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

In [53]:
```
mylist3 = [9,5,2,99,12,88,34]
mylist3.sort() # Sort list in ascending order
mylist3
```

Out[53]:
```
[2, 5, 9, 12, 34, 88, 99]
```

In [54]:
```
mylist3 = [9,5,2,99,12,88,34]
mylist3.sort(reverse=True) # Sort list in descending order
mylist3
```

Out[54]:
```
[99, 88, 34, 12, 9, 5, 2]
```

Loop through a list

In [56]:
```
list1
```

Out[56]:
```
['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']
```

In [57]:
```
for i in list1:
    print(i)
```

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

In [58]:
```
for i in enumerate(list1):
    print(i)
```

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

Count

In [59]:
```
list10 =['one', 'two', 'three', 'four', 'one', 'one', 'two', 'three']
```

In [60]:
```
list10.count('one') # Number of times item "one" occurred in the list.
```

Out[60]: 3

In [61]:
```
list10.count('two') # Occurence of item 'two' in the list
```

Out[61]: 2

In [62]:
```
list10.count('four') #Occurence of item 'four' in the list
```

Out[62]: 1

```
All / Any
The all() method returns:
True - If all elements in a list are true
False - If any element in a list is false
The any() function returns True if any element in the list is True. If not, any()
returns False.
```

In [63]:
```
L1 = [1,2,3,4,0]
```

In [64]:
```
all(L1) # Will Return false as one value is false (Value 0)
```

Out[64]: False

In [65]:
```
any(L1) # Will Return True as we have items in the list with True value
```

Out[65]:
```
True
```

In [66]:
```
L2 = [1,2,3,4,True,False]
```

In [67]:
```
all(L2) # Returns false as one value is false
```

Out[67]:
```
False
```

In [68]:
```
any(L2) # Will Return True as we have items in the list with True value
```

Out[68]:
```
True
```

In [69]:
```
L3 = [1,2,3,True]
```

In [70]:
```
all(L3) # Will return True as all items in the list are True
```

Out[70]:
```
True
```

In [71]:
```
any(L3) # Will Return True as we have items in the list with True value
```

Out[71]:
```
True
```

List Comprehensions

List Comprehensions provide an elegant way to create new lists. It consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses.

In [72]:
```
mystring = "WELCOME"
mylist = [ i for i in mystring ] # Iterating through a string Using List Comprehe
mylist
```

Out[72]:
```
['W', 'E', 'L', 'C', 'O', 'M', 'E']
```

In [73]:
```
mylist1 = [ i for i in range(40) if i % 2 == 0] # Display all even numbers betwee
mylist1
```

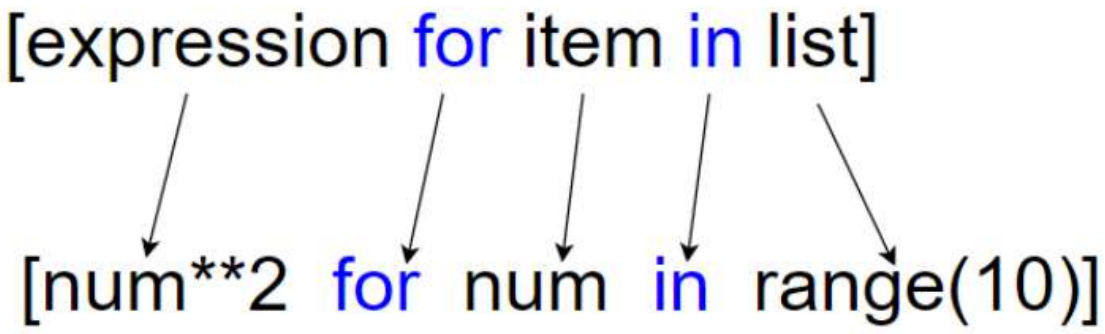
Out[73]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38]
```

In [74]:
```
mylist2 = [ i for i in range(40) if i % 2 == 1] # Display all odd numbers between
mylist2
```

Out[74]: [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39]

In [75]:
```
mylist3 = [num**2 for num in range(10)] # calculate square of all numbers between
mylist3
```

Out[75]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]



In [76]:
```
# Multiple whole list by 10
list1 = [2,3,4,5,6,7,8]
list1 = [i*10 for i in list1]
list1
```

Out[76]: [20, 30, 40, 50, 60, 70, 80]

In [77]:
```
#List all numbers divisible by 3 , 9 & 12 using nested "if" with List Comprehensi
mylist4 = [i for i in range(200) if i % 3 == 0 if i % 9 == 0 if i % 12 == 0]
mylist4
```

Out[77]: [0, 36, 72, 108, 144, 180]

In [79]:
```
# Extract numbers from a string
mystr = "One 1 two 2 three 3 four 4 five 5 six 6789"
numbers = [i for i in mystr if i.isdigit()]
numbers
```

Out[79]: ['1', '2', '3', '4', '5', '6', '7', '8', '9']

In [80]:
```
# Extract letters from a string
mystr = "One 1 two 2 three 3 four 4 five 5 six 6789"
numbers = [i for i in mystr if i.isalpha()]
numbers
```

Out[80]:
```
['O',
 'n',
 'e',
 't',
 'w',
 'o',
 't',
 'h',
 'r',
 'e',
 'e',
 'f',
 'o',
 'u',
 'r',
 'f',
 'i',
 'v',
 'e',
 's',
 'i',
 'x']
```

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 [94]:
```
tup1 = () # Empty tuple
```

In [95]:
```
tup2 = (10,30,60) # tuple of integers numbers
```

In [96]:
```
tup3 = (10.77,30.66,60.89) # tuple of float numbers
```

In [97]:
```
tup4 = ('one','two' , "three") # tuple of strings
```

In [98]:
```
tup5 = ('Asif', 25 ,(50, 100),(150, 90)) # Nested tuples
```

In [99]:
```
tup6 = (100, 'Asif', 17.765) # Tuple of mixed data types
```

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

In [101]:
```
len(tup7) #Length of list
```

Out[101]: 6

Tuple Indexing

In [102]:
```
tup2[0] # Retreive first element of the tuple
```

Out[102]: 10

In [103]:
```
tup4[0] # Retreive first element of the tuple
```

Out[103]: 'one'

In [104]:
```
tup4[0][0] # Nested indexing - Access the first character of the first tuple elem
```

Out[104]: 'o'

In [105]:
```
tup4[-1] # Last item of the tuple
```

Out[105]: 'three'

In [106]:
```
tup5[-1] # Last item of the tuple
```

Out[106]: (150, 90)

Tuple Slicing

In [107]:
```
mytuple = ('one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight')
```

In [108]:
```
mytuple[0:3] # Return all items from 0th to 3rd index location excluding the item
```

Out[108]: ('one', 'two', 'three')

In [109]:
```
mytuple[2:5] # List all items from 2nd to 5th index location excluding the item a
```

Out[109]: ('three', 'four', 'five')

In [110]:
```
mytuple[:3] # Return first three items
```

Out[110]: ('one', 'two', 'three')

In [111]:
```
mytuple[:2] # Return first two items
```

Out[111]: ('one', 'two')

In [112]:
```
mytuple[-3:] # Return last three items
```

Out[112]: ('six', 'seven', 'eight')

In [113]:
```
mytuple[-2:] # Return last two items
```

Out[113]: ('seven', 'eight')

In [114]:
```
mytuple[-1] # Return last item of the tuple
```

Out[114]: 'eight'

In [115]:
```
mytuple[:] # Return whole tuple
```

Out[115]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

Remove & Change Items

In [116]:
```
mytuple
```

Out[116]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

In [117]:
```
del mytuple[0] # Tuples are immutable which means we can't DELETE tuple items
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[117], line 1
----> 1 del mytuple[0]

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

In [118]:
```
mytuple[0] = 1 # Tuples are immutable which means we can't CHANGE tuple items
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[118], line 1
----> 1 mytuple[0] = 1

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

In [122]:
```
del mytuple # Deleting entire tuple object is possible
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[122], line 1
----> 1 del mytuple

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

Loop through a tuple

In [123]:
```
mytuple = ('one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight')
```

In [124]:
```
for i in mytuple:
    print(i)
```

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

In [125]:
```
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 [126]:
```
mytuple1 =('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')
```

In [127]:
```
mytuple1.count('one') # Number of times item "one" occurred in the tuple.
```

Out[127]: 3

In [128]:
```
mytuple1.count('two') # Occurence of item 'two' in the tuple
```

Out[128]: 2

In [129]:
```
mytuple1.count('four') #Occurence of item 'four' in the tuple
```

Out[129]: 1

Tuple Membership

In [130]:
```
mytuple
```

Out[130]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

In [131]:
```
'one' in mytuple # Check if 'one' exist in the list
```

Out[131]: True

In [132]:
```
'ten' in mytuple # Check if 'ten' exist in the list
```

Out[132]: False

In [133]:
```
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 [134]:
```
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 [135]:
```
mytuple.index('one') # Index of first element equal to 'one'
```

Out[135]:
```
0
```

In [136]:
```
mytuple.index('five') # Index of first element equal to 'five'
```

Out[136]:
```
4
```

In [137]:
```
mytuple1
```

Out[137]:
```
('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')
```

In [138]:
```
mytuple1.index('one') # Index of first element equal to 'one'
```

Out[138]:
```
0
```

Sorting

In [139]:
```
mytuple2 = (43,67,99,12,6,90,67)
```

In [140]:
```
sorted(mytuple2) # Returns a new sorted list and doesn't change original tuple
```

Out[140]:
```
[6, 12, 43, 67, 67, 90, 99]
```

In [141]:
```
sorted(mytuple2, reverse=True) # Sort in descending order
```

Out[141]:
```
[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.

In [142]:
```
myset = {1,2,3,4,5} # Set of numbers
myset
```

Out[142]:
```
{1, 2, 3, 4, 5}
```

In [143]:
```
my_set = {1,1,2,2,3,4,5,5}
my_set
```

Out[143]:
```
{1, 2, 3, 4, 5}
```

In [144]:
```
myset1 = {1.79,2.08,3.99,4.56,5.45} # Set of float numbers
myset1
```

Out[144]:
```
{1.79, 2.08, 3.99, 4.56, 5.45}
```

In [145]:
```
myset2 = {'Asif' , 'John' , 'Tyrion'} # Set of Strings
myset2
```

Out[145]:
```
{'Asif', 'John', 'Tyrion'}
```

In [146]:
```
myset3 = {10,20, "Hola", [11, 22, 32]} # set doesn't allow mutable items like lis
myset3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[146], line 1
----> 1 myset3 = {10,20, "Hola", [11, 22, 32]} # set doesn't allow mutable items
like lis
      2 myset3

TypeError: unhashable type: 'list'
```

In [147]:
```
myset4 = set() # Create an empty set
print(type(myset4))
```

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

In [148]:
```
my_set1 = set(('one' , 'two' , 'three' , 'four'))
my_set1
```

Out[148]:
```
{'four', 'one', 'three', 'two'}
```

Loop through a Set

In [149]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
for i in myset:
    print(i)
```

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

In [150]:
```
for i in enumerate(myset):
    print(i)
```

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

Set Membership

In [151]:
```
myset
```

Out[151]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [152]:
```
'one' in myset # Check if 'one' exist in the set
```

Out[152]:
```
True
```

In [153]:
```
'ten' in myset # Check if 'ten' exist in the set
```

Out[153]:
```
False
```

In [154]:
```
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 [155]:
```
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 [156]:
```
myset
```

Out[156]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [157]:
```
myset.add('NINE') # Add item to a set using add() method
myset
```

Out[157]:
```
{'NINE', 'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [158]:
```
myset.update(['TEN' , 'ELEVEN' , 'TWELVE']) # Add multiple item to a set using u
myset
```

Out[158]:
```
{'ELEVEN',
 'NINE',
 'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [159]:
```
myset.remove('NINE') # remove item in a set using remove() method
myset
```

Out[159]:
```
{'ELEVEN',
 'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [160]:
```
myset.discard('TEN') # remove item from a set using discard() method
myset
```

Out[160]:
```
{'ELEVEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [161]:
```
myset.clear() # Delete all items in a set
myset
```

Out[161]:
```
set()
```

In [162]:
```
del myset # Delete the set object
myset
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[162], line 2
      1 del myset # Delete the set object
----> 2 myset

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

Copy Set

In [163]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
myset
```

Out[163]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [164]:
```
myset1 = myset # Create a new reference "myset1"
myset1
```

Out[164]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [165]:
```
id(myset) , id(myset1) # The address of both myset & myset1 will be the same as
```

Out[165]:
```
(2938881877920, 2938881877920)
```

In [166]:
```
my_set = myset.copy() # Create a copy of the list
my_set
```

Out[166]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [167]:
```
id(my_set) # The address of my_set will be different from myset because my_set is
```

Out[167]:
```
2938881876800
```

In [168]:
```
myset.add('nine')
myset
```

Out[168]:
```
{'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

In [169]:
```
myset1 # myset1 will be also impacted as it is pointing to the same Set
```

Out[169]:
```
{'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

In [170]:
```
my_set # Copy of the set won't be impacted due to changes made on the original Se
```

Out[170]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

Set Operation

Union

In [171]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
C = {8,9,10}
```

In [172]:
```
A | B # Union of A and B (All elements from both sets. NO DUPLICATES)
```

Out[172]: {1, 2, 3, 4, 5, 6, 7, 8}

In [173]:
```
A.union(B) # Union of A and B
```

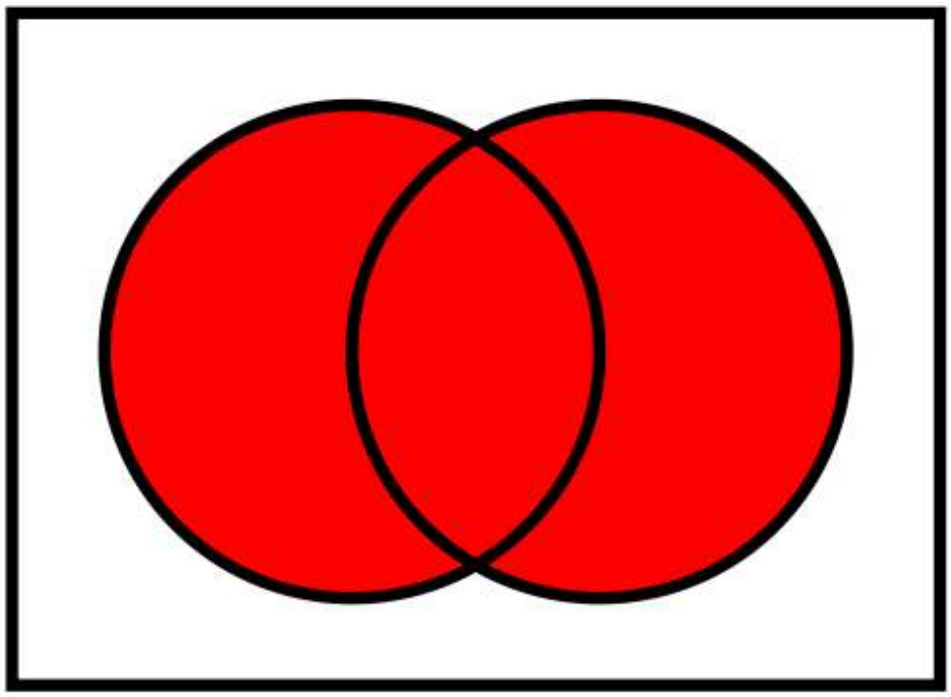
Out[173]: {1, 2, 3, 4, 5, 6, 7, 8}

In [174]:
```
A.union(B, C) # Union of A, B and C.
```

Out[174]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

In [175]:
```
"""
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[175]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}



Intersection

In [176]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [177]:
```
A & B # Intersection of A and B (Common items in both sets)
```

Out[177]:
```
{4, 5}
```

In [178]:
```
A.intersection(B) Intersection of A and B
```

```
  Cell In[178], line 1
    A.intersection(B) Intersection of A and B
                      ^
SyntaxError: invalid syntax
```

In [179]:
```
"""
Updates the set calling the intersection_update() method with the intersection of
For below example Set A will be updated with the intersection of A & B.
"""
A.intersection_update(B)
A
```

Out[179]:
```
{4, 5}
```

Difference

In [180]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [181]:
```
A - B # set of elements that are only in A but not in B
```

Out[181]:
```
{1, 2, 3}
```

In [182]:
```
A.difference(B) # Difference of sets
```

Out[182]:
```
{1, 2, 3}
```

In [183]:
```
B- A # set of elements that are only in B but not in A
```

Out[183]: {6, 7, 8}

In [184]:
```
B.difference(A)
```

Out[184]: {6, 7, 8}

In [186]:
```
"""
Updates the set calling the difference_update() method with the difference of set
For below example Set B will be updated with the difference of B & A.
"""
B.difference_update(A)
B
```

Out[186]: {6, 7, 8}

Symmetric Difference

In [187]:
```
A ^ B # Symmetric difference (Set of elements in A and B but not in both. "EXCLUD
```

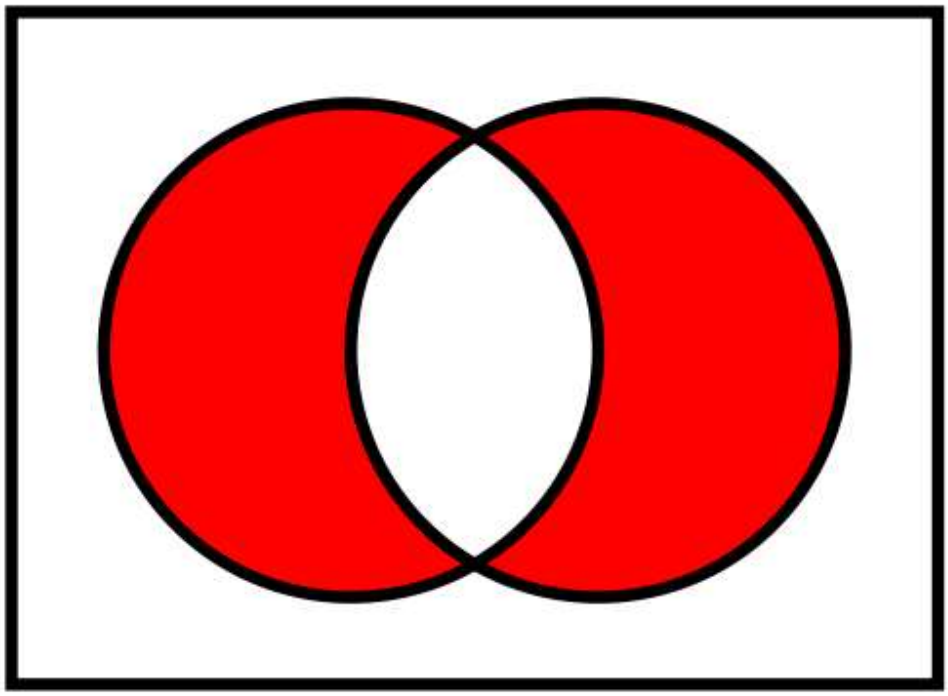
Out[187]: {1, 2, 3, 4, 5, 6, 7, 8}

In [188]:
```
A.symmetric_difference(B) # Symmetric difference of sets
```

Out[188]: {1, 2, 3, 4, 5, 6, 7, 8}

In [189]:
```
"""
Updates the set calling the symmetric_difference_update() method with the symmetr
For below example Set A will be updated with the symmetric difference of A & B.
"""
A.symmetric_difference_update(B)
A
```

Out[189]: {1, 2, 3, 4, 5, 6, 7, 8}



Subset , Superset & Disjoint

In [190]:
```
A = {1,2,3,4,5,6,7,8,9}
B = {3,4,5,6,7,8}
C = {10,20,30,40}
```

In [191]:
```
B.issubset(A) # Set B is said to be the subset of set A if all elements of B are
```

Out[191]: True

In [192]:
```
A.issuperset(B) # Set A is said to be the superset of set B if all elements of B
```

Out[192]: True

In [193]:
```
C.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common el
```

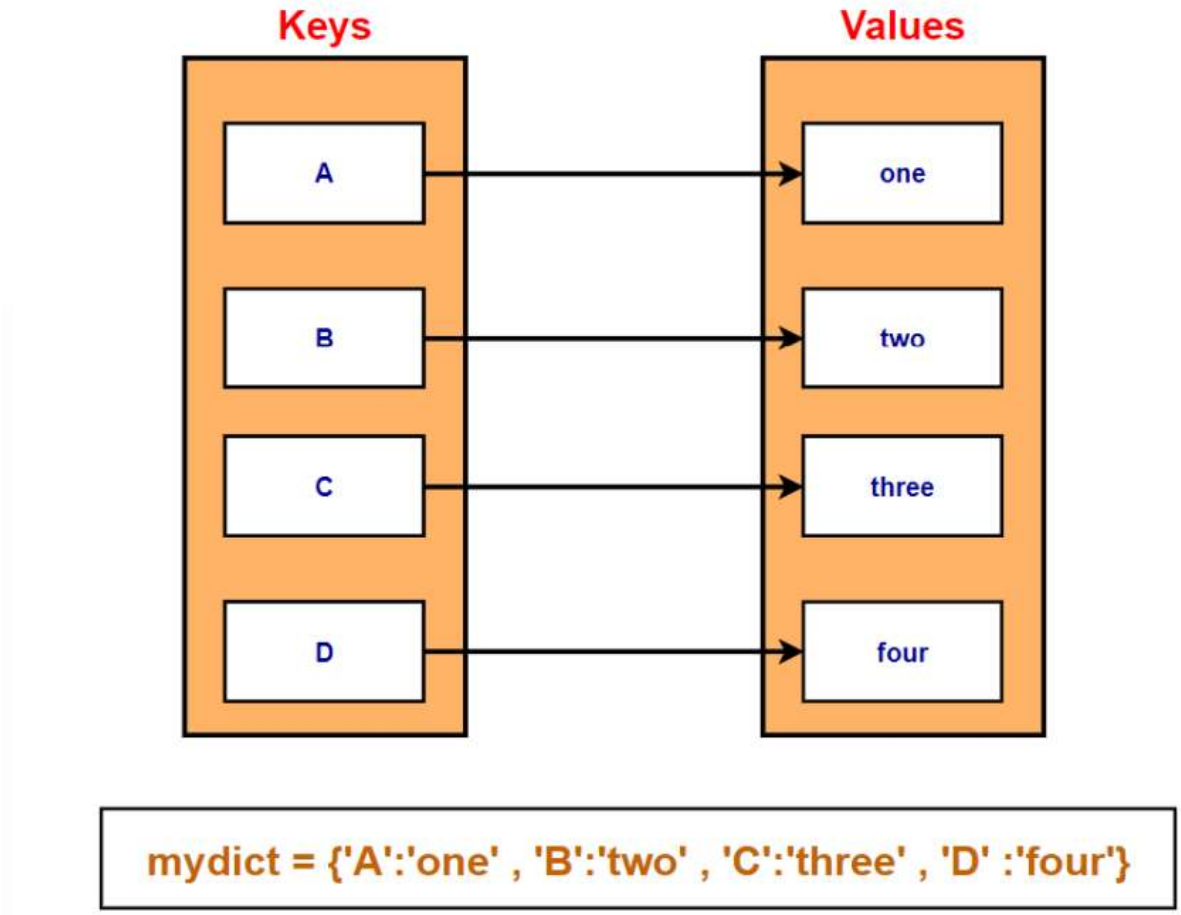
Out[193]: True

In [194]:
```
B.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common el
```

Out[194]: False

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.



Create Dictionary

In [2]:
```
mydict = dict() # empty dictionary
mydict
```

Out[2]: {}

In [3]:
```
mydict = {} # empty dictionary
mydict
```

Out[3]: {}

In [4]:
```
mydict = {1:'one' , 2:'two' , 3:'three'} # dictionary with integer keys
mydict
```

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

In [5]:
```
mydict = dict({1:'one' , 2:'two' , 3:'three'}) # Create dictionary using dict()
mydict
```

Out[5]:
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [6]:
```
mydict = {'A':'one' , 'B':'two' , 'C':'three'} # dictionary with character keys
mydict
```

Out[6]:
```
{'A': 'one', 'B': 'two', 'C': 'three'}
```

In [7]:
```
mydict = {1:'one' , 'A':'two' , 3:'three'} # dictionary with mixed keys
mydict
```

Out[7]:
```
{1: 'one', 'A': 'two', 3: 'three'}
```

In [8]:
```
mydict.keys() # Return Dictionary Keys using keys() method
```

Out[8]:
```
dict_keys([1, 'A', 3])
```

In [9]:
```
mydict.values() # Return Dictionary Values using values() method
```

Out[9]:
```
dict_values(['one', 'two', 'three'])
```

In [10]:
```
mydict.items() # Access each key-value pair within a dictionary
```

Out[10]:
```
dict_items([(1, 'one'), ('A', 'two'), (3, 'three')])
```

In [11]:
```
mydict = {1:'one' , 2:'two' , 'A':['asif' , 'john' , 'Maria']} # dictionary with
mydict
```

Out[11]:
```
{1: 'one', 2: 'two', 'A': ['asif', 'john', 'Maria']}
```

In [14]:
```
mydict = {1:'one' , 2:'two' , 'A':['asif' , 'john' , 'Maria'], 'B':('Bat' , 'cat')
mydict
```

Out[14]:
```
{1: 'one', 2: 'two', 'A': ['asif', 'john', 'Maria'], 'B': ('Bat', 'cat')}
```

In [16]:
```
mydict = {1:'one' , 2:'two' , 'A':{'Name':'asif' , 'Age' :20}, 'B':('Bat' , 'cat')
mydict
```

Out[16]:
```
{1: 'one', 2: 'two', 'A': {'Name': 'asif', 'Age': 20}, 'B': ('Bat', 'cat')}
```

In [17]:
```
keys = {'a' , 'b' , 'c' , 'd'}
mydict3 = dict.fromkeys(keys) # Create a dictionary from a sequence of keys
mydict3
```

Out[17]:
```
{'d': None, 'a': None, 'c': None, 'b': None}
```

In [18]:
```
keys = {'a' , 'b' , 'c' , 'd'}
value = 10
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of k
mydict3
```

Out[18]:
```
{'d': 10, 'a': 10, 'c': 10, 'b': 10}
```

In [19]:
```
keys = {'a' , 'b' , 'c' , 'd'}
value = [10,20,30]
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of k
mydict3
```

Out[19]:
```
{'d': [10, 20, 30], 'a': [10, 20, 30], 'c': [10, 20, 30], 'b': [10, 20, 30]}
```

In [20]:
```
value.append(40)
mydict3
```

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

Accessing Items

In [21]:
```
mydict = {1:'one' , 2:'two' , 3:'three' , 4:'four'}
mydict
```

Out[21]:
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [22]:
```
mydict[1] # Access item using key
```

Out[22]:
```
'one'
```

In [23]:
```
mydict.get(1) # Access item using get() method
```

Out[23]:
```
'one'
```

In [24]:
```
mydict1 = {'Name':'Asif' , 'ID': 74123 , 'DOB': 1991 , 'job' :'Analyst'}
mydict1
```

Out[24]:
```
{'Name': 'Asif', 'ID': 74123, 'DOB': 1991, 'job': 'Analyst'}
```

In [25]:
```
mydict1['Name'] # Access item using key
```

Out[25]:
```
'Asif'
```

In [26]:
```
mydict1.get('job') # Access item using get() method
```

Out[26]:
```
'Analyst'
```

Add, Remove & Change Items

In [27]:
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
mydict1
```

Out[27]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [28]:
```
mydict1['DOB'] = 1992 # Changing Dictionary Items
mydict1['Address'] = 'Delhi'
mydict1
```

Out[28]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1992, 'Address': 'Delhi'}
```

In [29]:
```
dict1 = {'DOB':1995}
mydict1.update(dict1)
mydict1
```

Out[29]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}
```

In [30]:
```
mydict1['Job'] = 'Analyst' # Adding items in the dictionary
mydict1
```

Out[30]:
```
{'Name': 'Asif',
 'ID': 12345,
 'DOB': 1995,
 'Address': 'Delhi',
 'Job': 'Analyst'}
```

In [31]:
```
mydict1.pop('Job') # Removing items in the dictionary using Pop method
mydict1
```

Out[31]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}
```

In [32]:
```
mydict1.popitem() # A random item is removed
```

Out[32]:
```
('Address', 'Delhi')
```

In [33]:
```
mydict1
```

Out[33]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995}
```

In [34]:
```
del[mydict1['ID']] # Removing item using del method
mydict1
```

Out[34]:
```
{'Name': 'Asif', 'DOB': 1995}
```

In [35]:
```
mydict1.clear() # Delete all items of the dictionary using clear method
mydict1
```

Out[35]:
```
{}
```

In [36]:
```
del mydict1 # Delete the dictionary object
mydict1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[36], line 2
      1 del mydict1 # Delete the dictionary object
----> 2 mydict1

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

Copy Dictionary

In [37]:
```
mydict = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
mydict
```

Out[37]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [38]:
```
mydict1 = mydict # Create a new reference "mydict1"
```

In [39]:
```
id(mydict) , id(mydict1) # The address of both mydict & mydict1 will be the same
```

Out[39]:
```
(1909215729280, 1909215729280)
```

In [40]:
```
mydict2 = mydict.copy() # Create a copy of the dictionary
```

In [41]:
```
id(mydict2) # The address of mydict2 will be different from mydict because mydict
```

Out[41]:
```
1909215061824
```

In [42]:
```
mydict
```

Out[42]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [43]:
```
mydict1 # mydict1 will be also impacted as it is pointing to the same dictionary
```

Out[43]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [44]:
```
mydict2 # Copy of list won't be impacted due to the changes made in the original
```

Out[44]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

Loop through a Dictionary

In [47]:
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki' }
mydict1
```

Out[47]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [48]:
```
for i in mydict1:
    print(i , ':' , mydict1[i]) # Key & value pair
```

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

In [49]:
```
for i in mydict1:
    print(mydict1[i]) # Dictionary items
```

```
Asif
12345
1991
Hilsinki
```

Dictionary Membership

In [50]:
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[50]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [51]:
```
'Name' in mydict1 # Test if a key is in a dictionary or not.
```

Out[51]:
```
True
```

In [52]:
```
'Asif' in mydict1 # Membership test can be only done for keys.
```

Out[52]:
```
False
```

In [53]:
```
'ID' in mydict1
```

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

In [54]:
```
'Address' in mydict1
```

Out[54]:
```
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 [55]:
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[55]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [56]:
```
any(mydict1) # Will Return True as we have items in the dictionary with True val
```

Out[56]:
```
True
```

In [57]:
```
mydict1[0] = 'test1'
mydict1
```

Out[57]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst', 0: 'test1'}
```

In [58]:
```
all(mydict1) # Returns false as one value is false
```

Out[58]:
```
False
```

In [59]:
```
any(mydict1) # Will Return True as we have items in the dictionary with True val
```

Out[59]:
```
True
```

In [ ]: