Python Data Structure

1. Tuples

1. Tuple is similiar to list expect that the object in tuple are immutablewich means we can not change the eliment of the tupple once assigned.
2. when we do not want to change the data over time, tuple is prefered data type.
3. Itreting over the elements of a tuple is faster compaired to itreting over a list.

Tuple Creation

In [1]:
```
t=(1,2,3,4)
t
```
Out[1]:
```
(1, 2, 3, 4)
```

In [2]:
```
tup1 = ()
```

In [3]:
```
tup2 = (10,30,60)
```

In [4]:
```
tup3 = (10.77,30.66,60.89) # tuple of float numbers
```

In [5]:
```
tup4 = ('one','two', "three") # tuple of string
```

In [6]:
```
tup5 = ('ASSif',25, (50,100),(150,90)) # nested tupple
```

In [7]:
```
tup6 = (100, 'Assif', 17.765) # mixed data types
```

In [8]:
```
tup7 = ('Asif', 25 ,[50, 100],[150, 90] , {'John' , 'David'} , (99,22,33))
```

In [9]:
```
len(tup7) #length of list
```
Out[9]:
```
6
```

Tuples indexing

In [10]:
```
tup2[0]
```
Out[10]:
```
10
```

In [11]:
```
tup4[0]
```
Out[11]:
```
'one'
```

In [12]:
```
tup4 [0][0]
```
Out[12]:
```
'o'
```

```
In [13]: tup4 [-1]
```

```
Out[13]: 'three'
```

```
In [14]: tup5 [-1]
```

```
Out[14]: (150, 90)
```

tuple slicing

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

```
In [16]: mytuple[0:3]
```

```
Out[16]: ('one', 'two', 'three')
```

```
In [17]: mytuple[2:9]
```

```
Out[17]: ('three', 'four', 'five', 'six', 'seven', 'eight')
```

```
In [18]: mytuple[:3]
```

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

```
In [19]: mytuple[:2]
```

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

```
In [20]: mytuple[-3:]
```

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

```
In [21]: mytuple[:-3]
```

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

```
In [22]: mytuple[-2:]
```

```
Out[22]: ('seven', 'eight')
```

```
In [23]: mytuple[:]
```

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

```
In [24]: mytuple[-1]
```

```
Out[24]: 'eight'
```

Remove & Change item

```
In [25]: mytuple
```

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

```
In [26]:  del mytuple[0] # Tuples are immutable which means we can't DELETE tuple items
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 del mytuple[0]

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

```
In [27]:  mytuple[0] = 1 # Tuples are immutable which means we can't CHANGE tuple items
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[27], line 1
----> 1 mytuple[0] = 1

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

Loop through a tuple

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

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

```
In [29]:  for i in mytuple:
              print(i)
```

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

```
In [30]:  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 [31]:  mytuple1 = ('one', 'two', 'three', 'four','one', 'one', 'two', 'three')
```

```
In [32]:  mytuple1.count('one')
```

```
Out[32]: 3

In [33]: mytuple1.count('two')

Out[33]: 2

In [34]: mytuple1.count('three')

Out[34]: 2

In [35]: mytuple1.count('four')

Out[35]: 1
```

Tuple membership

```
In [36]: mytuple

Out[36]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

In [37]: 'one' in mytuple

Out[37]: True

In [38]: 'ten' in mytuple

Out[38]: False

In [39]: mytuple

Out[39]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

In [40]: if 'three' in mytuple:
             print('three is present in mytuple')
         else:
             print('three is not present in mytuple')

         three is present in mytuple

In [41]: if 'eleven' in mytuple:
             print('eleven is present in mytuple')
         else:
             print('eleven is not present in mytuple')

         eleven is not present in mytuple
```

Index Position

```
In [42]: mytuple

Out[42]: ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

In [43]: mytuple.index('one')
```

```
Out[43]:  0

In [44]:  mytuple.index('five')

Out[44]:  4

In [45]:  mytuple1

Out[45]:  ('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')

In [46]:  mytuple1.index('one')

Out[46]:  0

In [47]:  mytuple1.index('three')

Out[47]:  2
```

Sorting

```
In [48]:  mytuple2 = (43,67,99,12,6,90,67)

In [49]:  sorted(mytuple2)  # Returns a new sorted list and doesn't change original tuple

Out[49]:  [6, 12, 43, 67, 67, 90, 99]

In [50]:  sorted(mytuple2, reverse=True)

Out[50]:  [99, 90, 67, 67, 43, 12, 6]
```

2. Sets

1. Unordered & Unindexed collection of item.
2. Set elements are unique. Duplicate element are not allowed.
3. Set Element are immutable.(cannot be changed.)
4. Set itself is mutable. We can add or remove item form it.

Set creation

```
In [51]:  myset = {1,2,3,4,5}
          myset

Out[51]:  {1, 2, 3, 4, 5}

In [52]:  len(myset)

Out[52]:  5

In [53]:  my_set = {1,1,2,2,3,3,4,4,5,5}
          my_set                          # Duplicate elements are not allowed.
```

```
Out[53]: {1, 2, 3, 4, 5}
```

```
In [54]: myset1 = {1.79,2.08,3.99,4.56,5.45} # set of float numbers
         myset1
```

```
Out[54]: {1.79, 2.08, 3.99, 4.56, 5.45}
```

```
In [55]: myset2 = {'Assif','Johan','Tyrion'} # set of string
         myset2
```

```
Out[55]: {'Assif', 'Johan', 'Tyrion'}
```

```
In [56]: myset3 = {10,20,"Hola",(11,22,32)} # mixed datatypes
         myset3
```

```
Out[56]: {(11, 22, 32), 10, 20, 'Hola'}
```

```
In [57]: myset3 = {10,20,"Hola",[11,22,32]} # set dosen't allow mutable items like
         myset3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[57], line 1
----> 1 myset3 = {10,20,"Hola",[11,22,32]} # set dosen't allow mutable items like
      2 myset3

TypeError: unhashable type: 'list'
```

```
In [58]: myset4 = set() #create on empty set
         print(type(myset4))
```

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

```
In [59]: my_set1 = set (('one', 'two', 'three', 'four'))
         my_set1
```

```
Out[59]: {'four', 'one', 'three', 'two'}
```

Loop through a Set

```
In [60]: myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}

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

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

```
In [61]: for i in enumerate(myset):
             print(i)
```

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

Set Membership

```
In [62]: myset
```

```
Out[62]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [63]: 'one' in myset # check if 'one' exit in set
```

```
Out[63]: True
```

```
In [64]: 'ten' in myset # check if 'one' exit in set
```

```
Out[64]: False
```

```
In [65]: if 'three' in myset:
             print('three is present in myset')
         else:
             print('three is not present in myset')
```

```
three is present in myset
```

```
In [66]: if 'eleven' in myset:
             print('eleven is present in myset')
         else:
             print('eleven is not present in myset')
```

```
eleven is not present in myset
```

```
In [67]: if 'seven' in myset:
             print('seven is present in myset')
         else:
             print('seven is not present in myset')
```

```
seven is present in myset
```

Add & Remove Items

```
In [68]: myset
```

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

```
In [69]: myset.add('Nine') # add item using set add() method
         myset
```

```
Out[69]: {'Nine', 'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [70]: myset.update(['Ten','Eleven', 'Twelve']) #add multiple items set using update()
```

```
myset
```

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

```
In [71]: myset.remove('Nine') # remove the item set using remove() method
         myset
```

```
Out[71]: {'Eleven',
           'Ten',
           'Twelve',
           'eight',
           'five',
           'four',
           'one',
           'seven',
           'six',
           'three',
           'two'}
```

```
In [72]: myset.discard('Ten') # remove the item form a set using discard() method
         myset
```

```
Out[72]: {'Eleven',
           'Twelve',
           'eight',
           'five',
           'four',
           'one',
           'seven',
           'six',
           'three',
           'two'}
```

```
In [73]: myset.clear() # delete the all item in set
         myset
```

```
Out[73]: set()
```

```
In [74]: del myset # Delete the set object
         myset
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[74], line 2
      1 del myset # Delete the set object
----> 2 myset

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

Copy set

```
In [75]: myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
         myset
```

```
Out[75]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [76]: myset1 = myset #create a new reference "myset1"
         myset1
```

```
Out[76]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [77]: id(myset), id(myset1) # address of the myset & myset1 in both as same
```

```
Out[77]: (2327843038016, 2327843038016)
```

```
In [78]: my_set = myset.copy #create the coppy of the list
         my_set
```

```
Out[78]: <function set.copy>
```

```
In [79]: id(my_set) # address of the my_set will be different of myset because
```

```
Out[79]: 2327843900608
```

```
In [80]: myset.add('nine')
         myset
```

```
Out[80]: {'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [81]: myset1 # myset1 is also impacted as it is pointing a same set
```

```
Out[81]: {'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [82]: my_set # copy of the cell won't be impacted due to chenges on orignal set
```

```
Out[82]: <function set.copy>
```

Set Opreation

Union

```
In [83]: A = {1,2,3,4,5}
         B ={4,5,6,7,8}
         C ={8,9,10}
```

```
In [84]: A | B # union of A & B (all elements are both sets. no duplicate)
```

```
Out[84]: {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [85]: A.union(B) # union A & B
```

```
Out[85]: {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [86]: A.union(B,C) # union A,B, & C
```

Out[86]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [87]:
```
A | B | C
```

Out[87]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [88]:
```
"""
Updates the set calling the update() method with union 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 [89]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [90]:
```
A & B # intersection A & B common item both
```

Out[90]:
```
{4, 5}
```

In [91]:
```
A.intersection(B) intersection A & B
```

```
  Cell In[91], line 1
    A.intersection(B) intersection A & B
                      ^
SyntaxError: invalid syntax
```

In [92]:
```
"""
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[92]:
```
{4, 5}
```

Difference

In [93]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [94]:
```
A - B # set of element that are only in A not in B
```

Out[94]:
```
{1, 2, 3}
```

In [95]:
```
A.difference(B) # Difference of sets
```

Out[95]:
```
{1, 2, 3}
```

In [96]:
```
B - A # set of element that are only in B not in A
```

Out[96]:
```
{6, 7, 8}
```

In [97]:
```
B.difference(A)
```

Out[97]:
```
{6, 7, 8}
```

In [98]:
```
"""
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[98]:
```
{6, 7, 8}
```

Symmetric Difference

In [99]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [100…
```
A ^ B # in symetric difference (set of element A & B but not in both)
```

Out[100…
```
{1, 2, 3, 6, 7, 8}
```

In [101…
```
A.symmetric_difference(B) # symmetric difference of sets
```

Out[101…
```
{1, 2, 3, 6, 7, 8}
```

In [102…
```
B ^ A
```

Out[102…
```
{1, 2, 3, 6, 7, 8}
```

In [103…
```

"""
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[103…
```
{1, 2, 3, 6, 7, 8}
```

Subset Superset & Disjoint

In [104…
```
A = {1,2,3,4,5,6,7,8,9}
B = {3,4,5,6,7,8}
C = { 10,20,30,40}
```

In [105…
```
B.issubset(A) # set B is said to be the subset of set A if all elements of B are
```

Out[105…
```
True
```

In [106…
```
A.issuperset(B) # set A is said to be the superset of set B if all elements of B
```

Out[106…
```
True
```

In [107…
```
C.isdisjoint(A) # two sets are said to be disjoint set if they have no common el
```
Out[107…
```
True
```
In [108…
```
B.isdisjoint(A) # two sets are said to be disjoint sets it have no common elemen
```
Out[108…
```
False
```
In [109…
```
A
```
Out[109…
```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```
In [110…
```
sum(A)
```
Out[110…
```
45
```
In [111…
```
max(A)
```
Out[111…
```
9
```
In [112…
```
min(A)
```
Out[112…
```
1
```
In [113…
```
len(A)
```
Out[113…
```
9
```
In [114…
```
list(enumerate(A))
```
Out[114…
```
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]
```
In [115…
```
D= sorted (A,reverse=True)
```
In [116…
```
D
```
Out[116…
```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```
In [117…
```
sorted(D)
```
Out[117…
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

3. Dictionary

1. Dictionary is the mutable data type in Python.
2. a Python dictonary is a collection of keys & Value pair seprate in colan (:) in curley braces {}.
3. Keys must be unique in dictionary, duplicate value are allowed.

Create Dictionary

In [118…
```
mydict = dict() # empty dictionary
mydict
```
Out[118…
```
{}
```
In [119…
```
mydict = {}
mydict
```
Out[119…
```
{}
```
In [120…
```
mydict = {1:'one', 2:'two', 3:'three'} # dictionary with intigers keys
mydict
```
Out[120…
```
{1: 'one', 2: 'two', 3: 'three'}
```
In [121…
```
mydict = dict({1:'one', 2:'two', 3:'three'}) # create dictionary using dict
mydict
```
Out[121…
```
{1: 'one', 2: 'two', 3: 'three'}
```
In [122…
```
mydict = {'A':'one', 'B':'two', 'C':'three'} # dictionary with charecter keys
mydict
```
Out[122…
```
{'A': 'one', 'B': 'two', 'C': 'three'}
```
In [123…
```
mydict = {1:'one', 'A':'two', 3:'three'} # dictionary with mixed keys
mydict
```
Out[123…
```
{1: 'one', 'A': 'two', 3: 'three'}
```
In [124…
```
mydict.keys()# return dictionary keys using keys() method
```
Out[124…
```
dict_keys([1, 'A', 3])
```
In [125…
```
mydict.values() # return dictionary values using values() method
```
Out[125…
```
dict_values(['one', 'two', 'three'])
```
In [126…
```
mydict.items() # access each key-value pair within a dictionary
```
Out[126…
```
dict_items([(1, 'one'), ('A', 'two'), (3, 'three')])
```
In [127…
```
mydict = {1:'one', 2:'two', 'A':['Assif', 'Johan', 'Maria']}
mydict
```
Out[127…
```
{1: 'one', 2: 'two', 'A': ['Assif', 'Johan', 'Maria']}
```
In [128…
```
mydict = {1:'one', 2:'two', 'A':['Assif', 'Johan', 'Maria'], 'B':('Bat', 'Cat',
mydict
```
Out[128…
```
{1: 'one',
 2: 'two',
 'A': ['Assif', 'Johan', 'Maria'],
 'B': ('Bat', 'Cat', 'Hat')}
```

In [129…
```
mydict = {1:'one', 2:'two', 'A':{'Name':'Assif', 'Age':20}, 'B':('Bat', 'Hat', '
mydict
```

Out[129…
```
{1: 'one',
 2: 'two',
 'A': {'Name': 'Assif', 'Age': 20},
 'B': ('Bat', 'Hat', 'Cat')}
```

In [130…
```
keys ={'a', 'b', 'c', 'd'}
mydict3 = dict.fromkeys(keys) # create a dictionary form a sequence of keys
mydict3
```

Out[130…
```
{'b': None, 'd': None, 'a': None, 'c': None}
```

In [131…
```
keys = {'a', 'b', 'c', 'd'}
value = 10
mydict3 =dict.fromkeys(keys, value)# create a dictionary form a sequence of keys
mydict3
```

Out[131…
```
{'b': 10, 'd': 10, 'a': 10, 'c': 10}
```

In [132…
```
keys = {'a', 'b', 'c', 'd'}
value = [10,20,30]
mydict3 = dict.fromkeys(keys , value)
mydict3
```

Out[132…
```
{'b': [10, 20, 30], 'd': [10, 20, 30], 'a': [10, 20, 30], 'c': [10, 20, 30]}
```

In [133…
```
value.append(40)
mydict3
```

Out[133…
```
{'b': [10, 20, 30, 40],
 'd': [10, 20, 30, 40],
 'a': [10, 20, 30, 40],
 'c': [10, 20, 30, 40]}
```

Accessing Items

In [134…
```
mydict = {1:'one', 2:'two', 3:'three', 4:'four'}
mydict
```

Out[134…
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [135…
```
mydict[1] # access item using key
```

Out[135…
```
'one'
```

In [136…
```
mydict.get(1) # access item using get() method
```

Out[136…
```
'one'
```

In [137…
```
mydict1 = {'Name':'Assif', 'Id':74123, 'DOB':1991, 'job':'Analyst'}
mydict1
```

Out[137…
```
{'Name': 'Assif', 'Id': 74123, 'DOB': 1991, 'job': 'Analyst'}
```

In [138…
```
mydict1['Name']
```
Out[138…
```
'Assif'
```
In [139…
```
mydict1.get('job')
```
Out[139…
```
'Analyst'
```

Add, Remove & Change Items

In [140…
```
mydict1 = {'Name':'Assif', 'ID':12345, 'DOB':1991, 'Address':'Hilsinki'}
mydict1
```
Out[140…
```
{'Name': 'Assif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```
In [141…
```
mydict1['DOB'] = 1992 # chenging Dictionary item
mydict1['Address'] = 'Delhi'
mydict1
```
Out[141…
```
{'Name': 'Assif', 'ID': 12345, 'DOB': 1992, 'Address': 'Delhi'}
```
In [142…
```
dict1 = {'DOB':1995}
mydict1.update(dict1)
mydict1
```
Out[142…
```
{'Name': 'Assif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}
```
In [143…
```
mydict1['job'] ='Analytics' # adding items in dictionary
mydict1
```
Out[143…
```
{'Name': 'Assif',
 'ID': 12345,
 'DOB': 1995,
 'Address': 'Delhi',
 'job': 'Analytics'}
```
In [144…
```
mydict1.pop('job') # Removing items in the dictionary using pop method
mydict1
```
Out[144…
```
{'Name': 'Assif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}
```
In [145…
```
mydict1.popitem() # ramdom item is removed
```
Out[145…
```
('Address', 'Delhi')
```
In [146…
```
mydict1
```
Out[146…
```
{'Name': 'Assif', 'ID': 12345, 'DOB': 1995}
```
In [147…
```
del[mydict1['ID']] # Removing item using del method
mydict1
```
Out[147…
```
{'Name': 'Assif', 'DOB': 1995}
```
In [148…
```
mydict1.clear() # Deleting all items in dictionary using clear method
mydict1
```

```
Out[148… {}
```

```
In [149… del mydict1 #Delete the dictionary object
         mydict1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[149], line 2
      1 del mydict1 #Delete the dictionary object
----> 2 mydict1

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

Copy Dictionary

```
In [150… mydict = {'Name':'Assif', 'ID':12345, 'DOB':1991, 'Address':'Hilsinki'}
         mydict
```

```
Out[150… {'Name': 'Assif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

```
In [151… mydict1 = mydict #create a new Reference "mydict1"
```

```
In [152… id(mydict), id(mydict1 #) Address of both mydict & mydict1 will be same
```

```
  Cell In[152], line 1
    id(mydict), id(mydict1 #) Address of both mydict & mydict1 will be same
                                                                           ^
SyntaxError: incomplete input
```

```
In [153… mydict2 = mydict.copy() # create a copy of Dictionary
```

```
In [154… id(mydict2) # the address of mydict2 will be different mydict
```

```
Out[154… 2327843922880
```

```
In [155… mydict['Address'] = 'Mumbai'
```

```
In [156… mydict
```

```
Out[156… {'Name': 'Assif', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}
```

```
In [157… mydict1 # mydict one is also impacted as it is pointing to the same dictionary
```

```
Out[157… {'Name': 'Assif', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}
```

```
In [158… mydict2 #copy on cellwon't impacted due to changes made in original
```

```
Out[158… {'Name': 'Assif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

Loop through a Dictionary

```
In [159… mydict1 = {'Name':'Assif', 'ID':12345, 'DOB':1991, 'Address':'Hilsinki', 'job':'
         mydict1
```

Out[159…

```
{'Name': 'Assif',
 'ID': 12345,
 'DOB': 1991,
 'Address': 'Hilsinki',
 'job': 'Analyst'}
```

In [160…

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

```
Name : Assif
ID : 12345
DOB : 1991
Address : Hilsinki
job : Analyst
```

In [161…

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

```
Assif
12345
1991
Hilsinki
Analyst
```

Dictionary Membership

In [162…

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

Out[162…

```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [163…

```
'Name' in mydict1
```

Out[163…

```
True
```

In [164…

```
'Asif' in mydict1
```

Out[164…

```
False
```

In [165…

```
'ID' in mydict1
```

Out[165…

```
True
```

In [166…

```
'Address' in mydict1
```

Out[166…

```
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 fase.l

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

```
In [167…  mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
          mydict1

Out[167…  {'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}

In [168…  all(mydict1) # will retyrn false as one value is false (value 0)

Out[168…  True
```