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.

In [2]:
```
tup1 = () # Empty tuple
```

In [3]:
```
tup1
```

Out[3]:
```
()
```

In [4]:
```
tup2=(10,20,30)  # tuple of integers numbers
tup2
```

Out[4]:
```
(10, 20, 30)
```

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

Out[5]:
```
(10.77, 30.66, 60.89)
```

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

Out[6]:
```
('one', 'two', 'three')
```

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

In [8]:
```
tup5
```

Out[8]:
```
('Asif', 25, (50, 100), (150, 90))
```

In [9]:
```
tup6 = (100, 'Asif', 17.765) # Tuple of mixed data types
tup6
```

Out[9]:
```
(100, 'Asif', 17.765)
```

In [10]:
```
tup7 = ('Asif', 25 ,[50, 100],[150, 90] , {'John' , 'David'} , (99,22,33))
tup7
```

Out[10]:
```
('Asif', 25, [50, 100], [150, 90], {'David', 'John'}, (99, 22, 33))
```

In [11]:
```
len(tup7) #Length of list
```

Out[11]:
```
6
```

Tuple Indexing

In [13]:
```
tup2[0]  # Retreive first element of the tuple
```
Out[13]:
```
10
```

In [14]:
```
tup4[0] # Retreive first element of the tuple
```
Out[14]:
```
'one'
```

In [15]:
```
tup4[0][0] # Nested indexing - Access the first character of the first tuple elemen
```
Out[15]:
```
'o'
```

In [16]:
```
tup4[-1] # Last item of the tuple
```
Out[16]:
```
'three'
```

In [17]:
```
tup5[-1] # Last item of the tuple
```
Out[17]:
```
(150, 90)
```

Tuple Slicing

In [20]:
```
mytuple = ('one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight')
```

In [21]:
```
mytuple[0:3] # Return all items from 0th to 3rd index location excluding the item
```
Out[21]:
```
('one', 'two', 'three')
```

In [22]:
```
mytuple[2:5]
```
Out[22]:
```
('three', 'four', 'five')
```

In [23]:
```
mytuple[:3] # Return first three items
```
Out[23]:
```
('one', 'two', 'three')
```

In [24]:
```
mytuple[2:5] # List all items from 2nd to 5th index location excluding the item
```
Out[24]:
```
('three', 'four', 'five')
```

In [25]:
```
mytuple[:3] # Return first three items
```
Out[25]:
```
('one', 'two', 'three')
```

In [26]:
```
mytuple[:2] # Return first two items
```
Out[26]:
```
('one', 'two')
```

In [27]:
```
mytuple[:2] # Return first two items
```
Out[27]:
```
('one', 'two')
```

In [28]:
```
mytuple[-3:] # Return last three items
```

Out[28]:
```
('six', 'seven', 'eight')
```

In [29]:
```
mytuple[-3:] # Return last three items
```

Out[29]:
```
('six', 'seven', 'eight')
```

In [30]:
```
mytuple[-2:] # Return last two items
```

Out[30]:
```
('seven', 'eight')
```

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

In [31]:
```
mytuple[:] # Return whole tuple
```

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

Remove & Change Items

In [32]:
```
mytuple
```

Out[32]:
```
('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

In [33]:
```
del mytuple[0] # Tuples are immutable which means we can't DELETE tuple items
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[33], line 1
----> 1 del mytuple[0] # Tuples are immutable which means we can't DELETE tuple item
s

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

In [38]:
```
mytuple[0] = 1 # Tuples are immutable which means we can't CHANGE tuple items
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[38], line 1
----> 1 mytuple[0] = 1 # Tuples are immutable which means we can't CHANGE tuple item
s

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

In [35]:
```
del mytuple # Deleting entire tuple object is possible
```

Loop through a tuple

In [46]:
```
mytuple = ('one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight')
```

In [48]:
```
for i in mytuple:
    print(i)
```

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

In [49]:
```
for i in enumerate(mytuple):
    print(i)
```

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

In [50]:
```
mytuple1 =('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')
```

In [51]:
```
mytuple1.count('one') # Number of times item "one" occurred in the tuple.
```

Out[51]:
```
3
```

In [52]:
```
mytuple1.count('two') # Occurence of item 'two' in the tuple
```

Out[52]:
```
2
```

In [53]:
```
mytuple1.count('four') #Occurence of item 'four' in the tuple
```

Out[53]:
```
1
```

In [54]:
```
mytuple1.count('four') #Occurence of item 'four' in the tuple
```

Out[54]:
```
1
```

Tuple Membership

In [55]:
```
mytuple
```

Out[55]:
```
('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

In [56]:
```
'one' in mytuple # check if exits or not
```

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

In [57]:
```
'ten' in mytuple # Check if 'ten' exist in the list
```

Out[57]:
```
False
```

In [59]:
```
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 [62]:
```
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 [63]:
```
mytuple
```

Out[63]:
```
('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

In [64]:
```
mytuple.index('one')
```

Out[64]:
```
0
```

In [66]:
```
mytuple.index('five')
```

Out[66]:
```
4
```

In [67]:
```
mytuple1
```

Out[67]:
```
('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')
```

In [68]:
```
mytuple1.index('one')
```

Out[68]:
```
0
```

Sorting

In [69]:
```
mytuple2 = (43,67,99,12,6,90,67)
```

In [70]:
```
sorted(mytuple2)
```

Out[70]:
```
[6, 12, 43, 67, 67, 90, 99]
```

In [71]:
```
sorted(mytuple2, reverse=True) # Sort in descending order
```

Out[71]:
```
[99, 90, 67, 67, 43, 12, 6]
```

Set Creation

In [72]:
```
my_set = {1,1,2,2,3,4,5,5}
```

In [73]:
```
my_set   # Duplicate elements are not allowed.
```

Out[73]:
```
{1, 2, 3, 4, 5}
```

In [74]:
```
myset1 = {1.79,2.08,3.99,4.56,5.45} # Set of float numbers
myset1
```

Out[74]:
```
{1.79, 2.08, 3.99, 4.56, 5.45}
```

In [75]:
```
myset2 = {'Asif' , 'John' , 'Tyrion'} # Set of Strings
```

In [76]:
```
myset2
```

Out[76]:
```
{'Asif', 'John', 'Tyrion'}
```

In [77]:
```
myset3 = {10,20, "Hola", (11, 22, 32)} # Mixed datatypes
```

In [78]:
```
myset3
```

Out[78]:
```
{(11, 22, 32), 10, 20, 'Hola'}
```

In [80]:
```
myset3 = {10,20, "Hola", [11, 22, 32]} # set doesn't allow mutable items like list
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[80], line 1
----> 1 myset3 = {10,20, "Hola", [11, 22, 32]} # set doesn't allow mutable items lik
e list

TypeError: unhashable type: 'list'
```

In [82]:
```
myset4 = set() # Create an empty set
print(type(myset4))
```

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

In [83]:
```
my_set1 = set(('one' , 'two' , 'three' , 'four'))
```

In [84]:
```
my_set1
```

Out[84]:
```
{'four', 'one', 'three', 'two'}
```

Loop through a Set

In [86]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
for i in myset:
    print(i)
```

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

In [87]:
```
for i in enumerate(myset):
    print(i)
```

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

Set Membership

In [88]:
```
myset
```

Out[88]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [89]:
```
'one' in myset # Check if 'one' exist in the set
```

Out[89]:
```
True
```

In [90]:
```
'ten' in myset # Check if 'ten' exist in the set
```

Out[90]:
```
False
```

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

In [92]:
```
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 [93]:
```
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 [3]:
```
myset ={'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [4]:
```
myset.add('NINE') # Add item to a set using add() method
```

In [5]:
```
myset
```

Out[5]:
```
{'NINE', 'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [6]:
```
myset.update(['TEN' , 'ELEVEN' , 'TWELVE']) # Add multiple item to a set using
```

In [7]:
```
myset
```

Out[7]:
```
{'ELEVEN',
 'NINE',
 'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [8]:
```

myset.remove('NINE') # remove item in a set using remove() method
myset
```

Out[8]:
```
{'ELEVEN',
 'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [9]:
```
myset.discard('TEN') # remove item from a set using discard() method
```

In [10]:
```
myset
```

Out[10]:
```
{'ELEVEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [11]:
```
myset.clear() # Delete all items in a set
myset
```

Out[11]:
```
set()
```

In [12]:
```
del myset # Delete the set object
myset
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[12], line 2
      1 del myset # Delete the set object
----> 2 myset

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

Copy Set

In [13]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
```

In [16]:
```
myset1=myset
myset1
```

Out[16]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [17]:
```
id(myset) , id(myset1) # The address of both myset & myset1 will be the same as
```

Out[17]:
```
(2249389753280, 2249389753280)
```

In [18]:
```
my_set = myset.copy() # Create a copy of the list
```

In [19]:
```
id(my_set) # The address of my_set will be different from myset because my_set
```

Out[19]:
```
2249389757984
```

In [20]:
```

myset.add('nine')
```

In [21]:
```
myset
```

Out[21]:
```
{'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

In [22]:
```
myset1 # myset1 will be also impacted as it is pointing to the same Set
```

Out[22]:
```
{'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

In [23]:
```
my_set # Copy of the set won't be impacted due to changes made on the original S et
```

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

Set Operation

Union

In [25]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
C = {8,9,10}
```

In [26]:
```
A | B # Union of A and B (All elements from both sets. NO DUPLICATES)
```

Out[26]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [27]:
```
A.union(B) # Union of A and B
```

Out[27]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [28]:
```

A.union(B, C) # Union of A, B and C.
```

Out[28]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

""" 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. """

In [31]:
```
A.update(B,C)
A
```

Out[31]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Intersection

In [34]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [36]:
```
A & B # Intersection of A and B (Common items in both sets)
```

Out[36]:
```
{4, 5}
```

In [107…
```
A.intersection(B) Intersection of A and B
```

```
  Cell In[107], line 1
    A.intersection(B) Intersection of A and B
                      ^
SyntaxError: invalid syntax
```

In [114…
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
A.intersection_update(B)
```

In [115…
```
A
```

Out[115…
```
{4, 5}
```

Difference

In [117…
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [118…
```
A - B # set of elements that are only in A but not in B
```

Out[118…
```
{1, 2, 3}
```

In [119…
```
A.difference(B) # Difference of sets
```

Out[119…
```
{1, 2, 3}
```

In [120…
```
B- A # set of elements that are only in B but not in A
```

Out[120…
```
{6, 7, 8}
```

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.

In [124…
```
B.difference_update(A)
```

In [125…
```
B
```

Out[125…
```
{6, 7, 8}
```

Symmetric Difference- Symmetric difference A ^ B=(A U B)-(A n B)

In [136…
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [137…
```
A^B
```

Out[137… {1, 2, 3, 6, 7, 8}

In [138…
```
A.symmetric_difference(B) # Symmetric difference of sets
```

Out[138… {1, 2, 3, 6, 7, 8}

In [139…
```
A.symmetric_difference_update(B)
```

In [140…
```
A
```

Out[140… {1, 2, 3, 6, 7, 8}

Subset , Superset & Disjoint

In [141…
```
A = {1,2,3,4,5,6,7,8,9}
B = {3,4,5,6,7,8}
C = {10,20,30,40}
B.issubset(A)
```

Out[141… True

In [142…
```
A.issuperset(B)
```

Out[142… True

In [143…
```
C.isdisjoint(A)
```

Out[143… True

In [144…
```
B.isdisjoint(A)
```

Out[144… False

In [145…
```
A
```

Out[145… {1, 2, 3, 4, 5, 6, 7, 8, 9}

In [147…
```
sum(A)
```

Out[147… 45

In [148…
```
max(A)
```

Out[148… 9

In [149…
```
min(A)
```

Out[149… 1

In [150…
```
len(A)
```

Out[150… 9

In [151…
```
list(enumerate(A))
```

Out[151… [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]

In [153…
```
D= sorted(A,reverse=True)
D
```

Out[153… [9, 8, 7, 6, 5, 4, 3, 2, 1]

In [154…
```
sorted(D)
```

Out[154… [1, 2, 3, 4, 5, 6, 7, 8, 9]

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.

In [225…
```
mydict = dict() # empty dictionary
mydict
```

Out[225… {}

In [158…
```
mydict = {} # empty dictionary
```

In [159…
```
mydict
```

Out[159… {}

In [160…
```
mydict = {1:'one' , 2:'two' , 3:'three'} # dictionary with integer keys
```

In [161…
```
mydict
```

Out[161… {1: 'one', 2: 'two', 3: 'three'}

In [167…
```
mydict = {'A':'one' , 'B':'two' , 'C':'three'} # dictionary with character keys
```

In [165…
```
mydict
```

Out[165… {'A': 'one', 'B': 'two', 'C': 'three'}

In [168…
```
mydict.keys()
```

Out[168… dict_keys(['A', 'B', 'C'])

In [169…
```
mydict.values()
```

Out[169… dict_values(['one', 'two', 'three'])

In [170…
```
mydict.items()  # Access each key-value pair within a dictionary
```

Out[170…
```
dict_items([('A', 'one'), ('B', 'two'), ('C', 'three')])
```

In [171…
```
mydict = {1:'one' , 2:'two' , 'A':['asif' , 'john' , 'Maria']} # dictionary with ke
mydict
```

Out[171…
```
{1: 'one', 2: 'two', 'A': ['asif', 'john', 'Maria']}
```

In [174…
```

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

Out[174…
```
{1: 'one',
 2: 'two',
 'A': ['asif', 'john', 'Maria'],
 'B': ('Bat', 'cat', 'hat')}
```

In [176…
```
mydict = {1:'one' , 2:'two' , 'A':{'Name':'asif' , 'Age' :20}, 'B':('Bat' , 'cat','
```

In [177…
```
mydict
```

Out[177…
```
{1: 'one',
 2: 'two',
 'A': {'Name': 'asif', 'Age': 20},
 'B': ('Bat', 'cat', 'hat')}
```

In [179…
```
keys = {'a' , 'b' , 'c' , 'd'}
mydict3 = dict.fromkeys(keys) # Create a dictionary from a sequence of keys
mydict3
```

Out[179…
```
{'a': None, 'd': None, 'b': None, 'c': None}
```

In [180…
```
keys = {'a' , 'b' , 'c' , 'd'}
value = 10
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[180…
```
{'a': 10, 'd': 10, 'b': 10, 'c': 10}
```

In [181…
```
keys = {'a' , 'b' , 'c' , 'd'}
value = [10,20,30]
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[181…
```
{'a': [10, 20, 30], 'd': [10, 20, 30], 'b': [10, 20, 30], 'c': [10, 20, 30]}
```

In [182…
```
value.append(40)
mydict3
```

Out[182…
```
{'a': [10, 20, 30, 40],
 'd': [10, 20, 30, 40],
 'b': [10, 20, 30, 40],
 'c': [10, 20, 30, 40]}
```

Accessing Items

In [183…
```
mydict = {1:'one' , 2:'two' , 3:'three' , 4:'four'}
mydict
```

Out[183…
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [184…
```
mydict[1] # Access item using key
```

Out[184…
```
'one'
```

In [185…
```
mydict.get(1) # Access item using get() method
```

Out[185…
```
'one'
```

In [186…
```
mydict1 = {'Name':'Asif' , 'ID': 74123 , 'DOB': 1991 , 'job' :'Analyst'}
mydict1
```

Out[186…
```
{'Name': 'Asif', 'ID': 74123, 'DOB': 1991, 'job': 'Analyst'}
```

In [187…
```
mydict1['Name'] # Access item using key
```

Out[187…
```
'Asif'
```

In [188…
```
mydict1.get('job') # Access item using get() method
```

Out[188…
```
'Analyst'
```

Add, Remove & Change Items

In [189…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
mydict1
```

Out[189…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [190…
```
mydict1['DOB'] = 1992 # Changing Dictionary Items
mydict1['Address'] = 'Delhi'
mydict1
```

Out[190…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1992, 'Address': 'Delhi'}
```

In [191…
```
dict1 = {'DOB':1995}
mydict1.update(dict1)
mydict1
```

Out[191…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}
```

In [192…
```
mydict1['Job'] = 'Analyst' # Adding items in the dictionary
mydict1
```

Out[192…
```
{'Name': 'Asif',
 'ID': 12345,
 'DOB': 1995,
 'Address': 'Delhi',
 'Job': 'Analyst'}
```

In [193…
```
mydict1.pop('Job') # Removing items in the dictionary using Pop method
mydict1
```

Out[193…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}
```

In [194…
```
mydict1.popitem()
```

Out[194…
```
('Address', 'Delhi')
```

In [195…
```
mydict1
```

Out[195…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995}
```

In [196…
```
del[mydict1['ID']] # Removing item using del method
mydict1
```

Out[196…
```
{'Name': 'Asif', 'DOB': 1995}
```

In [197…
```
mydict1.clear() # Delete all items of the dictionary using clear method
mydict1
```

Out[197…
```
{}
```

In [201…
```
del mydict1# Delete all items of the dictionary using clear method
mydict1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[201], line 2
      1 del mydict1# Delete all items of the dictionary using clear method
----> 2 mydict1

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

Copy Dictionary

In [202…
```
mydict = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
mydict
```

Out[202…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [203…
```
mydict1 = mydict # Create a new reference "mydict1"
id(mydict) , id(mydict1) # The address of both mydict & mydict1 will be the same
```

Out[203…
```
(2274873776576, 2274873776576)
```

In [204…
```
mydict2 = mydict.copy() # Create a copy of the dictionary
id(mydict2) # The address of mydict2 will be different from mydict because mydic
```

Out[204…
```
2274874033984
```

In [205…
```
mydict['Address'] = 'Mumbai'
mydict
```

Out[205…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}
```

In [206…
```
mydict1 # mydict1 will be also impacted as it is pointing to the same dictionary
```

Out[206…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}
```

In [207…
```
mydict2 # Copy of list won't be impacted due to the changes made in the original
```

Out[207…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

Loop through a Dictionary

In [209…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki' , 'jo
mydict1
```

Out[209…
```
{'Name': 'Asif',
 'ID': 12345,
 'DOB': 1991,
 'Address': 'Hilsinki',
 'job': 'Analyst'}
```

In [211…
```
for i in mydict1:
    print(i , ':' , mydict1[i]) # Key & value pair
```

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

In [212…
```
for i in mydict1:
    print(mydict1[i]) # Dictionary items
```

```
Asif
12345
1991
Hilsinki
Analyst
```

Dictionary Membership

In [213…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[213…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [215…
```
'Name' in mydict1
```

Out[215…
```
True
```

In [216…
```
'Asif' in mydict1
```

Out[216…
```
False
```

In [217…
```
'ID' in mydict1
```

Out[217…
```
True
```

In [218…
```
'Address' in mydict1
```

Out[218…
```
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 [222…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
```

In [224…
```
all(mydict1)
```

Out[224…
```
True
```

In [ ]: