Data Structures in Python

Lists

Tuples

Sets

Dictionaries etc.

```
In [ ]:
```

Lists --- Ordered collections of items that can be of any data type, including strings, integers, floats, and other lists. []

```
In [10]: my_list = []    # Empty list
         my_list

Out[10]: []

In [24]: my_list = list()    # Empty list
         my_list

Out[24]: []

In [26]: print(type(my_list))

<class 'list'>

In [28]: len(my_list)

Out[28]: 0

In [30]: my_list.append(1)        # Add an item to the list
         my_list

Out[30]: [1]

In [32]: my_list.append([2,3])        # Add an item to the list
         my_list

Out[32]: [1, [2, 3]]

In [36]: my_list.append((1, 3, 4, 7))        # Add an item to the list
         my_list

Out[36]: [1, [2, 3], (1, 3, 4, 7)]

In [38]: len(my_list)

Out[38]: 3

In [40]: id(my_list)
```

Out[40]:
```
2281302883072
```

In [42]:
```
import keyword
keyword.kwlist
```

Out[42]:
```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'async',
 'await',
 'break',
 'class',
 'continue',
 'def',
 'del',
 'elif',
 'else',
 'except',
 'finally',
 'for',
 'from',
 'global',
 'if',
 'import',
 'in',
 'is',
 'lambda',
 'nonlocal',
 'not',
 'or',
 'pass',
 'raise',
 'return',
 'try',
 'while',
 'with',
 'yield']
```

In [46]:
```
len(keyword.kwlist)
```

Out[46]:
```
35
```

In [72]:
```
my_list1 = my_list.copy()
my_list1
```

Out[72]:
```
[1, [2, 3], (1, 3, 4, 7)]
```

In [74]:
```
# Slicing
```

In [76]:
```
my_list1[:]
```

Out[76]:
```
[1, [2, 3], (1, 3, 4, 7)]
```

In [78]:
```
my_list1[:2]
```

```
Out[78]:  [1, [2, 3]]

In [80]:  my_list1[-1]

Out[80]:  (1, 3, 4, 7)

In [82]:  my_list1.append(2.3)
          my_list1.append(True)
          my_list1.append(3+2.7j)
          my_list1.append(2.3)

In [84]:  my_list1

Out[84]:  [1, [2, 3], (1, 3, 4, 7), 2.3, True, (3+2.7j), 2.3]

In [86]:  my_list1.count(2.3)

Out[86]:  2

In [90]:  my_list1.remove(True)    # Remove the element
          my_list1

Out[90]:  [[2, 3], (1, 3, 4, 7), 2.3, (3+2.7j), 2.3]

In [92]:  my_list1.clear()
          my_list1

Out[92]:  []

In [94]:  my_list2 = [1,3,4,5,6]
          my_list2

Out[94]:  [1, 3, 4, 5, 6]

In [98]:  my_list2[1]    # indexing

Out[98]:  3

In [100…  for i in my_list2:
              print(i)

        1
        3
        4
        5
        6

In [102…  for i, num in enumerate(my_list2):
              print(i, num)

        0 1
        1 3
        2 4
        3 5
        4 6

In [104…  for num in enumerate(my_list2):
    print(num)
```

```
(0, 1)
(1, 3)
(2, 4)
(3, 5)
(4, 6)
```

In [142…
```
my_list3 = [7, 4]
```

In [144…
```
my_list3.append(['Hi', 20, 3.2, False, 2+3j])  # Nested  list
my_list3
```

Out[144…
```
[7, 4, ['Hi', 20, 3.2, False, (2+3j)]]
```

In [146…
```
my_list3.pop(-1)
```

Out[146…
```
['Hi', 20, 3.2, False, (2+3j)]
```

In [148…
```
my_list3.insert(13, 1)
my_list3
```

Out[148…
```
[7, 4, 1]
```

In [150…
```
my_list3.index(4)
```

Out[150…
```
1
```

In [154…
```
my_list3.sort(reverse=True)    # Descending order
my_list3
```

Out[154…
```
[7, 4, 1]
```

In [156…
```
my_list3.sort(reverse=False)    # escending order
my_list3
```

Out[156…
```
[1, 4, 7]
```

In [ ]:

In [ ]:

Tuples --- Ordered, immutable collections of items that can be of any data type. ()

- Tuple is similar to List except that the objects in Tuple are immutable which means we can not change the items of a Tuple once assigned - Tuple is prefered data type when we do not want to change the data over time. - Iterating over the items of a tuple is faster compared to iterating over a list.

In [6]:
```
tup1 = ()   # Empty tuple
tup1
```

Out[6]:
```
()
```

In [7]:
```
print(type(tup1))
```

```
<class 'tuple'>
```

```
In [8]: tup2 = (10, 20, 30)     # tuple of intger numbers
```

```
In [9]: tup3 = (10.77, 30.66, 60.89)  # tuple of float numbers
```

```
In [10]: tup4 = ('one', 'two', 'three')  # tuple of strings
```

```
In [11]: tup5 = ('nested', 25, (50, 100), (150, 78))    # Nested tuple
```

```
In [12]: len(tup5)     # length of tuple
Out[12]: 4
```

```
In [13]: tup6 = (100, 'Asif', 17.65)     # Tuple with mixed data types
```

```
In [14]: tup7 = ('Asif', 25, [50, 100], [159, 48], {'mike', 'Jaden', 'Princess'}, (12, 23
```

```
In [15]: print(len(tup7))
6
```

Tuple Indexing

```
In [17]: tup2[0]   # Retreive first item in the tuple
Out[17]: 10
```

```
In [18]: tup4[0]
Out[18]: 'one'
```

```
In [19]: tup4
Out[19]: ('one', 'two', 'three')
```

```
In [20]: tup4[0][0]   # Nested indexing - accessing the first character of the first tupl
Out[20]: 'o'
```

```
In [21]: tup4[-1]   # last item in the tuple
Out[21]: 'three'
```

```
In [22]: tup5
Out[22]: ('nested', 25, (50, 100), (150, 78))
```

```
In [23]: tup5[-1]
Out[23]: (150, 78)
```

Tuple Slicing

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

In [26]:
```
mytuple[:]    # column (:) inside a square bracket with variable name, gives all
```
Out[26]:
```
('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

In [27]:
```
mytuple[0:3]   # Return all items from 0th to 3rd index location excluding the i
```
Out[27]:
```
('one', 'two', 'three')
```

In [28]:
```
mytuple[2:5]   # List all items from 2nd to 5th index loction excluding the item
```
Out[28]:
```
('three', 'four', 'five')
```

In [29]:
```
mytuple[:3]  # Return the first three items
```
Out[29]:
```
('one', 'two', 'three')
```

In [30]:
```
mytuple[-3:]   # Return last three items
```
Out[30]:
```
('six', 'seven', 'eight')
```

Remove and Change Items

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

We can not Remove or Change items in a Tuple because its IMMUTABLE but deleting (del) the entire tuple is possible

In [33]:
```
#del mytuple[0] = 1 # can not CHANGE TUPLE items
#del mytuple[0]     # can not DELETE TUPLE item
#del mytuple    # Deleting entire tuple is possible
```

In [ ]:
```

```

In [ ]:
```

```

Loop through a tuple

In [35]:
```
mytuple
```
Out[35]:
```
('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')
```

In [36]:
```
for item in mytuple:
    print(item)
```
```
one
two
three
four
five
six
seven
eight
```

In [37]:
```
### Enumerate
```

```
In [38]: for i in enumerate(mytuple):
             print(i)
```

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

```
In [39]: for index, item in enumerate(mytuple):
             print(index, item)
```

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

Count

```
In [41]: mytuple1 = ('one', 'two', 'three', 'four', 'one', 'two', 'three')
```

```
In [42]: mytuple1.count('one')    # Number of times item "one" occured in the tuple
```

```
Out[42]: 2
```

```
In [43]: mytuple1.count('four')
```

```
Out[43]: 1
```

Tuple Menmbership

```
In [45]: mytuple1
```

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

```
In [46]: 'one' in mytuple1   # Check if 'one' exist in the tuple
```

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

```
In [47]: 'ten' in mytuple1
```

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

```
In [48]: if 'three' in mytuple1:          # Check if 'three' exist in the tuple
             print('Three is present in the tuple')
         else:
             print('Three is not present in the tuple')
```

```
Three is present in the tuple
```

In [49]:
```
if 'ten' in mytuple1:
    print('Ten is present in the tuple')
else:
    print('Ten is not present in the tuple')
```
```
Ten is not present in the tuple
```

In [ ]:

Index Position

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

In [52]:
```
mytuple[3]   # indexing
```
Out[52]:
```
'four'
```

In [53]:
```
mytuple.index('five')  # Index of first element equal to four
```
Out[53]:
```
4
```

Sorting

In [55]:
```
mytuple2 = (43, 67, 99, 12, 6, 90, 67)
```

In [56]:
```
sorted(mytuple2)    # Return a new sorted list and does not change original tupl
```
Out[56]:
```
[6, 12, 43, 67, 67, 90, 99]
```

In [57]:
```
sorted(mytuple2, reverse=True)   # Sort in decending order
```
Out[57]:
```
[99, 90, 67, 67, 43, 12, 6]
```

In [ ]:

Sets Unordered collections of unique items. {}

- Unordered & Unindexed collection of items - Set elements are unique. Duplicate elements are not allowed - Set elements are immutable (cannot changed) - Set itself is mutable ie we can add or remove ites from it

In [59]:
```
myset = {1,2,3,4,5}     # Ste of a numbers
print(type(myset))
```
```
<class 'set'>
```

In [60]:
```
len(myset)    # Length of the set
```
Out[60]:
```
5
```

In [61]:
```
my_set = {1,1,2,2,3,4,5,5}    # Duplicate items are not allowed
my_set
```
Out[61]:
```
{1, 2, 3, 4, 5}
```

In [62]:
```
myset1 = {1.74, 2.07, 3.99, 4.23, 5.0}  # Set of foats
myset1
```
Out[62]:
```
{1.74, 2.07, 3.99, 4.23, 5.0}
```
In [63]:
```
myset2 = {'Asif', 'Mike', 'Jaden'}   # Set of string
myset2
```
Out[63]:
```
{'Asif', 'Jaden', 'Mike'}
```
In [64]:
```
#myset3 = {10, 20, 3.4, [11, 12, 13]}  # Set does not allow mutable items list l
```
In [65]:
```
mylist3 = [10, 20, 3.4, [11, 12, 13], {2,2,4}]   # List allow  mutable items
mylist3
```
Out[65]:
```
[10, 20, 3.4, [11, 12, 13], {2, 4}]
```
In [66]:
```
myset4 = set()    # Create an empty set
print(type(myset4))
```
```
<class 'set'>
```
In [67]:
```
my_set1 = set(('one', 'two', 'three', 'four'))
my_set1
```
Out[67]:
```
{'four', 'one', 'three', 'two'}
```

Loop through a Set

In [69]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
myset
```
Out[69]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```
In [70]:
```
for i in myset:
    print(i)
```
```
six
one
seven
two
three
five
four
eight
```
In [71]:
```
for i in enumerate(myset):
    print(i)
```
```
(0, 'six')
(1, 'one')
(2, 'seven')
(3, 'two')
(4, 'three')
(5, 'five')
(6, 'four')
(7, 'eight')
```

In [72]:
```
for i, x in enumerate(myset):
    print(i, x)
```
```
0 six
1 one
2 seven
3 two
4 three
5 five
6 four
7 eight
```

Set Membership

In [74]:
```
myset
```
Out[74]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [75]:
```
if 'one' in myset:
    print("One is present in the set")
else:
    print("One is not present in the set")
```
```
One is present in the set
```

In [76]:
```
'one' in myset
```
Out[76]:
```
True
```

Add and Remove Items

In [78]:
```
myset
```
Out[78]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [79]:
```
myset.add('NINE')     # Add item to a set using add() method
myset
```
Out[79]:
```
{'NINE', 'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [80]:
```
myset.update(['TEN', 'ELEVEN', 'TWELVE'])    # Add multiple item to a set using
myset
```
Out[80]:
```
{'ELEVEN',
 'NINE',
 'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

```
In [81]: myset.remove('NINE')    # Remove item from a set using remove() method
         myset
```

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

```
In [82]: myset.discard('TEN')  # Remove item from a set using discard() method
         myset
```

```
Out[82]: {'ELEVEN',
          'TWELVE',
          'eight',
          'five',
          'four',
          'one',
          'seven',
          'six',
          'three',
          'two'}
```

```
In [83]: myset.clear()   # Delete all items in a set
         myset
```

```
Out[83]: set()
```

```
In [84]: #del myset  # Delete the set object
```

Copy Set

```
In [86]: myset = {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
         myset
```

```
Out[86]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [87]: myset1 = myset     # Create a new reference 'myset1'
         myset1
```

```
Out[87]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [88]: id(myset), id(myset1)  #  The address of both myset1 wii be thesame as myset
```

```
Out[88]: (1912595856768, 1912595856768)
```

```
In [ ]:
```

```
In [89]: my_set = myset.copy()   # Create a copy of the list
         my_set
```

```
Out[89]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [90]: id(my_set)   # The address of my_set is different from myset
```

```
Out[90]: 1912595855648
```

```
In [91]: myset
```

```
Out[91]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [92]: myset.add('nine')
         myset
```

```
Out[92]: {'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [93]: myset1    # is impacted as it is pointing to thesame address
```

```
Out[93]: {'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

```
In [94]: my_set  # Copy of the set wont be impacted due to changes made on the original
```

```
Out[94]: {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

Set Operation

```
In [190…  # Union
```

```
In [192…  A = {1,2,3,4,5}
          B = {4,5,6,7,8}
          C = {8,9,10}
```

```
In [194…  A | B  # Union of A and B (All elements from both sets. NO Duplicate)
```

```
Out[194…  {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [196…  A.union(B) # Union of A and B
```

```
Out[196…  {1, 2, 3, 4, 5, 6, 7, 8}
```

```
In [200…  A.union(B, C) # Union of A, B and C
```

```
Out[200…  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

```
In [204…  """
          Update 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[204…  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [206…
```
# Intersection
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [210…
```
A & B # Intersection of A and B (Common items in both sets)
```

Out[210…
```
{4, 5}
```

In [212…
```
A.intersection(B)   # Intersection of A and B
```

Out[212…
```
{4, 5}
```

In [216…
```
A.intersection_update(B)    # Update the with the intersection of A and B
A
```

Out[216…
```
{4, 5}
```

In [218…
```
# Difference
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [220…
```
A - B  # Set of elements that are only in A but not in B
```

Out[220…
```
{1, 2, 3}
```

In [222…
```
B - A
```

Out[222…
```
{6, 7, 8}
```

In [224…
```
A.difference(B) # Difference of sets
```

Out[224…
```
{1, 2, 3}
```

In [226…
```
B.difference_update(A)
B
```

Out[226…
```
{6, 7, 8}
```

In [242…
```
# Symmetric Difference
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [244…
```
A ^ B # symmetrical difference (Set of elements in A and B but not in both. "EXC
```

Out[244…
```
{1, 2, 3, 6, 7, 8}
```

In [248…
```
A.symmetric_difference(B) # Symmentric difference of sets
```

Out[248…
```
{1, 2, 3, 6, 7, 8}
```

In [250…
```
"""
Updates the set calling the symmentric_difference_update() method with the symme
for below example set A will be updated with the symmentric difference of A & B
"""
```

```
A.symmetric_difference_update(B)
A
```

Out[250…
```
{1, 2, 3, 6, 7, 8}
```

Subset, Superset & Disjoint

In [253…
```
A = {1,2,3,4,5,6,7,8,9}
B = {3,4,5,6,7,8}
C = {10,20,30,40}
```

In [255…
```
B.issubset(A)  # Set B is said to be the subset of A if all elements of B are fo
```

Out[255…
```
True
```

In [257…
```
A.issuperset(B)  # Set A is said to the superset of B if all elements of B are c
```

Out[257…
```
True
```

In [259…
```
C.isdisjoint(A)  # Two set are set to be disjoint if they have no common element
```

Out[259…
```
True
```

In [261…
```
B.isdisjoint(A)
```

Out[261…
```
False
```

Other Built in functions

In [264…
```
A
```

Out[264…
```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [266…
```
sum(A)
```

Out[266…
```
45
```

In [268…
```
max(A)
```

Out[268…
```
9
```

In [270…
```
min(A)
```

Out[270…
```
1
```

In [272…
```
len(A)
```

Out[272…
```
9
```

In [274…
```
list(enumerate(A))
```

Out[274…
```
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]
```

In [280…
```
D = sorted(A, reverse=True)
D
```

Out[280…
```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

In [282…
```
sorted(D)
```

Out[282…
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Dictionary

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

In [287…
```
mydict = dict()   # empty dictionary
mydict
```

Out[287…
```
{}
```

In [293…
```
mydict = {1:'one', 2:'two', 3:'three'}   # dictionary with integer keys
mydict
```

Out[293…
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [297…
```
mydict = {'A':'one', 'B':'two', 'C':'three'}   # dictionary with character keys
mydict
```

Out[297…
```
{'A': 'one', 'B': 'two', 'C': 'three'}
```

In [301…
```
mydict.keys()  # Return dictionary keys using keys () method
```

Out[301…
```
dict_keys(['A', 'B', 'C'])
```

In [303…
```
mydict.values()  # Return dictionary keys using values () method
```

Out[303…
```
dict_values(['one', 'two', 'three'])
```

In [305…
```
mydict.items()  # Access each key-value pair within a dictionary
```

Out[305…
```
dict_items([('A', 'one'), ('B', 'two'), ('C', 'three')])
```

In [307…
```
mydict = {1:'one', 2:'two', 'A':['Asif', 'Jaden', 'Maria'], 'B':('Bat', 'cat', '
mydict
```

Out[307…
```
{1: 'one',
 2: 'two',
 'A': ['Asif', 'Jaden', 'Maria'],
 'B': ('Bat', 'cat', 'hat')}
```

In [311…
```
keys = {'a', 'b', 'c', 'd'}
mydict3 = dict.fromkeys(keys)    # Create a dictionary from a sequence of keys
mydict3
```

Out[311…
```
{'a': None, 'b': None, 'd': None, 'c': None}
```

In [ ]:
```
keys = {'a', 'b', 'c', 'd'}
```

In [315…
```
value = 10
```

In [320…
```
mydict3 = dict.fromkeys(keys, value)
mydict3
```

Out[320…
```
{'a': 10, 'b': 10, 'd': 10, 'c': 10}
```

In [322…
```
keys = {'a', 'b', 'c', 'd'}
value = [10, 20, 30]
mydict3 = dict.fromkeys(keys, value)
mydict3
```

Out[322…
```
{'a': [10, 20, 30], 'b': [10, 20, 30], 'd': [10, 20, 30], 'c': [10, 20, 30]}
```

In [324…
```
value.append(40)
mydict3
```

Out[324…
```
{'a': [10, 20, 30, 40],
 'b': [10, 20, 30, 40],
 'd': [10, 20, 30, 40],
 'c': [10, 20, 30, 40]}
```

Accessing items

In [331…
```
mydict = {1:'one', 2:'two', 3:'three', 4:'four'}
mydict
```

Out[331…
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [335…
```
mydict[1]    # Access item using key
```

Out[335…
```
'one'
```

In [339…
```
mydict.get(1)   # Access item using method ()
```

Out[339…
```
'one'
```

In [347…
```
mydict1 = {'Name':'Asif', 'ID': 2345, 'DOB': 1917, 'job':'Analyst'}
mydict
```

Out[347…
```
{'Name': 'Asif', 'ID': 2345, 'DOB': 1917, 'job': 'Analyst'}
```

In [355…
```
mydict1['Name']  # Access item using key
```

Out[355…
```
'Asif'
```

In [353…
```
mydict1.get('job')
```

Out[353…
```
'Analyst'
```

Add, Remove & Change Items

In [358…
```
mydict1 = {'Name':'Asif', 'ID': 2345, 'DOB': 1917, 'Address':'Stewart'}
mydict1
```

```
Out[358…  {'Name': 'Asif', 'ID': 2345, 'DOB': 1917, 'Address': 'Stewart'}
```

```
In [360…  mydict1['DOB'] = 1992    # Changing Dictionary Items
          mydict1['Address'] = 'Slateford'
          mydict1
```

```
Out[360…  {'Name': 'Asif', 'ID': 2345, 'DOB': 1992, 'Address': 'Slateford'}
```

```
In [362…  dict1 = {'DOB':1990}
          mydict1.update(dict1)
          mydict1
```

```
Out[362…  {'Name': 'Asif', 'ID': 2345, 'DOB': 1990, 'Address': 'Slateford'}
```

```
In [364…  mydict1['Job'] = 'Analyst'   # adding items in the dictionary
          mydict1
```

```
Out[364…  {'Name': 'Asif',
           'ID': 2345,
           'DOB': 1990,
           'Address': 'Slateford',
           'Job': 'Analyst'}
```

```
In [366…  mydict1.pop('Job')  # Removing items in the dictionary using pop method
          mydict1
```

```
Out[366…  {'Name': 'Asif', 'ID': 2345, 'DOB': 1990, 'Address': 'Slateford'}
```

```
In [368…  mydict1.popitem()   # A random item is removed
```

```
Out[368…  ('Address', 'Slateford')
```

```
In [370…  mydict1
```

```
Out[370…  {'Name': 'Asif', 'ID': 2345, 'DOB': 1990}
```

```
In [372…  del[mydict1['ID']]   # Removing item using del method
          mydict1
```

```
Out[372…  {'Name': 'Asif', 'DOB': 1990}
```

```
In [374…  mydict1.clear() # Delete all items of the dictionary using clear method
          mydict1
```

```
Out[374…  {}
```

```
In [ ]:
```

Cope Dictionary

```
In [381…  mydict1 = {'Name': 'Asif', 'ID': 2345, 'DOB': 1990, 'Address': 'Slateford'}
          mydict1
```

```
Out[381…  {'Name': 'Asif', 'ID': 2345, 'DOB': 1990, 'Address': 'Slateford'}
```

In [383…
```
mydict1 = mydict   # Create a new reference 'mydict1'
```

In [387…
```
id(mydict) , id(mydict1)  # The both address are same
```

Out[387…
```
(1912619811072, 1912619811072)
```

In [389…
```
mydict2 = mydict1.copy()     # Create a copy of the dictionary
```

In [391…
```
id(mydict2)
```

Out[391…
```
1912618042624
```

In [393…
```
mydict['Address'] = 'Creg'
mydict
```

Out[393…
```
{'Name': 'Asif', 'ID': 2345, 'DOB': 1917, 'job': 'Analyst', 'Address': 'Creg'}
```

In [397…
```
mydict1   # this is also impacted
```

Out[397…
```
{'Name': 'Asif', 'ID': 2345, 'DOB': 1917, 'job': 'Analyst', 'Address': 'Creg'}
```

In [401…
```
mydict2    # Copy of list wont be impacted due to changes amde in the original
```

Out[401…
```
{'Name': 'Asif', 'ID': 2345, 'DOB': 1917, 'job': 'Analyst'}
```

Loop through a Dictionary

In [408…
```
mydict1 = {'Name': 'Asif', 'ID': 2345, 'DOB': 1990, 'Address': 'Slateford'}
mydict1
```

Out[408…
```
{'Name': 'Asif', 'ID': 2345, 'DOB': 1990, 'Address': 'Slateford'}
```

In [410…
```
for i in mydict1:
    print(i, ':', mydict1[i])  # key and value pair
```

```
Name : Asif
ID : 2345
DOB : 1990
Address : Slateford
```

In [415…
```
for i in mydict1:
    print(mydict1[i])  # Dictionary items
```

```
Asif
2345
1990
Slateford
```

Dictionary Membership

In [418…
```
mydict1 = {'Name': 'Asif', 'ID': 2345, 'DOB': 1990, 'Job': 'Analyst'}
mydict1
```

Out[418…
```
{'Name': 'Asif', 'ID': 2345, 'DOB': 1990, 'Job': 'Analyst'}
```

```
In [420…  'Name' in mydict1   # Test if a key is in a dictionary or not
```

```
Out[420…  True
```

```
In [422…  'Asif' in mydict1
```

```
Out[422…  False
```

All / Any

The all() method returns: - True - if 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 id True. if not, any() returns False

```
In [425…  mydict1 = {'Name': 'Asif', 'ID': 2345, 'DOB': 1990, 'Job': 'Analyst'}
          mydict1
```

```
Out[425…  {'Name': 'Asif', 'ID': 2345, 'DOB': 1990, 'Job': 'Analyst'}
```

```
In [427…  all(mydict1)  # will Return false as one value is false (value 0)
```

```
Out[427…  True
```

Range

```
In [430…  range(10)
```

```
Out[430…  range(0, 10)
```

```
In [432…  list(range(0,10))
```

```
Out[432…  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [434…  list(range(10, 20))
```

```
Out[434…  [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
In [436…  list(range(10, 20, 3))  # 3 is the step
```

```
Out[436…  [10, 13, 16, 19]
```

```
In [438…  range1 = range(1, 10)
          range1
```

```
Out[438…  range(1, 10)
```

```
In [440…  for i in range1:
              print(i)
```

```
1
2
3
4
5
6
7
8
9
```

In [ ]:

In [ ]: