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.

SETS CREATION

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

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

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

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

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

In [4]:
```
myset1 = {1.79,2.08,3.99,4.56,5.45} # Set of float numbers
myset1
```

Out[4]:
```
{1.79, 2.08, 3.99, 4.56, 5.45}
```

In [5]:
```
myset2 = {'Asif' , 'John' , 'Tyrion'} # Set of Strings
myset2
```

Out[5]:
```
{'Asif', 'John', 'Tyrion'}
```

In [6]:
```
myset3 = {10,20, "Hola", (11, 22, 32)} # Mixed datatypes
myset3
```

Out[6]:
```
{(11, 22, 32), 10, 20, 'Hola'}
```

In [7]:
```
myset3 = {10,20, "Hola", [11, 22, 32]} # set doesn't allow mutable items like li
myset3
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[7], line 1
----> 1 myset3 = {10,20, "Hola", [11, 22, 32]}
      2 myset3

TypeError: unhashable type: 'list'
```

In [8]:
```
myset4 = set() # Create an empty set
print(type(myset4))
```

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

In [9]:
```
my_set1 = set(('one' , 'two' , 'three' , 'four'))
my_set1
```

Out[9]:
```
{'four', 'one', 'three', 'two'}
```

LOOP THROUGH A SET

In [10]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
for i in myset:
    print(i)
```

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

In [11]:
```
for i in enumerate(myset):
    print(i)
```

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

SET MEMBERSHIP

In [12]:
```
myset
```

Out[12]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [13]:
```
'one' in myset # Check if 'one' exist in the set
```

Out[13]:
```
True
```

In [14]:
```
'ten' in myset # Check if 'ten' exist in the set
```

Out[14]:
```
False
```

In [15]:
```
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 [16]:
```
if 'eleven' in myset: # Check if 'three' exist in the set
    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 [18]:
```
myset
```

Out[18]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [19]:
```
myset.add('NINE') # Add item to a set using add() method
myset
```

Out[19]:
```
{'NINE', 'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [20]:
```
myset.update(['TEN' , 'ELEVEN' , 'TWELVE']) # Add multiple item to a set using
myset
```

Out[20]:
```
{'ELEVEN',
 'NINE',
 'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [21]:
```
myset.remove('NINE') # remove item in a set using remove() method
myset
```

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

In [22]:
```
myset.discard('TEN') # remove item from a set using discard() method
myset
```

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

In [23]:
```
myset.clear() # Delete all items in a set
myset
```

Out[23]:
```
set()
```

In [24]:
```
del myset # Delete the set object
myset
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[24], line 2
      1 del myset
----> 2 myset

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

COPY SET

In [25]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
myset
```

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

In [26]:
```
myset1 = myset # Create a new reference "myset1"
myset1
```

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

In [27]:
```
id(myset) , id(myset1) # The address of both myset & myset1 will be the same as
```

Out[27]:
```
(2475760227872, 2475760227872)
```

In [28]:
```
my_set = myset.copy() # Create a copy of the list
my_set
```

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

In [29]:
```
id(my_set) # The address of my_set will be different from myset because my_set i
```

Out[29]:
```
2475760223616
```

In [30]:
```
myset.add('nine')
myset
```

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

In [31]:
```
myset1 # myset1 will be also impacted as it is pointing to the same Set
```

Out[31]:
```
{'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

In [32]:
```
my_set # Copy of the set won't be impacted due to changes made on the original S
```

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

SET OPERATION

In [33]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
C = {8,9,10}
```

In [34]:
```
A | B # Union of A and B (All elements from both sets. NO DUPLICATES)
```

Out[34]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [35]:
```
A.union(B) # Union of A and B
```

Out[35]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [36]:
```
A.union(B, C) # Union of A, B and C.
```

Out[36]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [38]:
```
"""
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[38]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

INTERSECTION

In [40]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [41]:
```
A & B
```

Out[41]: {4, 5}

In [42]:
```
A.intersection(B) Intersection of A and B
```

```
  Cell In[42], line 1
    A.intersection(B) Intersection of A and B
                      ^
SyntaxError: invalid syntax
```

In [43]:
```
"""
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[43]: {4, 5}

DIFFERENCE

In [44]:
```
B- A # set of elements that are only in B but not in A
```

Out[44]: {6, 7, 8}

In [45]:
```
B.difference(A)
```

Out[45]: {6, 7, 8}

In [46]:
```
"""
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[46]: {6, 7, 8}

SYMMETRIC DIFFERENCE

In [47]:
```
A = {1,2,3,4,5}
```

```
B = {4,5,6,7,8}
```

In [48]:
```
A ^ B # Symmetric difference (Set of elements in A and B but not in both. "EXCLU
```

Out[48]: {1, 2, 3, 6, 7, 8}

In [49]:
```
A.symmetric_difference(B) # Symmetric difference of sets
```

Out[49]: {1, 2, 3, 6, 7, 8}

In [50]:
```
"""
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[50]: {1, 2, 3, 6, 7, 8}

SUBSET, SUPERSET & DISJOINT

In [51]:
```
A = {1,2,3,4,5,6,7,8,9}
B = {3,4,5,6,7,8}
C = {10,30,40,50}
```

In [52]:
```
B.issubset(A) # # Set B is said to be the subset of set A if all elements of B are
```

Out[52]: True

In [53]:
```
A.issuperset(B) # Set A is said to be the superset of set B if all elements of B
```

Out[53]: True

In [54]:
```
C.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common e
```

Out[54]: True

In [56]:
```
B.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common e
```

Out[56]: False

OTHER BUILTIN FUNCTIONS

In [57]:
```
A
```

Out[57]: {1, 2, 3, 4, 5, 6, 7, 8, 9}

In [58]:
```
sum(A)
```

Out[58]: 45

In [59]:
```
max(A)
```

Out[59]: 9

In [60]:
```
min(A)
```

Out[60]: 1

In [61]:
```
len(A)
```

Out[61]: 9

In [62]:
```
list(enumerate(A))
```

Out[62]: [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]

In [63]:
```
D = sorted(A,reverse=True)
D
```

Out[63]: [9, 8, 7, 6, 5, 4, 3, 2, 1]

In [64]:
```
sorted(D)
```

Out[64]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

DICTIONARY

- 1)Dictionary is a mutable data type in Python.
- 2)A python dictionary is a collection of key and value pairs separated by a colon (:) & enclosed in curly braces {}.
- 3)Keys must be unique in a dictionary, duplicate values are allowed

CREATE DICTIONARY

In [65]:
```
mydict = dict() # empty dictionary
mydict
```

Out[65]: {}

In [66]:
```
mydict = {} # empty dictionary
mydict
```

Out[66]: {}

In [67]:
```
mydict = {1:'one' , 2:'two' , 3:'three'} # dictionary with integer keys
mydict
```

Out[67]:
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [68]:
```
mydict = dict({1:'one' , 2:'two' , 3:'three'}) # Create dictionary using dict()
mydict
```

Out[68]:
```
{1: 'one', 2: 'two', 3: 'three'}
```

In [69]:
```
mydict = {'A':'one' , 'B':'two' , 'C':'three'} # dictionary with character keys
mydict
```

Out[69]:
```
{'A': 'one', 'B': 'two', 'C': 'three'}
```

In [70]:
```
mydict = {1:'one' , 'A':'two' , 3:'three'} # dictionary with mixed keys
mydict
```

Out[70]:
```
{1: 'one', 'A': 'two', 3: 'three'}
```

In [71]:
```
mydict.keys() # Return Dictionary Keys using keys() method
```

Out[71]:
```
dict_keys([1, 'A', 3])
```

In [72]:
```
mydict.values() # Return Dictionary Values using values() method
```

Out[72]:
```
dict_values(['one', 'two', 'three'])
```

In [75]:
```
mydict.items() # Access each key-value pair within a dictionary
```

Out[75]:
```
dict_items([(1, 'one'), (2, 'two'), ('A', ['asif', 'john', 'Maria'])])
```

In [76]:
```
mydict = {1:'one' , 2:'two' ,'A':['asif' , 'john' , 'Maria']} # dictionary with
mydict
```

Out[76]:
```
{1: 'one', 2: 'two', 'A': ['asif', 'john', 'Maria']}
```

In [77]:
```
mydict = {1:'one' , 2:'two' ,'A':['asif' , 'john' , 'Maria'], 'B':('Bat' , 'cat' ,
mydict
```

Out[77]:
```
{1: 'one',
 2: 'two',
 'A': ['asif', 'john', 'Maria'],
 'B': ('Bat', 'cat', 'hat')}
```

In [78]:
```
keys = {'a' , 'b' , 'c' , 'd'}
mydict3 = dict.fromkeys(keys)   # Create a dictionary from a sequence of keys
mydict3
```

Out[78]:
```
{'a': None, 'b': None, 'd': None, 'c': None}
```

In [79]:
```
keys = {'a' , 'b' , 'c' , 'd'}
value = 10
```

```
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[79]:
```
{'a': 10, 'b': 10, 'd': 10, 'c': 10}
```

In [81]:
```
keys = {'a' , 'b' , 'c' , 'd'}
value = [10,20,30]
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[81]:
```
{'a': [10, 20, 30], 'b': [10, 20, 30], 'd': [10, 20, 30], 'c': [10, 20, 30]}
```

In [82]:
```
value.append(40)
mydict3
```

Out[82]:
```
{'a': [10, 20, 30, 40],
 'b': [10, 20, 30, 40],
 'd': [10, 20, 30, 40],
 'c': [10, 20, 30, 40]}
```

ACCESSING ITEMS

In [83]:
```
mydict = {1:'one' , 2:'two' , 3:'three' , 4:'four'}
mydict
```

Out[83]:
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [84]:
```
mydict[1] # Access item using key
```

Out[84]:
```
'one'
```

In [85]:
```
mydict.get(1) # Access item using get() method
```

Out[85]:
```
'one'
```

In [86]:
```
mydict1 = {'Name':'Asif' , 'ID': 74123 , 'DOB': 1991 , 'job' :'Analyst'}
mydict1
```

Out[86]:
```
{'Name': 'Asif', 'ID': 74123, 'DOB': 1991, 'job': 'Analyst'}
```

In [87]:
```
mydict1['Name'] # Access item using key
```

Out[87]:
```
'Asif'
```

In [88]:
```
mydict1.get('job') # Access item using get() method
```

Out[88]:
```
'Analyst'
```

ADD, REMOVE & CHANGE ITEMS

In [89]:
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
mydict1
```

Out[89]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}
```

In [90]:
```
mydict1['DOB'] = 1992 # Changing Dictionary Items
mydict1['Address'] = 'Delhi'
mydict1
```

Out[90]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1992, 'Address': 'Delhi'}
```

In [91]:
```
dict1 = {'DOB':1995}
mydict1.update(dict1)
mydict1
```

Out[91]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995, 'Address': 'Delhi'}
```

In [92]:
```
mydict1['Job'] = 'Analyst' # Adding items in the dictionary
mydict1
```

Out[92]:
```
{'Name': 'Asif',
 'ID': 12345,
 'DOB': 1995,
 'Address': 'Delhi',
 'Job': 'Analyst'}
```

In [98]:
```
mydict1.pop('Job') # Removing items in the dictionary using Pop method
mydict1
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[98], line 1
----> 1 mydict1.pop('Job') # Removing items in the dictionary using Pop method
      2 mydict1

KeyError: 'Job'
```

In [95]:
```
mydict1.popitem() # A random item is removed
```

Out[95]:
```
('Address', 'Delhi')
```

In [96]:
```
mydict1
```

Out[96]:
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1995}
```

In [97]:
```
del[mydict1['ID']]
mydict1
```

Out[97]:
```
{'Name': 'Asif', 'DOB': 1995}
```

In [99]:
```
mydict1.clear() # Delete all items of the dictionary using clear method
mydict1
```

Out[99]: {}

In [100…
```
del mydict1 # Delete the dictionary object
mydict1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[100], line 2
      1 del mydict1
----> 2 mydict1

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

COPY DICTIONARY

In [101…
```
mydict = {'Name':'Asi' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki'}
mydict
```

Out[101… {'Name': 'Asi', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}

In [102…
```
mydict1 = mydict # Create a new reference "mydict1"
```

In [104…
```
id(mydict1) , id(mydict1) # The address of both mydict & mydict1 will be the same
```

Out[104… (2475766987648, 2475766987648)

In [105…
```
mydict2 = mydict.copy() # Create a copy of the dictionary
```

In [106…
```
id(mydict2) # The address of mydict2 will be different from mydict because mydic
```

Out[106… 2475768897472

In [107…
```
mydict['Address'] = 'Mumbai'
```

In [108…
```
mydict
```

Out[108… {'Name': 'Asi', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}

In [109…
```
mydict1 # mydict1 will be also impacted as it is pointing to the same dictionary
```

Out[109… {'Name': 'Asi', 'ID': 12345, 'DOB': 1991, 'Address': 'Mumbai'}

In [110…
```
mydict2 # Copy of list won't be impacted due to the changes made in the original
```

Out[110… {'Name': 'Asi', 'ID': 12345, 'DOB': 1991, 'Address': 'Hilsinki'}

LOOP THROUGH A DICTIONARY

In [115…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Address' : 'Hilsinki' , 'Jo
mydict1
```

Out[115…
```
{'Name': 'Asif',
 'ID': 12345,
 'DOB': 1991,
 'Address': 'Hilsinki',
 'Job': 'Analyst'}
```

In [119…
```
for i in mydict1:
    print(i , ':' , mydict1[i]) # Key & value pair
```

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

In [120…
```
for i in mydict1:
    print(mydict1[i]) # Dictionary items
```

```
Asif
12345
1991
Hilsinki
Analyst
```

DICTIONARY MEMBERSHIP

In [121…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[121…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [122…
```
'Name' in mydict1 # Test if a key is in a dictionary or not.
```

Out[122…
```
True
```

In [123…
```
'Asif' in mydict1 # Membership test can be only done for keys.
```

Out[123…
```
False
```

In [124…
```
'ID' in mydict1
```

Out[124…
```
True
```

In [125…
```
'Address' in mydict1
```

Out[125…
```
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 [126…
```
mydict1 = {'Name':'Asif' , 'ID': 12345 , 'DOB': 1991 , 'Job': 'Analyst'}
mydict1
```

Out[126…
```
{'Name': 'Asif', 'ID': 12345, 'DOB': 1991, 'Job': 'Analyst'}
```

In [127…
```
all(mydict1) # Will Return false as one value is false (Value 0)
```

Out[127…
```
True
```

In [ ]: