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.

Set Creation

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

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

In [2]:
```
len(myset) #Length of the set
```

Out[2]:
```
5
```

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

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 = {'rohet' , 'John' , 'Tyrion'} # Set of Strings
myset2
```

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

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]} # set doesn't allow mutable items
like li
      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 [12]:
```
myset = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}
for i in myset:
    print(i)
```

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

In [14]:
```
for i in enumerate(myset):
    print(i)
```

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

Set Membership

In [15]:
```
myset
```

Out[15]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

In [16]:
```
'one' in myset # Check if 'one' exist in the set
```

Out[16]:
```
True
```

In [17]:
```
'ten' in myset # Check if 'ten' exist in the set
```

Out[17]:
```
False
```

In [19]:
```
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 [20]:
```
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 [21]:
```
myset
```

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

In [22]:
```
myset.add('NINE') # Add item to a set using add() method
myset
```

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

In [23]:
```
myset.update(['TEN' , 'ELEVEN' , 'TWELVE']) # Add multiple item to a set using
myset
```

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

In [24]:
```
myset.remove('NINE') # remove item in a set using remove() method
myset
```

Out[24]:
```
{'ELEVEN',
 'TEN',
 'TWELVE',
 'eight',
 'five',
 'four',
 'one',
 'seven',
 'six',
 'three',
 'two'}
```

In [25]:
```
myset.discard('TEN') # remove item from a set using discard() method
myset
```

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

In [26]:
```
myset.clear() # Delete all items in a set
myset
```

Out[26]:
```
set()
```

In [27]:
```
del myset # Delete the set object
myset
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[27], line 2
      1 del myset # Delete the set object
----> 2 myset

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

Copy Set

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

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

In [29]:
```
myset1 = myset # Create a new reference "myset1"
myset1
```

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

In [30]:
```
id(myset) , id(myset1) # The address of both myset & myset1 will be the same as
```

Out[30]:
```
(2328600143328, 2328600143328)
```

In [31]:
```
my_set = myset.copy() # Create a copy of the list
my_set
```

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

In [32]:
```
id(my_set) # The address of my_set will be different from myset because my_set i
```

Out[32]:
```
2328600142432
```

In [33]:
```
myset.add('nine')
myset
```

Out[33]:
```
{'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

In [34]:
```
myset1 # myset1 will be also impacted as it is pointing to the same Set
```

Out[34]:
```
{'eight', 'five', 'four', 'nine', 'one', 'seven', 'six', 'three', 'two'}
```

In [35]:
```
my_set # Copy of the set won't be impacted due to changes made on the original S
```

Out[35]:
```
{'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}
```

Set Operation

Union

In [36]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
C = {8,9,10}
```

In [37]:
```
A | B # Union of A and B (All elements from both sets. NO DUPLICATES)
```

Out[37]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [38]:
```
A.union(B) # Union of A and B
```

Out[38]:
```
{1, 2, 3, 4, 5, 6, 7, 8}
```

In [39]:
```
A.union(B, C) # Union of A, B and C.
```

Out[39]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [40]:
```
"""
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[40]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Intersection

In [42]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [43]:
```
A & B # Intersection of A and B (Common items in both sets)
```

Out[43]:
```
{4, 5}
```

In [44]:
```
"""
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[44]: {4, 5}

Difference

In [45]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [46]:
```
A - B # set of elements that are only in A but not in B
```

Out[46]: {1, 2, 3}

In [47]:
```
A.difference(B) # Difference of sets
```

Out[47]: {1, 2, 3}

In [48]:
```
B- A # set of elements that are only in B but not in A
```

Out[48]: {6, 7, 8}

In [49]:
```
"""
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[49]: {6, 7, 8}

Symmetric Difference

In [50]:
```
A = {1,2,3,4,5}
B = {4,5,6,7,8}
```

In [51]:
```
A ^ B # Symmetric difference (Set of elements in A and B but not in both. "EXCLU
```

Out[51]: {1, 2, 3, 6, 7, 8}

In [52]:
```
A.symmetric_difference(B) # Symmetric difference of sets
```

Out[52]: {1, 2, 3, 6, 7, 8}

In [53]:
```
"""
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[53]: {1, 2, 3, 6, 7, 8}

Subset , Superset & Disjoint

In [54]:
```
A = {1,2,3,4,5,6,7,8,9}
B = {3,4,5,6,7,8}
C = {10,20,30,40}
```

In [55]:
```
B.issubset(A) # Set B is said to be the subset of set A if all elements of B are
```

Out[55]: True

In [56]:
```
 A.issuperset(B) # Set A is said to be the superset of set B if all elements of
```

Out[56]: True

In [57]:
```
C.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common e
```

Out[57]: True

In [58]:
```
B.isdisjoint(A) # Two sets are said to be disjoint sets if they have no common e
```

Out[58]: False

Other Builtin functions

In [59]:
```
A
```

Out[59]: {1, 2, 3, 4, 5, 6, 7, 8, 9}

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

Out[60]: 45

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

Out[61]: 9

In [62]:
```
min(A)
```

Out[62]: 1

In [63]:
```
len(A)
```

Out[63]: 9

In [64]:
```
list(enumerate(A))
```

Out[64]: [(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]

In [65]:
```
D= sorted(A,reverse=True)
D
```

Out[65]: [9, 8, 7, 6, 5, 4, 3, 2, 1]

In [66]:
```
sorted(D)
```

Out[66]: [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 separated by a colon (:) & enclosed in curly braces {}.
- Keys must be unique in a dictionary, duplicate values are allowed.

Create Dictionary

In [67]:
```
mydict = dict() # empty dictionary
mydict
```

Out[67]: {}

In [68]:
```
mydict = {} # empty dictionary
mydict
```

Out[68]: {}

In [69]:
```
mydict = {1:'one' , 2:'two' , 3:'three'} # dictionary with integer keys
mydict
```

Out[69]: {1: 'one', 2: 'two', 3: 'three'}

In [70]:
```
mydict = dict({1:'one' , 2:'two' , 3:'three'}) # Create dictionary using dict()
mydict
```

Out[70]: {1: 'one', 2: 'two', 3: 'three'}

In [71]:
```
mydict = {'A':'one' , 'B':'two' , 'C':'three'} # dictionary with character keys
mydict
```

Out[71]: {'A': 'one', 'B': 'two', 'C': 'three'}

In [72]:
```
mydict = {1:'one' , 'A':'two' , 3:'three'} # dictionary with mixed keys
mydict
```

Out[72]: {1: 'one', 'A': 'two', 3: 'three'}

In [73]:
```
mydict.keys() # Return Dictionary Keys using keys() method
```

Out[73]: dict_keys([1, 'A', 3])

In [74]:
```
mydict.values() # Return Dictionary Values using values() method
```

Out[74]:
```
dict_values(['one', 'two', 'three'])
```

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

Out[75]:
```
dict_items([(1, 'one'), ('A', 'two'), (3, 'three')])
```

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 [80]:
```
mydict = {1:'one' , 2:'two' , 'A':['asif' , 'john' , 'Maria'], 'B':('Bat' , 'cat
mydict
```

Out[80]:
```
{1: 'one',
 2: 'two',
 'A': ['asif', 'john', 'Maria'],
 'B': ('Bat', 'cat', 'hat')}
```

In [82]:
```
mydict = {1:'one' , 2:'two' , 'A':{'Name':'asif' , 'Age' :20}, 'B':('Bat' , 'cat
mydict
```

Out[82]:
```
{1: 'one',
 2: 'two',
 'A': {'Name': 'asif', 'Age': 20},
 'B': ('Bat', 'cat', 'hat')}
```

In [83]:
```
keys = {'a' , 'b' , 'c' , 'd'}
mydict3 = dict.fromkeys(keys) # Create a dictionary from a sequence of keys
mydict3
```

Out[83]:
```
{'c': None, 'b': None, 'a': None, 'd': None}
```

In [84]:
```
keys = {'a' , 'b' , 'c' , 'd'}
value = 10
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[84]:
```
{'c': 10, 'b': 10, 'a': 10, 'd': 10}
```

In [85]:
```
keys = {'a' , 'b' , 'c' , 'd'}
value = [10,20,30]
mydict3 = dict.fromkeys(keys , value) # Create a dictionary from a sequence of
mydict3
```

Out[85]:
```
{'c': [10, 20, 30], 'b': [10, 20, 30], 'a': [10, 20, 30], 'd': [10, 20, 30]}
```

In [86]:
```
value.append(40)
mydict3
```

Out[86]:
```
{'c': [10, 20, 30, 40],
 'b': [10, 20, 30, 40],
 'a': [10, 20, 30, 40],
 'd': [10, 20, 30, 40]}
```

Accessing Items

In [87]:
```
mydict = {1:'one' , 2:'two' , 3:'three' , 4:'four'}
mydict
```

Out[87]:
```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

In [88]:
```
mydict[1] # Access item using key
```

Out[88]:
```
'one'
```

In [89]:
```
mydict.get(1) # Access item using get() method
```

Out[89]:
```
'one'
```

In [91]:
```
 mydict1 = {'Name':'razekh' , 'ID': 9096 , 'DOB': 2005 , 'job' :'data science'}
mydict1
```

Out[91]:
```
{'Name': 'razekh', 'ID': 9096, 'DOB': 2005, 'job': 'data science'}
```

In [92]:
```
mydict1['Name'] # Access item using key
```

Out[92]:
```
'razekh'
```

In [93]:
```
mydict1.get('job') # Access item using get() method
```

Out[93]:
```
'data science'
```

Add, Remove & Change Items

In [95]:
```
mydict1 = {'Name':'Razekh' , 'ID': 9096 , 'DOB': 2005 , 'Address' : 'maharashtra
mydict1
```

Out[95]:
```
{'Name': 'Razekh', 'ID': 9096, 'DOB': 2005, 'Address': 'maharashtra'}
```

In [96]:
```
mydict1['DOB'] = 1992 # Changing Dictionary Items
mydict1['Address'] = 'Delhi'
mydict1
```

Out[96]:
```
{'Name': 'Razekh', 'ID': 9096, 'DOB': 1992, 'Address': 'Delhi'}
```

In [103…
```
dict1 = {'DOB':2005}
mydict1.update(dict1)
mydict1
```

Out[103…
```
{'Name': 'Razekh', 'DOB': 2005}
```

In [104…
```
mydict1['Job'] = 'Analyst' # Adding items in the dictionary
mydict1
```

Out[104…
```
{'Name': 'Razekh', 'DOB': 2005, 'Job': 'Analyst'}
```

In [105…
```
mydict1.pop('Job') # Removing items in the dictionary using Pop method
mydict1
```

Out[105…
```
{'Name': 'Razekh', 'DOB': 2005}
```

In [106…
```
mydict1.popitem() # A random item is removed
```
Out[106…
```
('DOB', 2005)
```
In [107…
```
mydict1
```
Out[107…
```
{'Name': 'Razekh'}
```
In [109…
```
mydict1.clear() # Delete all items of the dictionary using clear method
mydict1
```
Out[109…
```
{}
```

Copy Dictionary

In [117…
```
mydict = {'Name':'razekh' , 'ID': 9096 , 'DOB': 2005 , 'Address' : 'maharashtra'
mydict
```
Out[117…
```
{'Name': 'razekh', 'ID': 9096, 'DOB': 2005, 'Address': 'maharashtra'}
```
In [118…
```
mydict1 = mydict # Create a new reference "mydict1"
```
In [119…
```
id(mydict) , id(mydict1) # The address of both mydict & mydict1 will be the same
```
Out[119…
```
(2328621219072, 2328621219072)
```
In [120…
```
mydict2 = mydict.copy() # Create a copy of the dictionary
```
In [121…
```
id(mydict2) # The address of mydict2 will be different from mydict because mydic
```
Out[121…
```
2328621219008
```
In [122…
```
mydict['Address'] = 'Mumbai'
```
In [123…
```
mydict
```
Out[123…
```
{'Name': 'razekh', 'ID': 9096, 'DOB': 2005, 'Address': 'Mumbai'}
```
In [124…
```
mydict1 # mydict1 will be also impacted as it is pointing to the same dictionary
```
Out[124…
```
{'Name': 'razekh', 'ID': 9096, 'DOB': 2005, 'Address': 'Mumbai'}
```
In [125…
```
mydict2 # Copy of list won't be impacted due to the changes made in the original
```
Out[125…
```
{'Name': 'razekh', 'ID': 9096, 'DOB': 2005, 'Address': 'maharashtra'}
```

Loop through a Dictionary

In [126…
```
mydict = {'Name':'razekh' , 'ID': 9096 , 'DOB': 2005 , 'Address' : 'maharashtra'
mydict
```

Out[126… {'Name': 'razekh', 'ID': 9096, 'DOB': 2005, 'Address': 'maharashtra'}

In [ ]:
```
In [9096]:
for i in mydict1:
print(mydict1[i]) # Dictionary items
```

Dictionary Membership

In [129…
```
mydict = {'Name':'razekh' , 'ID': 9096 , 'DOB': 2005 , 'Address' : 'maharashtra'
mydict
```

Out[129… {'Name': 'razekh', 'ID': 9096, 'DOB': 2005, 'Address': 'maharashtra'}

In [130…
```
'Name' in mydict1 # Test if a key is in a dictionary or not.
```

Out[130… True

In [131…
```
'Razekh' in mydict1 # Membership test can be only done for keys.
```

Out[131… False

In [132…
```
'ID' in mydict1
```

Out[132… True

In [133…
```
 'Address' in mydict1
```

Out[133… True

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 [135…
```
mydict = {'Name':'razekh' , 'ID': 9096 , 'DOB': 2005 , 'Address' : 'maharashtra'
mydict
```

Out[135… {'Name': 'razekh', 'ID': 9096, 'DOB': 2005, 'Address': 'maharashtra'}

In [136…
```
all(mydict1) # Will Return false as one value is false (Value 0)
```

Out[136… True

In [ ]: