Set

- A set is an unordered collection of unique items.
- It does not allow duplicate values. Each element in a set must be unique.
- A set can contain elements of different data types, such as integers, strings, and tuples.
- Elements of a set cannot be accessed by index because sets are unordered and do not support indexing.

In [196…
```
s = {1,5,8,7,9}
print(s)
```

```
{1, 5, 7, 8, 9}
```

In [198…
```
s1= {'a','c','e','b'}
print(s1)
```

```
{'b', 'e', 'a', 'c'}
```

In [200…
```
s2 = {'t', True, 1+2j}
s2
```

Out[200…
```
{(1+2j), True, 't'}
```

In [202…
```
s3  = {1,2,3,36,7,2}
s3
```

Out[202…
```
{1, 2, 3, 7, 36}
```

Indexing

In [204…
```
s3[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[204], line 1
----> 1 s3[0]

TypeError: 'set' object is not subscriptable
```

Add

- **add()**:Add an element to a set randomly

In [206…
```
s3
```

Out[206…
```
{1, 2, 3, 7, 36}
```

In [208…
```
s3.add(100)
s3
```

Out[208… {1, 2, 3, 7, 36, 100}

In [210…
```
s3.add('hello')
s3
```

Out[210… {1, 100, 2, 3, 36, 7, 'hello'}

In [212…
```
s3.add(1+2j)
s3
```

Out[212… {(1+2j), 1, 100, 2, 3, 36, 7, 'hello'}

In [216…
```
s3.add(0)
s3
```

Out[216… {(1+2j), 0, 1, 100, 2, 3, 36, 7, 'hello'}

Note:

A set that contains the integer `1` cannot have the boolean `True` added to it because `True` is considered equivalent to `1` in Python. Similarly, a set that contains the integer `0` cannot have the boolean `False` added to it because `False` is considered equivalent to `0`. Consequently, adding `True` to a set that already contains `1` or adding `False` to a set that already contains `0` will not change the set.

In [218…
```
s3.add(True)
s3
```

Out[218… {(1+2j), 0, 1, 100, 2, 3, 36, 7, 'hello'}

In [220…
```
s3.add(False)
s3
```

Out[220… {(1+2j), 0, 1, 100, 2, 3, 36, 7, 'hello'}

Remove

- **remove()**: Removes the specified element from the set. If the element is not present, it raises a `KeyError`.

In [222…
```
s3
```

Out[222… {(1+2j), 0, 1, 100, 2, 3, 36, 7, 'hello'}

In [224…
```
s3.remove(100)# the element is present in set it will remove the element
s3
```

Out[224… {(1+2j), 0, 1, 2, 3, 36, 7, 'hello'}

In [226…
```
s3.remove(200) # the element is not present in setit will raise error
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[226], line 1
----> 1 s3.remove(200) # the element is not present in setit will raise error

KeyError: 200
```

In [228…
```
s3.discard(36)# the element is present in set it will remove the element
s3
```

Out[228… {(1+2j), 0, 1, 2, 3, 7, 'hello'}

In [230…
```
s3.discard(3000)# the element is not present in set, but  it will not any raise err
s3
```

Out[230… {(1+2j), 0, 1, 2, 3, 7, 'hello'}

Copy

- **copy()**:Return a shallow copy of a set.

In [232…
```
s4 =s3.copy()
s4
```

Out[232… {(1+2j), 0, 1, 2, 3, 7, 'hello'}

Clear

- **clear()**: Remove all elements from this set

In [235…
```
s4.clear()
s4
```

Out[235… set()

In [237…
```
len(s4)
```

Out[237… 0

pop

- **pop()**:Remove and return an arbitrary set element.

Raises KeyError if the set is empty.

In [244…
```
s3
```

Out[244…
```
{(1+2j), 0, 1, 2, 3, 7, 'hello'}
```

In [248…
```
s3.pop()
```

Out[248…
```
0
```

In [250…
```
s3.pop()
```

Out[250…
```
1
```

In [252…
```
s3
```

Out[252…
```
{(1+2j), 2, 3, 7, 'hello'}
```

in operator

The in operator in Python returns True if the specified element is present in the set; otherwise, it returns False.

In [257…
```
s3
```

Out[257…
```
{(1+2j), 2, 3, 7, 'hello'}
```

In [259…
```
4 in s3
```

Out[259…
```
False
```

In [261…
```
2 in s3
```

Out[261…
```
True
```

Update

The update() method in Python adds multiple elements to a set from an iterable (such as lists, tuples, sets, strings, or dictionaries). If any elements in the iterable are already present in the set, they will not be added again, as sets only contain unique elements. Non-iterable objects will cause a TypeError.

In [274…
```
s5 ={1,2,3}
```

In [283…
```
s5.update([44,55,66])# Updating the set with elements from a list
s5
```

Out[283… {1, 2, 3, 44, 55, 66}

In [289…
```
s5.update({7,8,9})# Updating the set with elements from another set
s5
```

Out[289… {1, 2, 3, 7, 8, 9, 44, 55, 66}

In [308…
```
s5.update((7,10,11))# Updating the set with elements from a tuple
s5
```

Out[308… {1, 10, 11, 2, 3, 44, 55, 66, 7, 8, 9, 'a', 'b', 'c', 'key1', 'key2'}

In [299…
```
s5.update('abc')# Updating the set with elements from a string
s5
```

Out[299… {1, 10, 11, 2, 3, 44, 55, 66, 7, 8, 9, 'a', 'b', 'c'}

In [303…
```
s5.update({'key1': 'value1', 'key2': 'value2'})# Updating the set with elements fro
s5
```

Out[303… {1, 10, 11, 2, 3, 44, 55, 66, 7, 8, 9, 'a', 'b', 'c', 'key1', 'key2'}

In [305…
```
s5.update(6)
s5# Output: Error: 'int' object is not iterable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[305], line 1
----> 1 s5.update(6)
      2 s5

TypeError: 'int' object is not iterable
```

Set Operations

Union:

a. The union() method in Python returns a new set that contains all the elements from the original set and all the elements from the provided iterables (such as another set, list, or tuple).

c. The union() method does not modify the original set but creates a new set with the combined elements.

b. If any elements are duplicates, they are included only once in the resulting set.

In [51]:
```
A={1,2,3,4,5}
B={4,5,6,7,8}
C={9,10}
D = [10,11,12]
```

In [10]:
```
A.union(B)
```

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

In [12]:
```
A.union(B,C)
```

Out[12]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

In [14]:
```
A.union(B,C,D)
```

Out[14]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

Notes:

The '|'operator achieves the same result as the union() method, providing a more concise way to perform the union operation.

In [16]:
```
A|B|C
```

Out[16]:
```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

Intersection:

a.The intersection() method in Python returns a new set that contains only the elements that are present in all of the sets or iterables passed as arguments.

b.This method does not modify the original set but creates a new set with the common elements.

In [47]:
```
X={1,2,3,4,5}
Y={4,5,6,7,8}
Z={7,8,9,10}
```

In [20]:
```
X.intersection(Y)# return new set with common elements
```

Out[20]:
```
{4, 5}
```

In [22]:
```
Y.intersection(Z) # return new set with common elements
```

Out[22]:
```
{7, 8}
```

In [24]:
```
X.intersection(Z)# there are no common elements so it will return empty set
```

Out[24]:
```
set()
```

Notes:

The '&'operator achieves the same result as the intersection() method, providing a more concise way to perform the intersection operation.

In [27]:
```
X & Y # return new set with common elements
```

Out[27]: {4, 5}

In [31]:
```
Y & Z  # return new set with common elements
```

Out[31]: {7, 8}

In [33]:
```
X & Z # there are no common elements so it will return empty set
```

Out[33]: set()

intersection_update

The 'intersection_update()' method in Python is used with sets to update a set with the intersection of itself and another set (or sets). This means it keeps only the elements that are present in both sets.

In [53]:
```
print(A)
print(B)
```

```
{1, 2, 3, 4, 5}
{4, 5, 6, 7, 8}
```

In [55]:
```
A.intersection_update(B)# Updates A to contain only elements present in both A and
A
```

Out[55]: {4, 5}

difference

a.The "difference method()" creates a new set that contains elements from the first set that are not present in the second set.

b.It does not modify the original sets.

c. instead of the "difference method()" we can use "-" for same operation

In [65]:
```
print(X)
print(Y)
```

```
{1, 2, 3, 4, 5}
{4, 5, 6, 7, 8}
```

In [71]:
```
X.difference(Y)  # set of elements that are only in X but not in Y
```

Out[71]: {1, 2, 3}

In [73]:
```
Y-X # set of the elemets that are only  in Y but not in Y
```

Out[73]: {6, 7, 8}

difference_update()

The "difference_update()" Removes elements from the first set that are also in the second set. Changes the original set.

In [78]:
```
print(X)
print(Y)
```

```
{1, 2, 3, 4, 5}
{4, 5, 6, 7, 8}
```

In [86]:
```
X.difference_update(Y)# X is updated to remove elements that are also in Y
print(X)
```

```
{1, 2, 3}
```

symmetric_difference():

a.The "symmetric_difference()" method creates a new set with elements that are in either of the two sets, but not in both.

b.Instead of the symmetric_difference() method, we can use the ^ operator for the same operation.

In [91]:
```
sm1 ={1,2,3,4,5}
sm2 ={4,5,6,7,8}
print(sm1)
print(sm2)
```

```
{1, 2, 3, 4, 5}
{4, 5, 6, 7, 8}
```

In [95]:
```
sm1.symmetric_difference(sm2) #Set of elements in sm1 and sm2 but not in both
```

Out[95]:
```
{1, 2, 3, 6, 7, 8}
```

In [103…
```
sm2^sm1 #Set of elements in sm2 and sm1 but not in both
```

Out[103…
```
{1, 2, 3, 6, 7, 8}
```

symmetric_difference_update()

Updates the original set to contain elements in either of the two sets but not in both.

In [106…
```
print(sm1)
print(sm2)
```

```
{1, 2, 3, 4, 5}
{4, 5, 6, 7, 8}
```

In [108…
```
sm1.symmetric_difference_update(sm2)
```

In [110…
```
sm1
```

Out[110…
```
{1, 2, 3, 6, 7, 8}
```

Subset

The issubset() method checks if all elements of one set are present in another set. It returns True if every element of the first set is also in the second set; otherwise, it returns False.

In [116…
```
A = {1,2,3,4,5,6,7,8,9}
B = {3,4,5,6,7,8}
C = {10,20,30,40}
```

In [118…
```
B.issubset(A) # Set B is said to be the subset of set A if all elements of B are pr
```

Out[118…
```
True
```

In [120…
```
B <= A # "<=" operator Checks if B set is a subset of set A.
```

Out[120…
```
True
```

In [122…
```
C.issubset(A) #Since the elements of set C are not present in set A, set C is not a
```

Out[122…
```
False
```

Superset

a. A set is considered a superset of another set if it contains all elements of that other set

b "The issuperset() method is used to check if one set is a superset of anothe."

In [131…
```
print(A)
print(B)
print(C)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{3, 4, 5, 6, 7, 8}
{40, 10, 20, 30}
```

In [133…
```
A.issuperset(B)
```

Out[133…
```
True
```

In [135…
```
B.issuperset(A)
```

Out[135…
```
False
```

Disjoint

a. Two sets are disjoint if they have no elements in common. Their intersection is an empty set.

b."isdisjoint()" method checks if two sets have no elements in common. c.Returns True if the sets have no elements in common; False if they have at least one element in common

In [139…
```
print(A)
print(B)
print(C)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{3, 4, 5, 6, 7, 8}
{40, 10, 20, 30}
```

In [145…
```
B.isdisjoint(A) # it will give false because set B elements are present in set A
```

Out[145… False

In [147…
```
C.isdisjoint(A) # it will give True because set C elements are not present in set A
```

Out[147… True

Few inbuilt functions

sum() : to find the sum of set

In [156…
```
print(A)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [158…
```
total = sum(A)
print(total)
```

```
45
```

max(): is used to find the maximum value in an set

In [162…
```
print(A)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [175…
```
maximum = max(A)
print(maximum)
```

```
9
```

min(): is used to find the minimum value in an set

In [177…
```
print(A)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [181…
```
minumum = min(A)
print(minumum)
```

```
1
```

sorted()

In [186…
```
A
```

Out[186…
```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [188…
```
sorted(A, reverse=True)
```

Out[188…
```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

In [192…
```
sorted(A)
```

Out[192…
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

frozenset():

The frozenset in Python is a built-in data type that represents an immutable set. Unlike regular sets, frozensets cannot be modified after they are created

In [197…
```
A
```

Out[197…
```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [199…
```
f_s = frozenset(A)
```

In [201…
```
f_s.add(22)# in frozenset we can't add elements so it will give error
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[201], line 1
----> 1 f_s.add(22)

AttributeError: 'frozenset' object has no attribute 'add'
```

In [203…
```
f_s.remove(9) #  in frozenset we can't remove elements so it will give error
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[203], line 1
----> 1 f_s.remove(9)

AttributeError: 'frozenset' object has no attribute 'remove'
```

In [205…
```
for i in enumerate(A):
    print(i)
```

```
(0, 1)
(1, 2)
(2, 3)
(3, 4)
(4, 5)
(5, 6)
(6, 7)
(7, 8)
(8, 9)
```

In [207…
```
list(enumerate(A))
```

Out[207…
```
[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]
```

Dictionary:

- A Python dictionary consists of a key and then an associated value.
- Duplicate keys are not allowed but values can be duplicated.
- Heterogeneous objects are allowed for both keys and values.
- Dictionaries are mutable(can change or update values).
- Indexing and slicing concepts are not applicable.
- key and value pairs separated by a colon (:) & enclosed

in curly braces {}.

In [214…
```
d ={}
type(d)
```

Out[214…
```
dict
```

In [222…
```
d = dict({'one':1, 'two':2, 'three':3}) # dictionary with dict() function
print(type(d))
print(d)
```

```
<class 'dict'>
{'one': 1, 'two': 2, 'three': 3}
```

In [324…
```
d1 ={'name':'sunder', 'age':23, 'address':'Hyderabad'}# normal dictionary
d1
```

Out[324…
```
{'name': 'sunder', 'age': 23, 'address': 'Hyderabad'}
```

In [226…
```
d2 ={1:'orange', 'A':'apple', 2:'berry'} # dictionary with mixed keys
d2
```

Out[226…
```
{1: 'orange', 'A': 'apple', 2: 'berry'}
```

keys()

The "keys()" functioned used to get the keys from the dictionary

In [232…
```
d
```

Out[232…
```
{'one': 1, 'two': 2, 'three': 3}
```

In [234…
```
d.keys()
```

Out[234…
```
dict_keys(['one', 'two', 'three'])
```

values()

The "values()" functioned used to get the values from the dictionary

In [236…
```
d.values()
```

Out[236…
```
dict_values([1, 2, 3])
```

items():

The "items()" functioned used to get the key and value pairs from dictionary

In [244…
```
d.items()
```

Out[244…
```
dict_items([('one', 1), ('two', 2), ('three', 3)])
```

fromkeys()

The fromkeys() method in Python is a class method used with dictionaries to create a new dictionary with specified keys and a common value

In [249…
```
keys = {'a','b','c'}
myd =dict.fromkeys(keys)
print(myd)
```

```
{'c': None, 'a': None, 'b': None}
```

In [251…
```
keys = {'a','b','c'}
value =10
myd =dict.fromkeys(keys,value)
print(myd)
```

```
{'c': 10, 'a': 10, 'b': 10}
```

In [253…
```
keys = {'a','b','c'}
value =[10,20,40]
```

```
myd =dict.fromkeys(keys,value)
print(myd)
```

```
{'c': [10, 20, 40], 'a': [10, 20, 40], 'b': [10, 20, 40]}
```

In [261…
```
value.append(40)
myd
```

Out[261…
```
{'c': [10, 20, 40, 40], 'a': [10, 20, 40, 40], 'b': [10, 20, 40, 40]}
```

Accessing Items

In dictionary each item is indexed by key

In [272…
```
mydict ={1:'a',2:'b',3:'c'}
mydict
```

Out[272…
```
{1: 'a', 2: 'b', 3: 'c'}
```

In [276…
```
mydict[1]
```

Out[276…
```
'a'
```

get()

In [278…
```
mydict.get(1)
```

Out[278…
```
'a'
```

Add, Remove & Change Items

Adding items in the dictionary:

In [326…
```
d1
```

Out[326…
```
{'name': 'sunder', 'age': 23, 'address': 'Hyderabad'}
```

In [328…
```
d1['job'] ='data scientist' # in d1 dictionary we adding job
d1
```

Out[328…
```
{'name': 'sunder', 'age': 23, 'address': 'Hyderabad', 'job': 'data scientist'}
```

Changing Dictionary Item

In [330…
```
d1
```

Out[330…
```
{'name': 'sunder', 'age': 23, 'address': 'Hyderabad', 'job': 'data scientist'}
```

In [332…
```
d1['age'] = 24
```

```
d1['address'] ='karimnagar'
print(d1)
```

```
{'name': 'sunder', 'age': 24, 'address': 'karimnagar', 'job': 'data scientist'}
```

update()

In [334…
```
d1
```

Out[334…
```
{'name': 'sunder', 'age': 24, 'address': 'karimnagar', 'job': 'data scientist'}
```

In [336…
```
dict1 ={'age':23}
d1.update(dict1)
d1
```

Out[336…
```
{'name': 'sunder', 'age': 23, 'address': 'karimnagar', 'job': 'data scientist'}
```

pop()

The pop() method in Python dictionaries is used to remove a key-value pair from the dictionary and return the value associated with the specified key. If the key is not found, it can raise a KeyError, unless a default value is provided.

In [338…
```
d1
```

Out[338…
```
{'name': 'sunder', 'age': 23, 'address': 'karimnagar', 'job': 'data scientist'}
```

In [340…
```
d1.pop('job')
```

Out[340…
```
'data scientist'
```

In [342…
```
d1.pop("DOB")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[342], line 1
----> 1 d1.pop("DOB")

KeyError: 'DOB'
```

popitem()

The popitem() method in Python dictionaries is used to remove and return an radom key-value pair from the dictionary.

In [344…
```
d1
```

Out[344…
```
{'name': 'sunder', 'age': 23, 'address': 'karimnagar'}
```

In [346…
```
d1.popitem()
```

Out[346… ('address', 'karimnagar')

In [348…
```
d1
```

Out[348… {'name': 'sunder', 'age': 23}

delete

In [362…
```
d1
```

Out[362… {'name': 'sunder', 'age': 23}

In [364…
```
del[d1['age']] # Removing item using del metho
d1
```

Out[364… {'name': 'sunder'}

In [366…
```
del d1 # Delete the dictionary object
d1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[366], line 2
      1 del d1 # Delete the dictionary object
----> 2 d1

NameError: name 'd1' is not defined
```

clear()

In [369…
```
d
```

Out[369… {'one': 1, 'two': 2, 'three': 3}

In [371…
```
d.clear()
d
```

Out[371… {}

copy dictionary

In [380…
```
d2
```

Out[380… {1: 'orange', 'A': 'apple', 2: 'berry'}

In [384…
```
d3 = d2
d3
```

Out[384… {1: 'orange', 'A': 'apple', 2: 'berry'}

In [386…
```
d4 = d3.copy()
d4
```

Out[386… {1: 'orange', 'A': 'apple', 2: 'berry'}

In [388…
```
d2['A']='bread'
```

In [390…
```
d2
```

Out[390… {1: 'orange', 'A': 'bread', 2: 'berry'}

In [392…
```
d3
```

Out[392… {1: 'orange', 'A': 'bread', 2: 'berry'}

In [394…
```
d4
```

Out[394… {1: 'orange', 'A': 'apple', 2: 'berry'}

loop through a dictionary

In [403…
```
d2
```

Out[403… {1: 'orange', 'A': 'bread', 2: 'berry'}

In [405…
```
for i in  d2:
    print(i)
```

```
1
A
2
```

In [407…
```
for i in d2:
    print(i, d2[i])
```

```
1 orange
A bread
2 berry
```

Dictionary Membership

In [413…
```
d2
```

Out[413… {1: 'orange', 'A': 'bread', 2: 'berry'}

In [415…
```
1 in d2 # Test if a key is in a dictionary or not.
```

Out[415… True

In [417…
```
'orange' in d2 # Membership test can be only done for keys.
```

Out[417…
```
False
```

In [ ]:
```

```