IOTA ACADEMY

Become Job Ready...

Sets

- A set is an **unordered collection** of items.
- Every element is **unique** (no duplicates).
- The set itself is **mutable**. We can add or remove items from it.
- Sets can be used to perform **mathematical set operations** like union, intersection, symmetric difference etc.
- Sets only store hashable objects (immutable objects) like tuple, integer, float, string etc.
- Sets are not subscriptable(indexing not possible).

More on hashable/immutable objects: https://realpython.com/lessons/immutable-vs-hashable/#:~:text=Python%20sets%20can%20only%20include,strings%2C%20integers%2C%20and%20Booleans.

Set Creation

In [33]:
```python
# set of integers
s = {1, 2, 3,3}
print(s)

# print type of s
print(type(s))
```
```
{1, 2, 3}
<class 'set'>
```

In [34]:
```python
# set doesn't allow duplicates. They store only one instance.
s = {1, 3, 5,1,-1, 4,'a','b', 'c', 'd'}
print(s)
```
```
{1, 'b', 3, 4, 5, 'd', 'a', 'c', -1}
```

In [35]:
```python
# we can make a set from a tuple
s = set(('12ertui',))
print(s)
```
```
{'12ertui'}
```

In [36]:
```python
# make set from list
s = set([3,4,57,8])
print(s)
```
```
{8, 57, 3, 4}
```

initialize a set with set() method

We cannot create a blank set using curly brackets '{}'. This will create a blank dictionary (another data type in Python.)

To create blank set use: set() function (like type casting)

In [37]:
```python
#initialize a set  with set() method
s = set()

print(type(s))
```
```
<class 'set'>
```

In [38]:
```python
s = {}  #this will give dictionary and not set
print(type(s))
```
```
<class 'dict'>
```

set object doesn't support indexing because they are unordered

In [39]:
```python
s = {1,2,3, 7, 8, 4,5,6}

print(s[0]) # will get TypeError
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[39], line 3
      1 s = {1,2,3, 7, 8, 4,5,6}
----> 3 print(s[0])

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

Set Methods

Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two or more sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with another set, or any other iterable

In [40]:
```python
# printing set methods
print(dir(set()))
```
```
['__and__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '_
_format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__iand__', '__init__', '__init_subclass__', '_
_ior__', '__isub__', '__iter__', '__ixor__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__r
and__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof_
_', '__str__', '__sub__', '__subclasshook__', '__xor__', 'add', 'clear', 'copy', 'difference', 'difference_upda
te', 'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove',
'symmetric_difference', 'symmetric_difference_update', 'union', 'update']
```

Add element to a Set

`add()`

The add() method adds a given element to a set. If the element is already present, it doesn't add any element.

Syntax: `set.add(object)`

In [41]:
```python
s = {1, 3}
print(s)
s.add(5)  # changes will happen in-place
print(s)
```
```
{1, 3}
{1, 3, 5}
```

`update()`

Update the set with another set, or any other iterable

Syntax: `set.update(iterale)`

In [42]:
```python
# we can add single element using add() method and add multiple elements using update() method
s = {1, 3}
print(s)
s.update([6,7,8])
print(s)
```
```
{1, 3}
{1, 3, 6, 7, 8}
```

In [43]:
```python
# The update() method can take any number of arguments.
# add list and set
s.update([6,7,8,9],[11,12,13],{44,55,66})
print(s)
```
```
{1, 66, 3, 6, 7, 8, 9, 11, 12, 13, 44, 55}
```

In [44]:
```python
s1 = {22,33,44}
s1.update(45)  # it will give TypeError because integer is not iterable
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 2
      1 s1 = {22,33,44}
----> 2 s1.update(45)

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

Remove elements from a Set

A particular item can be removed from set using methods `discard()` and `remove()`.

`discard()`

Remove an element from a set if it is a member.

If the element is not a member, do nothing.

Syntax: `set.discard(object)`

In [45]:
```python
s = {1, 2, 3, 5, 4}
print(s)

s.discard(4)    # 4 is removed from set s

print(s)
```
```
{1, 2, 3, 4, 5}
{1, 2, 3, 5}
```

In [46]:
```python
#discard an element not present in a set s

s.discard(7)   # no error
print(s)
```
```
{1, 2, 3, 5}
```

`remove()`

Remove an element from a set; it must be a member.

If the element is not a member, raise a KeyError.

Syntax: `set.remove(object)`

In [47]:
```python
# remove an element
s.remove(3)  # 3 is removed

print(s)
```
```
{1, 2, 5}
```

In [48]:
```python
# remove an element not present in a set s

s.remove(7) # will get KeyError
```
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[48], line 3
      1 # remove an element not present in a set s
----> 3 s.remove(7)

KeyError: 7
```

In [49]:
```python
# example
s1 = {1,2,3}
s1.remove(1)
print(s1)
s1.remove(1)
```
```
{2, 3}
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[49], line 5
      3 s1.remove(1)
      4 print(s1)
----> 5 s1.remove(1)

KeyError: 1
```

`pop()`

Remove and return an arbitrary set element. Raises KeyError if the set is empty.

Syntax: `set.pop()`

In [50]:
```python
# we can remove item using pop() method

s = {1, 2, 3, 5, 4, 0, 7, -1}

x = s.pop() # remove random element

print(s)
print(x)
```
```
{1, 2, 3, 4, 5, 7, -1}
0
```

`clear()`

Remove all elements from this set.

Syntax: `set.clear()`

In [51]:
```python
s = {1, 5, 2, 3, 6}

s.clear()   # remove all items in set using clear() method

print(s)
```
```
set()
```

Python Set Operations

Explaination using Venn Diagram

Union of Sets

In [52]:
```python
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

# union of 2 sets using | operator (pipe operator)

print(set1 | set2)  # Return a set containing the union of sets
```
```
{1, 2, 3, 4, 5, 6, 7}
```

In [53]:
```python
# another way of getting union of 2 sets

print(set1)
print(set2)
set_union = set1.union(set2) # Return a set containing the union of sets

print(set_union)
```
```
{1, 2, 3, 4, 5}
{3, 4, 5, 6, 7}
{1, 2, 3, 4, 5, 6, 7}
```

Intersection of Sets

In [54]:
```python
#intersection of 2 sets using & operator

print(set1)
print(set2)

set3 = set1 & set2  # ampersand operator
print(set3)
```
```
{1, 2, 3, 4, 5}
{3, 4, 5, 6, 7}
{3, 4, 5}
```

In [55]:
```python
#intersection of 2 sets using method

print(set1)
print(set2)

set3 = set1.intersection(set2)    # returns a set that is intersection of two or more sets.
print(set3)
```
```
{1, 2, 3, 4, 5}
{3, 4, 5, 6, 7}
{3, 4, 5}
```

In [56]:
```python
# use intersection_update method

print(set1)
print(set2)

set1.intersection_update(set2)   # update the given set, instead of returning the new set.
print(set1.intersection_update(set2)) #inplace change, hence returns None

print(set1)
print(set2)
```
```
{1, 2, 3, 4, 5}
{3, 4, 5, 6, 7}
None
{3, 4, 5}
{3, 4, 5, 6, 7}
```

Set Difference

In [57]:
```python
# set Difference: set of elements that are only in set1 but not in set2
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

print(set1 - set2)
```
```
{1, 2}
```

In [58]:
```python
 #use diffrence method

print(set1)
print(set2)
print(set1.difference(set2))
```
```
{1, 2, 3, 4, 5}
{3, 4, 5, 6, 7}
{1, 2}
```

Symmetric Difference

Symmetric difference: set of elements in both set1 and set2 OR except those that are common in both.

In [59]:
```python
# Symmetric difference: uncommon items
# use ^ operator (XOR operator)

set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

print(set1^set2)
```
```
{1, 2, 6, 7}
```

In [60]:
```python
# use symmetric_difference method
print(set1.symmetric_difference(set2))
```
```
{1, 2, 6, 7}
```

Subset and Superset

In [61]:
```python
# find issubset()
x = {"a","b","c","d","e"}
y = {"c","d"}

# check x is subset of y
print("set 'x' is subset of 'y' ?", x.issubset(y))

# check y is subset of x
print("set 'y' is subset of 'x' ?", y.issubset(x))
```
```
set 'x' is subset of 'y' ? False
set 'y' is subset of 'x' ? True
```

In [62]:
```python
print(x)
print(y)

#check x is superset of y

print(x.issuperset(y))
```
```
{'b', 'd', 'a', 'e', 'c'}
{'d', 'c'}
True
```

Frozen Sets

Frozen sets has the characteristics of sets, but it can't be changed once it's assigned. While tuple are immutable lists,

> frozen sets are immutable sets

Frozensets can be created using the function

> **frozenset()**

Sets being mutable are unhashable, so they can't be used as dictionary keys. On the other hand,

> frozensets are hashable and can be used as keys to a dictionary.

This datatype supports methods like
copy(), difference(), intersection(), isdisjoint(), issubset(), issuperset(), symmetric_difference() and union().
Being immutable it does not have method that add or remove elements.

In [63]:
```python
print(dir(frozenset()))
```
```
['__and__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '_
_format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '_
_le__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__repr_
_', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__',
'__xor__', 'copy', 'difference', 'intersection', 'isdisjoint', 'issubset', 'issuperset', 'symmetric_differenc
e', 'union']
```

In [64]:
```python
set1 = frozenset((1, 2, 3, 4))
set2 = frozenset([2, 3, 13, 14])

print(set2)
print(set1)
```
```
frozenset({2, 3, 13, 14})
frozenset({1, 2, 3, 4})
```

In [65]:
```python
print(set1[1]) # frozen set doesn't support indexing
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[65], line 1
----> 1 print(set1[1])

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

In Python, the subscripting is nothing but indexing. Strings, lists, tuples, and dictionaries fall in subscriptable category.

In [66]:
```python
print(set1 | set2) #union of 2 sets
```
```
frozenset({1, 2, 3, 4, 13, 14})
```

In [67]:
```python
#intersection of two sets
print(set1 & set2)

#or
print(set1.intersection(set2))
```
```
frozenset({2, 3})
frozenset({2, 3})
```

In [68]:
```python
#symmetric difference
print(set1 ^ set2)

#or
print(set1.symmetric_difference(set2))
```
```
frozenset({1, 4, 13, 14})
frozenset({1, 4, 13, 14})
```

That's Great