Sets

Sets

- Mathematically a set is a collection of items not in any particular order.
- A Python set is similar to this mathematical definition with below additional conditions.
- A set is an unordered collection of items.
- The elements in the set cannot be duplicates.
- A set itself may be modified but the elements in the set are immutable (cannot be modified).
- There is no index attached to any element in a python set. So they do not support any indexing or slicing operation.
- In Python sets are written with curly brackets { }.

Creating a SET

*** A set can be defined with curly braces ({})**

```
>>>x = {1, 2, 3, 4}

>>> y = {'apple', 'ball', 'cat'}
```

*** you can also define a set with the built-in set() function:**

```
>>> x = set(['foo', 'bar', 'baz', 'foo', 'qux'])
>>> x
{'qux', 'foo', 'bar', 'baz'}

>>> x = set(('foo', 'bar', 'baz', 'foo', 'qux'))
>>> x
{'qux', 'foo', 'bar', 'baz'}
```

Creating a SET

```
>>> x1 = set(‘spam')      # Prepare set from a string
>>> print (x1)
{'s', 'a', 'p', 'm'}

>>> x2 = set(“sspam’)     # Duplicate values will be ignored
>>> print(x2)
{'s', 'm', 'p', 'a'}

>>>x3 = {"apple", "banana", "cherry", "apple"}
>>>print(x3)
{'cherry', 'apple', 'banana'}
```

Get the Length of a Set

To determine how many items a set has, use the len() method.

```
>>> y1 = {"apple", "banana", "cherry"}

>>> print(len(y1))

3
```

Get the Length of a Set

To determine how many items a set has, use the len() method.

```
>>> y1 = {"apple", "banana", "cherry"}

>>> print(len(y1))

3
```

Get the Length of a Set

- Set items can be of any data type
- A set can contain different data types

```
>>> set1 = {"apple", "banana", "cherry"}

>>> set2 = {1, 5, 7, 9, 3}

>>> set3 = {True, False, False}

>>> set1 = {"abc", 34, True, 40, "male"}
```

Add Items - add()

- Once a set is created, you cannot change its items, but you can add new items.
- To add one item to a set use the add() method.

```
>>>a1 = {"apple", "banana", "cherry"}
>>>a1.add("orange")
>>>print(a1)
{'orange', 'apple', 'banana', 'cherry'}

>>>Days=set(["Mon","Tue","Wed","Thu","Fri","Sat","Sun"])
>>>print(Days)
{'Mon', 'Sat', 'Tue', 'Sun', 'Wed', 'Fri', 'Thu'}
```

```
>>> S.add({'a':1})
TypeError: unhashable type: 'dict'
```

Works for tuples:

```
>>> S.add((1, 2, 3))
>>> S
{1.23, (1, 2, 3)}
```

```
>>> S | {(4, 5, 6), (1, 2, 3)}

{1.23, (4, 5, 6), (1, 2, 3)}

>>> (1, 2, 3) in S      # Check for tuple as a whole
True

>>> (1, 4, 3) in S
False
```

Add Sets - update()

- To add items from another set into the current set, use the update() method.
- Add elements from tropical into thisset:

```
>>>q1 = {"apple", "banana", "cherry"}
>>>q2 = {"pineapple", "mango", "papaya"}
>>>q1.update(q2)
>>> print(q1)

{'pineapple', 'banana', 'apple', 'papaya', 'cherry', 'mango'}
```

Add Sets - update()

- To add elements of a list to at set:

```
>>>q1 = {"apple", "banana", "cherry"}
>>>l1 = ["kiwi", "orange"]
>>>q1.update(l1)
>>>print(q1)

{'apple', 'cherry', 'banana', 'orange', 'kiwi'}
```

Remove Item - clear()

- All elements will **removed** from a set.

```
>>> cities = {"Stuttgart", "Konstanz", "Freiburg"}
>>> cities.clear()


>>> cities
set()                    # shows empty set
```

Remove Item - discard(item)

item will be removed from the set, if it is contained in the set and nothing will be done otherwise

```
>>> x = {"a","b","c","d","e"}
>>> x.discard("a")

>>> x
{'c', 'b', 'e', 'd'}

>>> x.discard("z")

>>> x
{'c', 'b', 'e', 'd'}
```

Remove Item - remove(item)

works like **discard(),** but if item is **not a member** of the set, a KeyError will be raised.

```
>>> x = {"a","b","c","d","e"}
>>> x.remove("a")
>>> x
{'c', 'b', 'e', 'd'}

>>> x.remove("z")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'z'
```

Remove Item - pop()

- pop() removes the last item and returns the removed item.
- Remember that sets are unordered, so you will not know what item that gets removed.
- The method raises a KeyError if the set is empty

```
>>> x = {"a","b","c","d","e"}
>>> x.pop()
'a‘

>>> x.pop()
'c'
```

Remove Item - del()

- The del keyword will delete the set completely:

```
>>>x = {"apple", "banana", "cherry"}
>>>del x
>>>print(x)

Traceback (most recent call last):
  File "demo_set_del.py", line 5, in <module>
    print(thisset) #this will raise an error because the set no
longer exists
NameError: name 'thisset' is not defined
```

Set Operations

Let S1 = {1, 2, 3, 4}

Union (|)

S2 = **{1, 5, 3, 6} | S1**

print(S2) **# prints {1, 2, 3, 4, 5, 6}**

Intersection (&)

S2 = **S1 & {1, 3,7}**

print(S2) **# prints {1, 3}**

Set Operations

Let S1 = {1, 2, 3, 4}

Difference (-)

S2 = **S1 - {1, 3, 4}**

print(S2) **# prints {2}**

```
>>> {1, 2, 3} | {3, 4}       #union
{1, 2, 3, 4}


>>> {1, 2, 3} | [3, 4]
TypeError: unsupported operand type(s) for |: 'set' and 'list'

>>> {1, 2, 3} | set([3, 4])     #Convert list to set and work
{1,2,3,4}
```

* a set cannot have mutable elements like lists, sets or dictionaries as its elements.

Set Methods - union()

```
>>> {1, 2, 3}.union([3, 4])
{1,2,3,4}

>>> {1, 2, 3}.union({3, 4})
{1,2,3,4}

>>>x = {"apple", "banana", "cherry"}
>>>y = {"google", "microsoft", "apple"}
>>>z = x.union(y)
>>>print(z)
{'banana', 'microsoft', 'google', 'apple', 'cherry'}
```

Set Methods - intersection()

- Returns a set, that is the intersection of two or more sets

```
>>>x = {"apple", "banana", "cherry"}
>>> y = {"google", "microsoft", "apple"}

>>> z = x.intersection(y)

>>> print(z)
{'apple'}
```

Set Methods - difference()

- Returns a set containing the difference between two or more sets

```
>>>x = {"apple", "banana", "cherry"}
>>>y = {"google", "microsoft", "apple"}

>>>z = x.difference(y)

>>>print(z)
{'cherry', 'banana'}
```

Set Methods - difference()

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'baz', 'qux', 'quux'}

>>> x1.difference(x2)
{'foo', 'bar'}

>>> x1 - x2
{'foo', 'bar'}
```

Set Methods - difference()

```
>>> a = {1, 2, 3, 30, 300}
>>> b = {10, 20, 30, 40}
>>> c = {100, 200, 300, 400}

>>> a.difference(b, c)
{1, 2, 3}

>>> a - b - c
{1, 2, 3}
```

When multiple sets are specified, the operation is performed from left to right. In the example above, a - b is computed first, resulting in {1, 2, 3, 300}. Then c is subtracted from that set, leaving {1, 2, 3}

Set Methods - **isdisjoint()**

- This method returns True if two sets have a null intersection. Return True if no items in set x is present in set y

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "facebook"}

z = x.isdisjoint(y)

print(z)
```

True

Set Methods - issubset()

- Return True if all items in set x are present in set y
- x.issubset(y) returns True, if x is a subset of y

```
x = {"a", "b", "c“}
y = {"f", "e", "d", "c", "b", "a"}
z = x.issubset(y)
print(z)
```

TRUE

- Set Methods - issubset()

```
>>> x1 = {'foo', 'bar'}
>>> x2 = {'foo', 'bar', 'baz'}
>>> x1 < x2
True

>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'foo', 'bar', 'baz'}
>>> x1 < x2
False

>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'foo', 'bar', 'baz'}
>>> x1 <= x2
True
```

Set Methods - **issuperset()**

- x**.issuperset(y)** returns True, if x is a superset of y.
- Return True if all items set y are present in set x

```
>>>x = {"f", "e", "d", "c", "b", "a"}

>>> y = {"a", "b", "c"}

>>> z = x.issuperset(y)

>>> print(z)

TRUE
```

Set Methods - **issuperset()**

```
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'foo', 'bar'}
>>> x1 > x2
True
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'foo', 'bar', 'baz'}
>>> x1 > x2
False
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'foo', 'bar', 'baz'}
>>> x1 > =x2
True
```

```
>>> x = {"a","b","c","d","e"}
>>> y = {"c","d"}
>>> x.issuperset(y)
True
```

Immutable constraints and frozen sets

- Can only contain immutable (a.k.a. “hashable”) object types
- **lists and dictionaries cannot be embedded in sets**, but **tuples can** if you need to store compound values.
- Tuples **compare by their full values** when used in set operations:

```
>>> S = {1.23}

>>> S.add([1, 2, 3])
TypeError: unhashable type: 'list'
```

Copy

Creates a **shallow copy,** which is returned.

```
>>> more_cities = {"Winterthur","Schaffhausen","St. Gallen"}

>>> cities_backup = more_cities.copy()

>>> more_cities.clear()

>>> cities_backup  # copied value is still available
{'St. Gallen', 'Winterthur', 'Schaffhausen‘}
```

symmetric_difference()

The symmetric difference of two sets A and B is the set of elements that are in either A or B, but not in their intersection.

```
>>>A = {'a', 'b', 'c', 'd'}
>>> B = {'c', 'd', 'e' }
>>> C = {}
>>> print(A.symmetric_difference(B))
{'b', 'a', 'e'}

>>> print(B.symmetric_difference(A))
{'b', 'e', 'a'}

>>> print(A.symmetric_difference(C))
{'b', 'd', 'c', 'a'}

>>> print(B.symmetric_difference(C))
{'d', 'e', 'c'}
```

Set comprehensions

- run a loop and collect the result of an expression on each iteration
- result is a new set you create by running the code, with all the normal set behaviour

```
>>> {x ** 2 for x in [1, 2, 3, 4]}
{16, 1, 4, 9}

>>> {x for x in 'spam'}
{'m', 's', 'p', 'a'}
```

```
>>> S = {c * 4 for c in 'spam'}

>>> print(S)
{'pppp','aaaa','ssss', 'mmmm'}



>>>> S = {c * 4 for c in 'spamham'}
{'pppp','aaaa','ssss', 'mmmm','hhhh'}

>>>S | {'mmmm', 'xxxx'}
{'pppp', 'xxxx', 'mmmm', 'aaaa', 'ssss'}

>>> S & {'mmmm', 'xxxx'}
{'mmmm'}
```

Get Input from user for SET

```
people = set()

for i in range(5):
    people.add(input())

print("\nSet after adding element:", end = " ")
print(people)
```

Problem:

An University has published the results of the term end examination conducted in April. List of failures in physics, mathematics, chemistry and computer science is available. Write a program to find the number of failures in the examination. This includes the count of failures in one or more subjects

PAC For University Result Problem

Input	Processing	Output
Read the register number of failures in Maths, Physics, Chemistry and Computer Science	Create a list of register numbers who have failed in one or more subjects Count the count of failures	Print Count

Pseudocode

```
READ maths_failure, physics_failure, chemistry_failure and cs_failure
Let failure be empty
FOR each item in maths_failure
        ADD item to failure
FOR each item in physics_failure
        IF item is not in failure THEN
                ADD item to failure
        END IF
FOR each item in chemistry_failure
        IF item is not in failure THEN
                ADD item to failure
        END IF
FOR each item in cs_failure
        IF item is not in failure THEN
                ADD item to failure
        END IF
SET count = 0
FOR each item in failure
        count = count + 1
PRINT count
```