Notes On Sets In Python

> ## Compiled by- Md Farmanul Haque
>
> ## Technical Trainer,
>
> GLA University,
>
> Mathura

Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Create a Set:

```
set_example = {"python", "machine learning", "data science"}
print(set_example)
```

```
{'python', 'data science', 'machine learning'}
```

Note: Sets are unordered, so we cannot be sure in which order the items will appear.

We cannot access items in a set by referring to an index, since sets are unordered the items has no index.

But we can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.

Loop through the set, and print the values:

```
set_example = {"python", "machine learning", "data science"}
for course in set_example:
  print("Course\n",course)
```

```
Course
 python
Course
 data science
Course
 machine learning
```

Check if element is present in the set:

```
print("python" in set_example)
```

```
True
```

Change Items

Once a set is created, you cannot change its items, but you can add new items.

Add Items

- To add one item to a set use the add() method.
- To add more than one item to a set use the update() method.

```
set_example.add("deep learning")
print(set_example)
```

```
{'python', 'data science', 'deep learning', 'machine learning'}
```

```
set_example.update(["NLP", "Computer Vision", "Big DATA"])
print(set_example)
```

```
{'Computer Vision', 'Big DATA', 'python', 'data science', 'deep learning', 'NLP', 'm
```

Get the Length of a Set

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

```
print(len(set_example))
```

```
7
```

Remove Item

To remove an item in a set, use the remove(), or the discard() method.

```
#Remove "Big DATA" by using the remove() method:
set_example.remove("Big DATA")
print(set_example)
```

```
{'Computer Vision', 'python', 'data science', 'deep learning', 'NLP', 'machine learn
```

```
#Remove "Computer Vision" by using the discard() method:
set_example.discard("Computer Vision")
print(set_example)
```

```
{'python', 'data science', 'deep learning', 'NLP', 'machine learning'}
```

You can also use the **pop()**, method to remove an item, but this method will remove the last item. Remember that sets are unordered, so we will not know what item that gets removed.

The return value of the **pop()** method is the removed item.

```
x = set_example.pop()

print(x)

print(set_example)
```

```
python
{'data science', 'deep learning', 'NLP', 'machine learning'}
```

Empties set using clear() method.

```
set_example_1 = {10,20,30,40}
print(set_example_1)
set_example_1.clear()
print(set_example_1)
```

```
{40, 10, 20, 30}
set()
```

Delete set using del keyword

```
set_example_2 = {10,20,30,40}
print(set_example_2)
del set_example_2
```

```
{40, 10, 20, 30}
```

Join Two Sets

There are several ways to join two or more sets in Python.

We can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another:

The **union()** method returns a new set with all items from both sets:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set3 = set1.union(set2)
print(set3)
```

```
{1, 2, 'a', 3, 'c', 'b'}
```

The **update()** method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}

set1.update(set2)
print(set1)
```

```
{1, 2, 'a', 3, 'c', 'b'}
```

Note: Both union() and update() will exclude any duplicate items.

The set() Constructor

Using the set() constructor to make a set:

```
set_example_3 = set(("ml","ds","nlp"))
print(set_example_3)
```

```
{'nlp', 'ds', 'ml'}
```

Set v/s Dictionary

Sets are different in the sense that order does not matter and these will be used in cases where order does not matter.

Sets are especially useful when you need to ensure that an item only appears once in the set.

Dictionaries are used to store key, value pairs and are used when you want to easily find a value using a key, just like in a dictionary.

For example, you could store a list of items and links to more information about these items in a dictionary.

Issubset

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

z = x.issubset(y) # The issubset() method returns True if all items in the set exists in t
```

```
print(z)
```

```
True
```

issuperset()

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

z = x.issuperset(y) # The issuperset() method returns True if all items in the specified s



print(z)
```

```
True
```

Copy

```
s={1,2,3}
s2=s.copy() # creates a copy of the set same as writing s2=s
print(s2)
```

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

InterSection

```
x = {"a", "b", "c",}
y = {"g", "m", "a"}

z = x.intersection(y)
#Return a set that contains the items that exist in both set x, and set y
#returns only the common elements in both the sets


print(z)
```

```
{'a'}
```

The difference() method returns a set that contains the difference between two sets.

Meaning: The returned set contains items that exist only in the first set, and not in both sets.

```
x = {"a", "b", "c"}
y = {"g", "m", "a"}

z = x.difference(y)
print("x is",x)
#since 'a' is common z will contain all elements from x except 'a'
```

```
#since "a" is common z will contain all elements from x except "a"
print(z)
```

```
x is {'c', 'b', 'a'}
{'c', 'b'}
```

```
x = {"a", "b", "c"}
y = {"g", "m", "a"}

x.difference_update(y) #The difference_update() method removes the items that exist in bot

#The difference_update() method is different from the difference() method,
#because the difference() method returns a new set, without the unwanted items,
#and the difference_update() method removes the unwanted items from the original set.
print(x)
```

```
{'c', 'b'}
```

The discard() method removes the specified item from the set.

This method is different from the remove() method, because the remove() method will raise an error if the specified item does not exist, and the discard() method will not.

```
thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)
```

```
{'cherry', 'apple'}
```

```
x = {"a", "b", "c"}
y = {"g", "m", "a"}

x.intersection_update(y)
#just like difference update this will do the intersection and store in x
print(x)
```

```
{'a'}
```

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

z = x.isdisjoint(y) #returns true if there is no intersection else returns false

print(z)
```

```
True
```

The symmetric_difference() method returns a set that contains all items from both set, but not the items that are present in both sets.

Meaning: The returned set contains a mix of items that are not present in both sets.

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

z = x.symmetric_difference(y)  # the formula for symmetric difference is union of both set

print(z)
```

```
{'microsoft', 'google', 'cherry', 'banana'}
```

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

x.symmetric_difference_update(y) #does the symmetric difference and stores in x

print(x)
```

```
{'banana', 'google', 'cherry', 'microsoft'}
```

✓ 0s completed at 7:03 PM