Sets

July 25, 2022

Python Data Structures - Sets

Sets

Python set is an unordered collection of unique items comma-separated sequence of items in curly braces { }. They are commonly used for computing mathematical operations such asunion, intersection, difference, and symmetric difference.

Properties of Python Sets :

Sets are unordered – Items stored in a set aren't kept in any particular order.

Set items are unique – don't allow duplicates. They are automatically removed during the creation of a set.

Sets are changeable (mutable) - A set itself may be modified, but the items in the set must be immutable type.

Sets are unindexed – You cannot access set items by referring to an index.

Sets are iterable.

Creating a Set

[1]:
```
#Set in Python can be created by just placing the sequence inside the { }␣
↪separated by commas (,).

# Set having integers
evenNumber = {2, 4, 6}
print(evenNumber,"-->",type(evenNumber))

# set with mixed datatypes
stringAndNum = {"one", 2, 3}
print(stringAndNum,"-->",type(stringAndNum))
```

```
{2, 4, 6} --> <class 'set'>
{2, 3, 'one'} --> <class 'set'>
```

[2]:
```
#Empty curly braces {} will make an empty dictionary in Python.
#To make a set without any elements, we use the set() function without any␣
↪argument.

# Distinguish set and dictionary while creating empty set
# initialize a with {}
a = {}

# check data type of a
print(a,"-->",type(a))

# initialize a with set()
a = set()

# check data type of a
print(a ,"-->", type(a))
```

```
{} --> <class 'dict'>
set() --> <class 'set'>
```

[3]:
```
#set() takes an iterable as an argument.
#In python, string, lists, and dictionaries are iterable so you can pass them␣
↪inside set()

lst = [1, 2, 'three']
listToSet = set(lst)
print(listToSet,"-->", type(listToSet))

dictToSet = set({"one": 1, "two": 2})
print(dictToSet,"-->", type(dictToSet))

strToSet = set('python')
print(strToSet,"-->", type(strToSet))
```

```
{1, 2, 'three'} --> <class 'set'>
{'one', 'two'} --> <class 'set'>
{'n', 'o', 't', 'y', 'h', 'p'} --> <class 'set'>
```

[4]:
```
# set cannot have duplicates
my_set = {1, 2, 3, 4, 3, 2}
print(my_set)
```

```
{1, 2, 3, 4}
```

[5]:
```
#Remove duplicate values from list
lst = [1, 2, 'three',1, 2]
listToSet = set(lst)
lst=list(listToSet )
print(lst)
```

```
[1, 2, 'three']
```

```
[6]: #A set itself may be modified, but the items in the set must be  immutable type
     # This wont work
     mySetOne = {1, 2, [3, 4]}
     print(mySetOne)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-7e54b5d42354> in <module>
      1 #A set itself may be modified, but the items in the set must be ␣
 ↪immutable type
      2 # This wont work
----> 3 mySetOne = {1, 2, [3, 4]}
      4 print(mySetOne)

TypeError: unhashable type: 'list'
```

```
[7]: # This will work
     mySetTwo = {1, 2, (3, 4)}
     print(mySetTwo)
```

```
{1, 2, (3, 4)}
```

Adding Items in a Python Set

There are two methods to add elements in a set

We can add a single element using the add( ) method

We can add multiple elements using the update( ) method.

```
[8]: # set of cities
     cities = {'Madrid', 'Valencia', 'Barcelona'}

     # add an element to a set
     cities.add('Munich')
     print(cities)
     # {'Valencia', 'Barcelona', 'Munich', 'Madrid'}

     # add an element to the set (already present) - the element is not added, since␣
      ↪duplicates are not allowed
     cities.add('Madrid')
     print(cities)
     # {'Valencia', 'Barcelona', 'Munich', 'Madrid'}
```

```
{'Munich', 'Valencia', 'Madrid', 'Barcelona'}
{'Munich', 'Valencia', 'Madrid', 'Barcelona'}
```

```
[9]: #add mutiple items using update
     initialSet = {1, 2}
```

```python
print("Initial Set : ",initialSet)

#update() takes an iterable like string, list, dictionary as an argument.
toAdd = [4, 5]
initialSet.update(toAdd)
print("Updated Set : ",initialSet)

# add list and set
initialSet.update([4, 5], {1, 6, 8})
print("Updated Set : ",initialSet)
```

```
Initial Set :  {1, 2}
Updated Set :  {1, 2, 4, 5}
Updated Set :  {1, 2, 4, 5, 6, 8}
```

Delete Set Elements

remove( ) - This method removes the element from the set. If the element is not present then it will throw an error.

discard( ) - This also removes the element but , If the element is not present then the set will remain unchanged.

pop( ) - This method removes and returns an arbitrary element from a set.

clear( ) - This method is used to delete all the set elements.

[11]:
```python
mySet = {1, 2, 3, 5}
print("Before: ", mySet)

mySet.remove(3)
print("Using remove: ", mySet)

mySet.discard(2)
print("Using discard: ", mySet)

mySet.discard(4)
#mySet.remove(4)

#In comparison to lists, the pop method does not take any arguments
removedSet = mySet.pop()
print("Popped Element: ", removedSet)

mySet.clear()
print(mySet)
```

```
Before:  {1, 2, 3, 5}
Using remove:  {1, 2, 5}
Using discard:  {1, 5}
Popped Element:  1
set()
```

Deleting a Set

```
[12]: #You can delete a set as a whole

tempSet = {1, 2, 3, 4, 5}
print(tempSet)

del tempSet
print(tempSet) # throws NameError: name 'tempSet' is not defined
```

```
{1, 2, 3, 4, 5}
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-4087e25361e0> in <module>
      5
      6 del tempSet
----> 7 print(tempSet) # throws NameError: name 'tempSet' is not defined

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

Copying a Set

```
[13]: #shallow Copy
numbers = {1, 2, 3, 4}
print('numbers: ',numbers)
# copies the items of numbers to new_numbers
new_numbers = numbers.copy()
print('new_numbers: ',new_numbers)

# add 5 to the copied set
new_numbers.add(5)
print('numbers: ',numbers)
print('new_numbers: ', new_numbers)
```

```
numbers:  {1, 2, 3, 4}
new_numbers:  {1, 2, 3, 4}
numbers:  {1, 2, 3, 4}
new_numbers:  {1, 2, 3, 4, 5}
```

ACCESSING ELEMENTS OF A SET : Iterating Through a Set

```
[14]: #The elements of the set are not accessed with the help of index numbers
#rather they are accessed by looping through it.

dataScientist = {'Python', 'R', 'SQL', 'Git', 'Tableau', 'SAS'}
for skill in dataScientist:
   print(skill)
```

```
# If you look at the output of printing each of the values in dataScientist,
# notice that the values printed in the set are not in the order they were␣
↪added in.
# This is because sets are unordered.
```

```
Git
R
Python
Tableau
SAS
SQL
```

[15]:
```
#Transform Set into Ordered Values
#We can use the sorted function which outputs a list that is ordered.

print(sorted(dataScientist))
type(sorted(dataScientist))
```

```
['Git', 'Python', 'R', 'SAS', 'SQL', 'Tableau']
```

[15]:
```
list
```

Set Operations in Python

A common use of sets in Python is computing standard math operations such as union, intersection, difference, and symmetric difference. Most operations on set can be performed in two ways.

Using operator

Using methods

[16]:
```
#To know what all methods can be used on sets, you can use the dir() function.
My_Set={1,'s',7.8}
dir(My_Set)
```

[16]:
```
['__and__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__iand__',
 '__init__',
 '__init_subclass__',
 '__ior__',
 '__isub__',
 '__iter__',
 '__ixor__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__or__',
 '__rand__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__ror__',
 '__rsub__',
 '__rxor__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__xor__',
 'add',
 'clear',
 'copy',
 'difference',
 'difference_update',
 'discard',
 'intersection',
 'intersection_update',
 'isdisjoint',
 'issubset',
 'issuperset',
 'pop',
 'remove',
 'symmetric_difference',
 'symmetric_difference_update',
 'union',
 'update']
```

[18]:
```
help(My_Set.discard)
My_Set.discard()
```

```
Help on built-in function discard:

discard(…) method of builtins.set instance
```

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

If the element is not a member, do nothing.
```

Union of Sets

Union refers to the concatenation of two or more sets into a single set by adding all unique elements present in both sets. This can be done in two ways:

Using pipeline “|”

Using union( ) function

[19]:
```
#let’s start by initializing three sets dataScientist , dataEngineer and␣
↪graphicDesigner
dataScientist = set(['Python', 'R', 'SQL', 'Git', 'Tableau', 'SAS'])
dataEngineer = set(['Python', 'Java', 'Scala', 'Git', 'SQL', 'Hadoop'])
graphicDesigner = {'Illustrator', 'InDesign', 'Photoshop'}
```

[20]:
```
# set built-in function union
print(dataScientist.union(dataEngineer))

# Equivalent Result
print(dataScientist | dataEngineer)
```

```
{'Git', 'R', 'Python', 'Tableau', 'Scala', 'Hadoop', 'SAS', 'Java', 'SQL'}
{'Git', 'R', 'Python', 'Tableau', 'Scala', 'Hadoop', 'SAS', 'Java', 'SQL'}
```

[21]:
```
#Union of three sets
print("Union of three sets= ",dataScientist.union(dataEngineer,graphicDesigner))
```

```
Union of three sets=  {'Git', 'R', 'Illustrator', 'Python', 'Photoshop',
'Tableau', 'Scala', 'Hadoop', 'InDesign', 'SAS', 'Java', 'SQL'}
```

Intersection of Sets

The intersection of two or more sets is a new set consisting of only the common elements present in those sets. This can be done in two ways:

Using “&” symbol

Using intersection( ) function

[22]:
```
# Intersection operation
print(dataScientist.intersection(dataEngineer))
# Equivalent Result
print(dataScientist & dataEngineer)

#"A.intersection(B)" is the same as "B.intersection(A)"
```

```
{'Git', 'SQL', 'Python'}
{'Git', 'SQL', 'Python'}
```

Difference of Sets

The difference of sets produces a new set consisting of elements that are present only in one of those sets. This means that all elements except the common elements of those sets will be returned. This can be done in two ways:

Using the '-' symbol

Using difference( ) function

[24]:
```
# Difference Operation
print(dataScientist.difference(dataEngineer))
# Equivalent Result
print(dataEngineer - dataScientist)
#The output consists of all elements of set 'dataScientist' except those that␣
↪are present in 'dataEngineer'
```

```
{'Tableau', 'SAS', 'R'}
{'Java', 'Scala', 'Hadoop'}
```

[25]:
```
#difference of three sets
print("Difference of three sets= ",dataScientist.
↪difference(dataEngineer,graphicDesigner))
#The output consists of all elements of set 'dataScientist' except those that␣
↪are present in 'dataEngineer' and 'graphicDesigner'.
```

```
Difference of three sets=  {'Tableau', 'SAS', 'R'}
```

Symmetric Difference of Sets

The symmetric difference of two sets denoted   , is the set of all values that are values of exactly one of two sets, but not both.

This can be done in two ways:

Using the '^' symbol

Using symmetric_difference( ) function

[26]:
```
# Symmetric Difference Operation
print(dataScientist.symmetric_difference(dataEngineer))
# Equivalent Result
print(dataEngineer ^ dataScientist)

#The symmetric difference method finds the elements that exist in only set A or␣
↪only set B.
#Thus, it returns the union of "A difference B" and "B difference A".
```

```
{'R', 'Tableau', 'Scala', 'Hadoop', 'SAS', 'Java'}
{'R', 'Tableau', 'Scala', 'Hadoop', 'SAS', 'Java'}
```

Some Other Python Set Methods

Difference Update of Sets

The difference_update( ) method computes the difference between two sets (A - B) and updates set A with the resulting set.

[27]:
```
# sets of numbers
A = {1, 3, 5, 7, 9}
B = {2, 3, 5, 7, 11}

# computes A - B and updates A with the resulting set
print(A.difference(B))
A.difference_update(B)

print('A = ', A)

# Output: A =  {1, 9}
```

```
{1, 9}
A =  {1, 9}
```

Intersection Update of Sets

The intersection_update() finds the intersection of different sets and updates it to the set that calls the method.

[28]:
```
A = {1, 2, 3, 4}
B = {2, 3, 4, 5}
C = {4, 5, 6, 9, 10}

# updates set A with the items common to both sets A and B
print(A.intersection(B))
A.intersection_update(B)

print('A =', A) # Output: A = {2, 3, 4}

# performs intersection between A, B and C and updates the result to set A
print(A.intersection(B,C))
A.intersection_update(B, C)

print('A =', A)
```

```
{2, 3, 4}
A = {2, 3, 4}
{4}
A = {4}
```

Is Disjoint of Sets

The isdisjoint( ) method returns True if two sets don't have any common items between them, i.e. they are disjoint. Else the returns False.

[29]:
```
A = {1, 2, 3}
B = {4, 5, 6}
C = {1, 4, 5}

# checks if set A and set B are disjoint
print(A.isdisjoint(B)) # Output: True
print(A.isdisjoint(C)) # Output: False
```

```
True
False
```

Subsets and Supersets

A set A is a subset of a set B (A   B) or equivalently set B is a superset of set A (B   A), if all elements of set A are contained in set B.

To check whether set A is a subset/superset of set B, we can use the following methods:

A.issubset(B) → This method returns True if all elements of set A are contained in set B.

A.issuperset(B) → This method returns True if all elements of set B are contained in set A.

[32]:
```
whole = {0,1,2,3,4,5,6,7,8} #superset

#subsets
even = {2,4,6,8}
odd = {1,3,5,7}
natural = {1,2,3,4,5}
```

[33]:
```
# three sets
mammals = {'Tiger', 'Camel', 'Sheep', 'Whale', 'Walrus'} #superset
aquatic = {'Octopus', 'Squid', 'Crab', 'Whale', 'Walrus'}
aquatic_mammals = {'Whale', 'Walrus'} #subset

# mammals is a superset of aquatic_mammals
print(mammals.issuperset(aquatic_mammals))
# True

# aquatic_mammals is a subset of mammals
print(aquatic_mammals.issubset(mammals))
# True

# aquatic is not a subset of mammals
print(aquatic.issubset(mammals))
# False

# aquatic is not a subset of mammals
print(aquatic_mammals.issubset(aquatic))
```

```
True
True
```

```
False
True
```

Check if Item Exists in a Set

To check if a specific item is present in a set, you can use in and not in operators with if statement.

[34]:
```python
# Check for presence
S = {'red', 'green', 'blue'}
if 'red' in S:
    print('yes')

print('red' in S)

# Check for absence
S = {'red', 'green', 'blue'}
if 'yellow' not in S:
    print('yes')

print('red' not in S)
```

```
yes
True
yes
False
```

[36]:
```python
#Concatitation and repetation are not supported in sets
#print(mammals + aquatic)
print(mammals*3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-20fc550c3307> in <module>
      1 #Concatitation and repetation are not supported in sets
      2 #print(mammals + aquatic)
----> 3 print(mammals*3)

TypeError: unsupported operand type(s) for *: 'set' and 'int'
```

Python Frozenset

Frozenset is a new class that has the characteristics of a set, but its elements cannot be changed once assigned

While tuples are immutable lists, frozensets are immutable sets.

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.

You cannot normally have nested sets ,as sets cannot contain mutable values including sets

Frozensets can be created using the frozenset( ) function.

```
[38]: # Initialize a frozenset
      immutableSet = frozenset()
      print(immutableSet)
```

```
frozenset()
```

```
[39]: a={1, 2,5, 4.6, 7.8, 'r', 's'}
      b=frozenset(a)
      print(b)
```

```
frozenset({1, 2, 4.6, 5, 's', 7.8, 'r'})
```

```
[40]: #Notice that we don't get any errors and the set is created successfully.
      a = {frozenset([1, 2, 3]), frozenset([1, 2, 4])}
      a
```

```
[40]: {frozenset({1, 2, 3}), frozenset({1, 2, 4})}
```

```
[41]: #Unlike sets, frozensets are unchangeable so they can be used as keys to a␣
       ↪dictionary.
      myDict = {frozenset(['Programming', 'Lanuguage', 'Name']): "Python", 'type':␣
       ↪'interpreted'}
      print(myDict)
```

```
{frozenset({'Lanuguage', 'Name', 'Programming'}): 'Python', 'type':
'interpreted'}
```

```
[42]: #This data type supports methods like copy(), difference(), intersection(),
      #isdisjoint(), issubset(), issuperset(),symmetric_difference() and union().

      # initialize A and B
      A = frozenset([1, 2, 3, 4])
      B = frozenset([3, 4, 5, 6])


      print(A.isdisjoint(B))

      print(A.difference(B))

      print(A | B)
```

```
False
frozenset({1, 2})
frozenset({1, 2, 3, 4, 5, 6})
```

```
[44]: #Being immutable, it does not have methods that add or remove elements.
      #A.add(3)
      A.pop()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-44-280fed7cdae8> in <module>
      1 #Being immutable, it does not have methods that add or remove elements.
      2 #A.add(3)
----> 3 A.pop()

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

Set Comprehension

```
[45]: {skill for skill in ['SQL', 'SQL', 'PYTHON', 'PYTHON']}
```

```
[45]: {'PYTHON', 'SQL'}
```

```
[46]: {skill for skill in ['GIT', 'PYTHON', 'SQL'] if skill not in {'GIT', 'PYTHON',␣
      ↪'JAVA'}}
```

```
[46]: {'SQL'}
```

Built-in Functions with Set

Built-in functions like all( ), any( ), enumerate( ), len( ), max( ), min( ), sorted( ), sum( ) etc. are commonly used with sets to perform different tasks.

Python Interview Questions on Sets

Does converting an object to a set maintain the object's order?

Is a set a subset of itself ?

What is a set?

What is the difference between a Python Set and Python frozen set ?

Can a set be accessed by index ?

What is the difference between a subset and a proper subset ?

To-Do:

Write a program to Check if two lists have at-least one element common

Write a program to Check if two sets have any elements in common. If yes, display the common elements

Write a program in python as a function that determines the intersection of two sets without using any predefined functions.

Write a python program as a function that determines the union of two sets without using any predefined functions

Write a program in python as a function which examines whether two sets are disjoint or not without using any predefined functions

Write a python program as a function that determines the minimum and maximum values of given a set withowt using any predefined function