Python Sets

In Python, a **Set** is an unordered collection of data types that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements. The major advantage of using a set, as opposed to a list, is that it has a highly optimized method for checking whether a specific element is contained in the set.

Creating a Set

Sets can be created by using the built-in **set()** function with an iterable object or a sequence by placing the sequence inside curly braces, separated by a 'comma'.

> ***Note:*** *A set cannot have mutable elements like a list or dictionary, as it is immutable.*

Python3

```
# Python program to demonstrate
# Creation of Set in Python

# Creating a Set
set1 = set()
print("Initial blank Set: ")
print(set1)

# Creating a Set with
# the use of a String
set1 = set("GeeksForGeeks")
print("\nSet with the use of String: ")
print(set1)

# Creating a Set with
# the use of Constructor
# (Using object to Store String)
```

```
String = 'GeeksForGeeks'
set1 = set(String)
print("\nSet with the use of an Object: " )
print(set1)

# Creating a Set with
# the use of a List
set1 = set(["Geeks", "For", "Geeks"])
print("\nSet with the use of List: ")
print(set1)

# Creating a Set with
# the use of a tuple
t=("Geeks","for","Geeks")
print("\nSet with the use of Tuple: ")
print(set(t))

# Creating a Set with
# the use of a dictionary
d={"Geeks":1,"for":2,"Geeks":3}
print("\nSet with the use of Dictionary: ")
print(set(d))
```

Ouput

```
Initial blank Set:
set()

Set with the use of String:
{'e', 'G', 's', 'F', 'o', 'r', 'k'}

Set with the use of an Object:
{'e', 'G', 's', 'F', 'o', 'r', 'k'}

Set with the use of List:
{'For', 'Geeks'}

Set with the use of Tuple:
{'for', 'Geeks'}

Set with the use of Dictionary:
{'for', 'Geeks'}
```

Time complexity: O(n), where n is the length of the input string, list, tuple or dictionary .
Auxiliary space: O(n), where n is the length of the input string, list, tuple or dictionary, since the size of the set created depends on the size of the input.

A set contains only unique elements but at the time of set creation, multiple duplicate values can also be passed. Order of elements in a set is undefined and is unchangeable. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

Python3

```python
# Creating a Set with
# a List of Numbers
# (Having duplicate values)
set1 = set([1, 2, 4, 4, 3, 3, 3, 6, 5])
print("\nSet with the use of Numbers: ")
print(set1)

# Creating a Set with
# a mixed type of values
# (Having numbers and strings)
set1 = set([1, 2, 'Geeks', 4, 'For', 6, 'Geeks'])
print("\nSet with the use of Mixed Values")
print(set1)
```

Output

```
Set with the use of Numbers:
{1, 2, 3, 4, 5, 6}

Set with the use of Mixed Values
{1, 2, 4, 6, 'Geeks', 'For'}
```

Creating a set with another method

Python3

```
# Another Method to create sets in Python3

# Set containing numbers
my_set = {1, 2, 3}

print(my_set)

# This code is contributed by sarajadhav12052009
```

Output

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

Adding Elements to a Set

Using add() method

Elements can be added to the Set by using the built-in **add()** function. Only one element at a time can be added to the set by using add() method, loops are used to add multiple elements at a time with the use of add() method.

> ***Note:*** *Lists cannot be added to a set as elements because Lists are not hashable whereas Tuples can be added because tuples are immutable and hence Hashable.*

Python3

```
# Python program to demonstrate
# Addition of elements in a Set

# Creating a Set
set1 = set()
print("Initial blank Set: ")
print(set1)

# Adding element and tuple to the Set
set1.add(8)
set1.add(9)
set1.add((6, 7))
print("\nSet after Addition of Three elements: ")
print(set1)

# Adding elements to the Set
# using Iterator
for i in range(1, 6):
    set1.add(i)
print("\nSet after Addition of elements from 1-5: ")
print(set1)
```

Output

```
Initial blank Set:
set()

Set after Addition of Three elements:
{8, 9, (6, 7)}

Set after Addition of elements from 1-5:
{1, 2, 3, (6, 7), 4, 5, 8, 9}
```

Using update() method

For the addition of two or more elements Update() method is used. The update() method accepts lists, strings, tuples as well as other sets as its arguments. In all of these cases, duplicate elements are avoided.

Python3

```
# Python program to demonstrate
# Addition of elements in a Set
```

```
# Addition of elements to the Set
# using Update function
set1 = set([4, 5, (6, 7)])
set1.update([10, 11])
print("\nSet after Addition of elements using Update: ")
print(set1)
```

Output

```
Set after Addition of elements using Update:
{4, 5, (6, 7), 10, 11}
```

Accessing a Set

Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you 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.

Python3

```
# Python program to demonstrate
# Accessing of elements in a set

# Creating a set
set1 = set(["Geeks", "For", "Geeks."])
print("\nInitial set")
print(set1)

# Accessing element using
# for loop
print("\nElements of set: ")
for i in set1:
    print(i, end=" ")

# Checking the element
# using in keyword
print("\n")
print("Geeks" in set1)
```

Output

```
Initial set
{'Geeks.', 'For', 'Geeks'}

Elements of set:
Geeks. For Geeks

True
```

Removing elements from the Set

Using remove() method or discard() method:

Elements can be removed from the Set by using the built-in remove() function but a KeyError arises if the element doesn't exist in the set. To remove elements from a set without KeyError, use discard(), if the element doesn't exist in the set, it remains unchanged.

Python3

```python
# Python program to demonstrate
# Deletion of elements in a Set

# Creating a Set
set1 = set([1, 2, 3, 4, 5, 6,
            7, 8, 9, 10, 11, 12])
print("Initial Set: ")
print(set1)

# Removing elements from Set
# using Remove() method
set1.remove(5)
set1.remove(6)
print("\nSet after Removal of two elements: ")
print(set1)

# Removing elements from Set
# using Discard() method
set1.discard(8)
set1.discard(9)
print("\nSet after Discarding two elements: ")
print(set1)

# Removing elements from Set
# using iterator method
for i in range(1, 5):
```

```
    set1.remove(i)
print("\nSet after Removing a range of elements: ")
print(set1)
```

Output

```
Initial Set:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

Set after Removal of two elements:
{1, 2, 3, 4, 7, 8, 9, 10, 11, 12}

Set after Discarding two elements:
{1, 2, 3, 4, 7, 10, 11, 12}

Set after Removing a range of elements:
{7, 10, 11, 12}
```

Using pop() method:

Pop() function can also be used to remove and return an element from the set, but it removes only the last element of the set.

> ***Note:*** *If the set is unordered then there's no such way to determine which element is popped by using the pop() function.*

Python3

```
# Python program to demonstrate
# Deletion of elements in a Set

# Creating a Set
set1 = set([1, 2, 3, 4, 5, 6,
            7, 8, 9, 10, 11, 12])
print("Initial Set: ")
print(set1)

# Removing element from the
# Set using the pop() method
set1.pop()
```

```
print("\nSet after popping an element: ")
print(set1)
```

Output

```
Initial Set:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

Set after popping an element:
{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

Using clear() method:

To remove all the elements from the set, clear() function is used.

Python3

```
#Creating a set
set1 = set([1,2,3,4,5])
print("\n Initial set: ")
print(set1)


# Removing all the elements from
# Set using clear() method
set1.clear()
print("\nSet after clearing all the elements: ")
print(set1)
```

Output

```
 Initial set:
{1, 2, 3, 4, 5}

Set after clearing all the elements:
set()
```

Frozen sets in Python are immutable objects that only support methods and operators that produce a result without affecting the frozen set or sets to

which they are applied. While elements of a set can be modified at any time, elements of the frozen set remain the same after creation.

If no parameters are passed, it returns an empty frozenset.

Python3

```
# Python program to demonstrate
# working of a FrozenSet

# Creating a Set
String = ('G', 'e', 'e', 'k', 's', 'F', 'o', 'r')

Fset1 = frozenset(String)
print("The FrozenSet is: ")
print(Fset1)

# To print Empty Frozen Set
# No parameter is passed
print("\nEmpty FrozenSet: ")
print(frozenset())
```

Output

```
The FrozenSet is:
frozenset({'F', 's', 'o', 'G', 'r', 'e', 'k'})

Empty FrozenSet:
frozenset()
```

Typecasting Objects into sets

Python3

```
# Typecasting Objects in Python3 into sets

# Typecasting list into set
my_list = [1, 2, 3, 3, 4, 5, 5, 6, 2]
my_set = set(my_list)
print("my_list as a set: ", my_set)

# Typecasting string into set
my_str = "GeeksforGeeks"
my_set1 = set(my_str)
```

```
print("my_str as a set: ", my_set1)

# Typecasting dictionary into set
my_dict = {1: "One", 2: "Two", 3: "Three"}
my_set2 = set(my_dict)
print("my_dict as a set: ", my_set2)

# This code is contributed by sarajadhav12052009
```

Output

```
my_list as a set:  {1, 2, 3, 4, 5, 6}
my_str as a set:  {'G', 'f', 'r', 'e', 'k', 'o', 's'}
my_dict as a set:  {1, 2, 3}
```

Example: Implementing all functions:

Python3

```
def create_set():
    my_set = {1, 2, 3, 4, 5}
    print(my_set)

def add_element():
    my_set = {1, 2, 3, 4, 5}
    my_set.add(6)
    print(my_set)

def remove_element():
    my_set = {1, 2, 3, 4, 5}
    my_set.remove(3)
    print(my_set)

def clear_set():
    my_set = {1, 2, 3, 4, 5}
    my_set.clear()
    print(my_set)

def set_union():
    set1 = {1, 2, 3}
    set2 = {4, 5, 6}
    my_set = set1.union(set2)
    print(my_set)

def set_intersection():
    set1 = {1, 2, 3, 4, 5}
    set2 = {4, 5, 6, 7, 8}
```

```
    my_set = set1.intersection(set2)
    print(my_set)

def set_difference():
    set1 = {1, 2, 3, 4, 5}
    set2 = {4, 5, 6, 7, 8}
    my_set = set1.difference(set2)
    print(my_set)

def set_symmetric_difference():
    set1 = {1, 2, 3, 4, 5}
    set2 = {4, 5, 6, 7, 8}
    my_set = set1.symmetric_difference(set2)
    print(my_set)

def set_subset():
    set1 = {1, 2, 3, 4, 5}
    set2 = {2, 3, 4}
    subset = set2.issubset(set1)
    print(subset)

def set_superset():
    set1 = {1, 2, 3, 4, 5}
    set2 = {2, 3, 4}
    superset = set1.issuperset(set2)
    print(superset)

if __name__ == '__main__':
    create_set()
    add_element()
    remove_element()
    clear_set()
    set_union()
    set_intersection()
    set_difference()
    set_symmetric_difference()
    set_subset()
    set_superset()
```

Output

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

```
{1, 2, 3, 6, 7, 8}
True
True
```

Advantages:

- **Unique Elements**: Sets can only contain unique elements, so they can be useful for removing duplicates from a collection of data.
- **Fast Membership Testing**: Sets are optimized for fast membership testing, so they can be useful for determining whether a value is in a collection or not.
- **Mathematical Set Operations:** Sets support mathematical set operations like union, intersection, and difference, which can be useful for working with sets of data.
- **Mutable**: Sets are mutable, which means that you can add or remove elements from a set after it has been created.

Disadvantages:

- **Unordered**: Sets are unordered, which means that you cannot rely on the order of the data in the set. This can make it difficult to access or process data in a specific order.
- **Limited Functionality:** Sets have limited functionality compared to lists, as they do not support methods like append() or pop(). This can make it more difficult to modify or manipulate data stored in a set.
- **Memory Usage:** Sets can consume more memory than lists, especially for small datasets. This is because each element in a set requires additional memory to store a hash value.
- **Less Commonly Used:** Sets are less commonly used than lists and dictionaries in Python, which means that there may be fewer resources or libraries available for working with them. This can make it more difficult to find solutions to problems or to get help with debugging.

Overall, sets can be a useful data structure in Python, especially for removing duplicates or for fast membership testing. However, their lack of ordering and limited functionality can also make them less versatile than lists or dictionaries, so it is important to carefully consider the advantages

and disadvantages of using sets when deciding which data structure to use in your Python program.

Set Methods

Function	Description
add()	Adds an element to a set
remove()	Removes an element from a set. If the element is not present in the set, raise a KeyError
clear()	Removes all elements form a set
copy()	Returns a shallow copy of a set
pop()	Removes and returns an arbitrary set element. Raise KeyError if the set is empty
update()	Updates a set with the union of itself and others
union()	Returns the union of sets in a new set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set

Function	Description
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns True if two sets have a null intersection
issubset()	Returns True if another set contains this set
issuperset()	Returns True if this set contains another set
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another

Recent Articles on Python Sets

Set Programs

- Program to accept the strings which contains all vowels
- Python program to find common elements in three lists using sets
- Find missing and additional values in two lists
- Pairs of complete strings in two sets
- Check whether a given string is Heterogram or not
- Maximum and Minimum in a Set
- Remove items from Set
- Python Set difference to find lost element from a duplicated array
- Minimum number of subsets with distinct elements using Counter
- Check if two lists have at-least one element common
- Program to count number of vowels using sets in given string
- Difference between two lists
- Python set to check if string is panagram
- Python set operations (union, intersection, difference and symmetric difference)
- Concatenated string with uncommon characters in Python