Python Sets

A set is a collection of unique data, meaning that elements within a set cannot be duplicated.

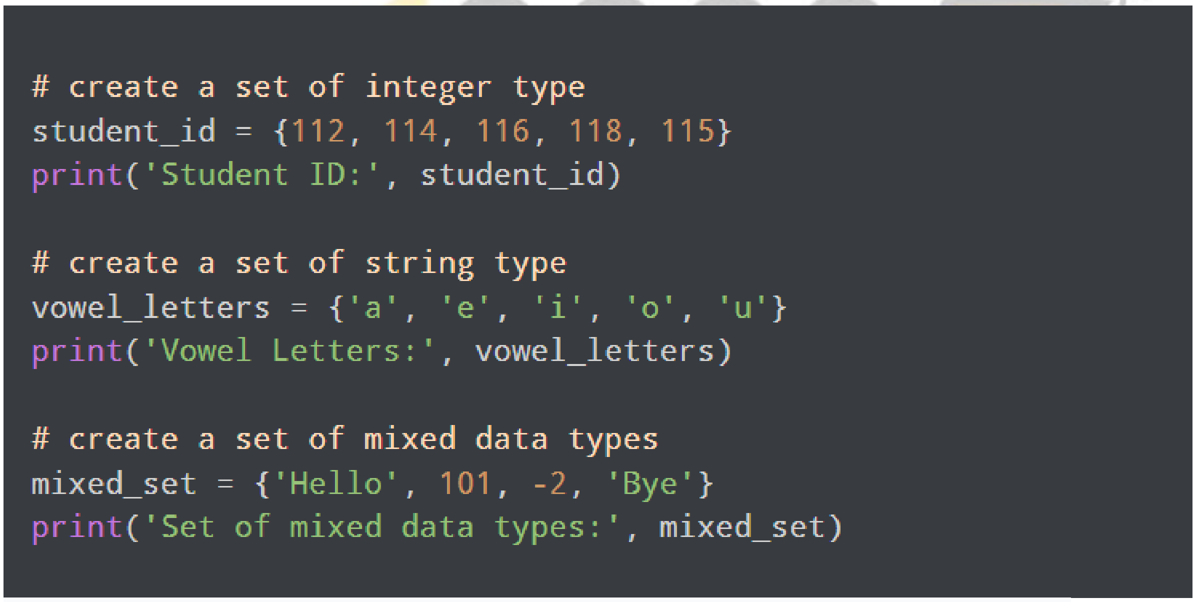
For instance, if we need to store information about student IDs, a set is suitable since student IDs cannot have duplicates.

Create a Set in Python

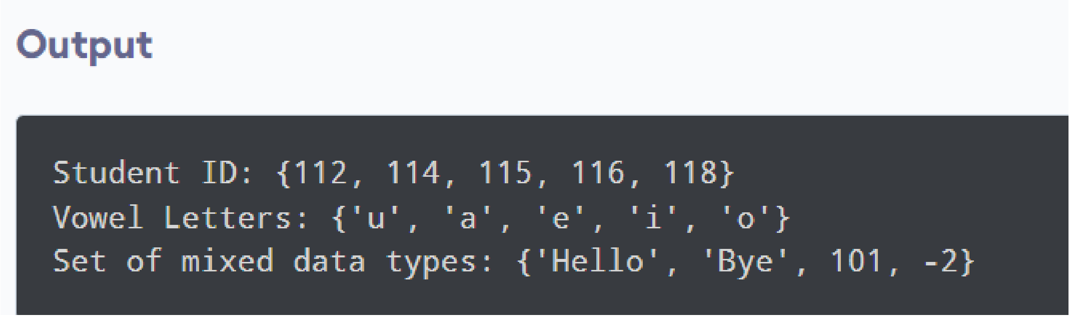
In Python, we create sets by placing all the elements inside curly braces {}, separated by commas.

A set can have any number of items and they may be of different types (integer, float, tuple, string, etc.). But a set cannot have mutable elements like lists, sets or dictionaries as its elements.

Let's see an example,



```python
# create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)

# create a set of string type
vowel_letters = {'a', 'e', 'i', 'o', 'u'}
print('Vowel Letters:', vowel_letters)

# create a set of mixed data types
mixed_set = {'Hello', 101, -2, 'Bye'}
print('Set of mixed data types:', mixed_set)
```



Output

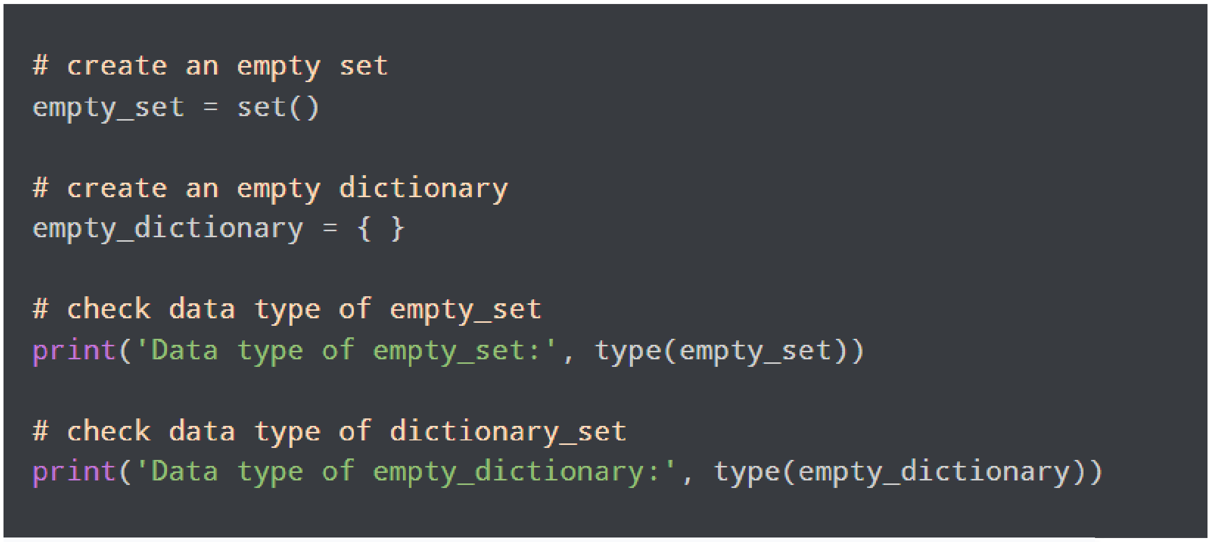
```
Student ID: {112, 114, 115, 116, 118}
Vowel Letters: {'u', 'a', 'e', 'i', 'o'}
Set of mixed data types: {'Hello', 'Bye', 101, -2}
```

In the above example, we have created different types of sets by placing all the elements inside the curly braces {}.

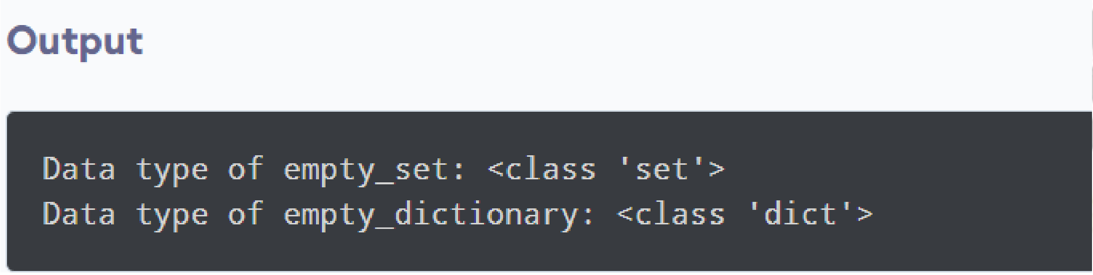
Create an Empty Set in Python

Creating an empty set is a bit tricky. 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. For example,



```
# create an empty set
empty_set = set()

# create an empty dictionary
empty_dictionary = { }

# check data type of empty_set
print('Data type of empty_set:', type(empty_set))

# check data type of dictionary_set
print('Data type of empty_dictionary:', type(empty_dictionary))
```



Output

```
Data type of empty_set: <class 'set'>
Data type of empty_dictionary: <class 'dict'>
```

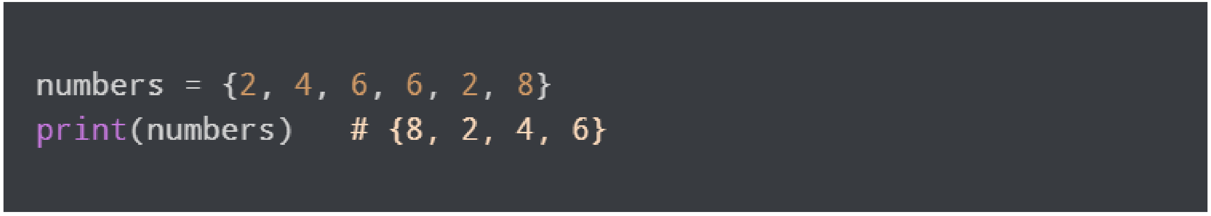
Here,

- empty_set - an empty set created using set()
- empty_dictionary - an empty dictionary created using {}

Finally, we have used the type() function to know which class empty_set and empty_dictionary belong to.

Duplicate Items in a Set

Let's see what will happen if we try to include duplicate items in a set.



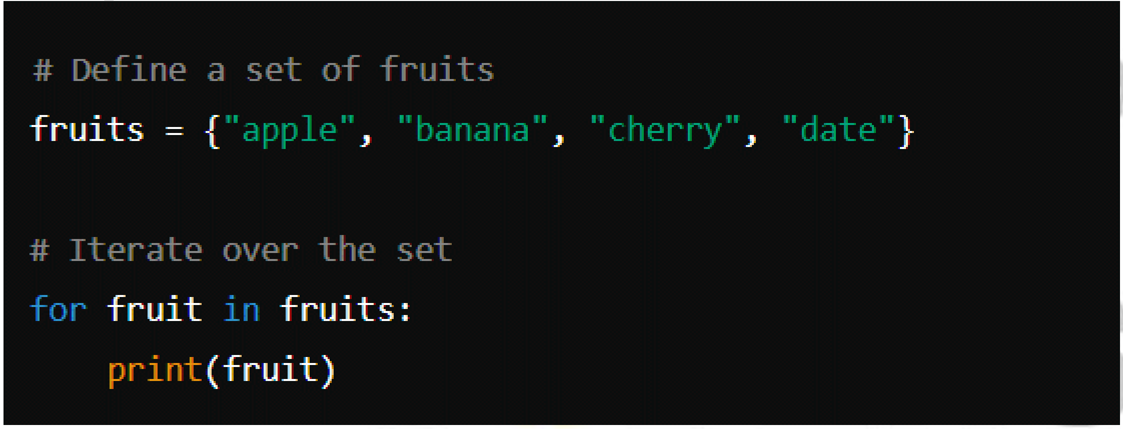
```
numbers = {2, 4, 6, 6, 2, 8}
print(numbers)   # {8, 2, 4, 6}
```

Here, we can see there are no duplicate items in the set as a set cannot contain duplicates.

Iterating Over Set Elements in Python

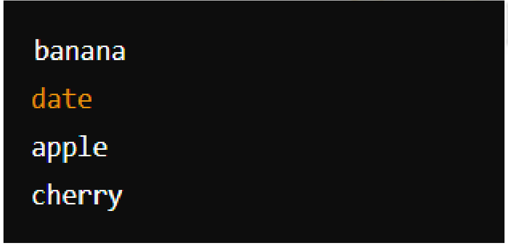
Since sets are unordered collections of unique elements, you cannot access elements by index, but you can iterate over the elements of a set using a loop, such as a for loop. Each iteration will give you one element from the set.

Example



```
# Define a set of fruits
fruits = {"apple", "banana", "cherry", "date"}

# Iterate over the set
for fruit in fruits:
    print(fruit)
```

Output



```
banana
date
apple
cherry
```

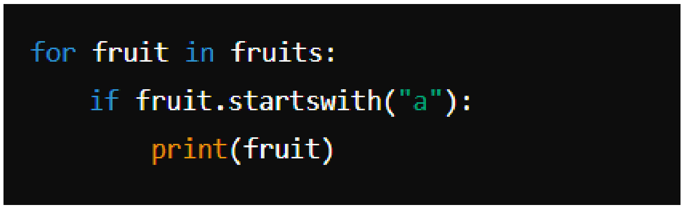
Explanation:

Set Definition: fruits is a set containing four elements: "apple", "banana", "cherry", and "date".

For Loop: The for loop iterates over each element in the set. In each iteration, the variable fruit takes the value of one of the elements in the set.

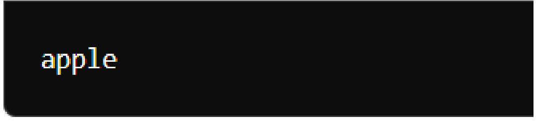
Iterating with Additional Logic

You can also perform additional operations during iteration. For example, if you want to print only fruits that start with the letter "a":



```python
for fruit in fruits:
    if fruit.startswith("a"):
        print(fruit)
```

Output



```
apple
```

Properties of Python Sets:

- Unordered Collection:

Sets do not maintain any particular order of elements. The order of items can change and is not guaranteed to be the same each time you access it.

- Unique Elements:

Sets only store unique elements. Duplicate elements are automatically removed.

- Mutable:

You can add or remove elements from a set after it has been created.

- Heterogeneous:

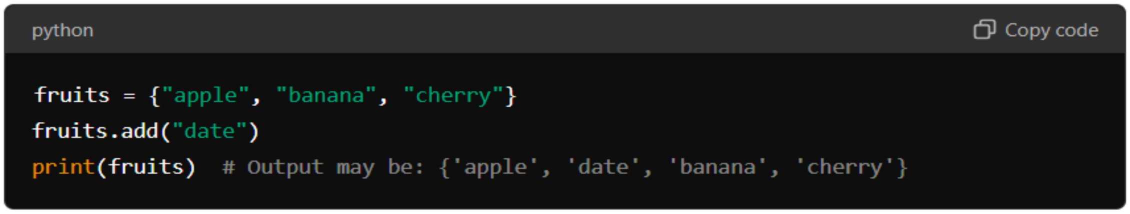
A set can contain elements of different data types (e.g., integers, strings, tuples, etc.).

- Unindexed:

Elements in a set cannot be accessed by index. Instead, you can iterate over the set.

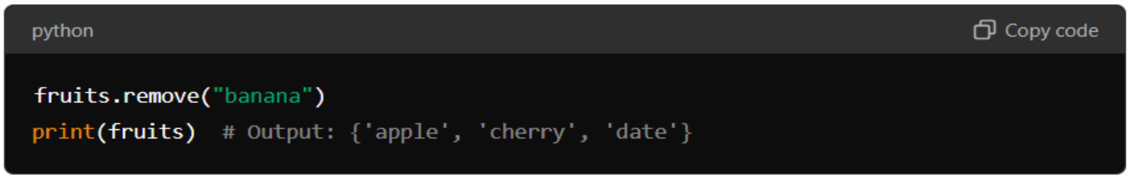
Set Methods in Python

1. `add()`: Adds an element to the set.



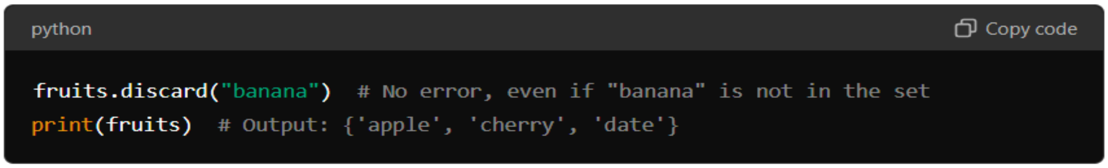
```python
fruits = {"apple", "banana", "cherry"}
fruits.add("date")
print(fruits)  # Output may be: {'apple', 'date', 'banana', 'cherry'}
```

2. `remove()`: Removes an element from the set. Raises a `KeyError` if the element is not found.



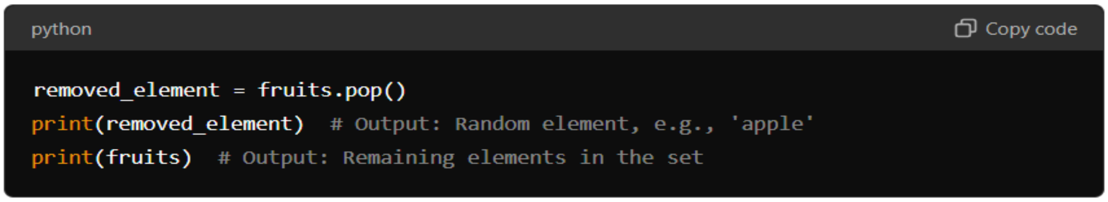
```python
fruits.remove("banana")
print(fruits)  # Output: {'apple', 'cherry', 'date'}
```

3. `discard()`: Removes an element from the set without raising an error if the element is not found.



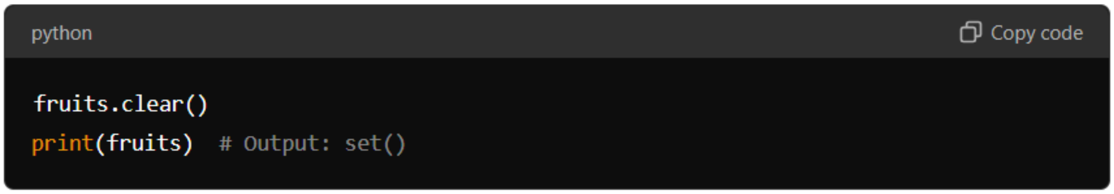
```python
fruits.discard("banana")  # No error, even if "banana" is not in the set
print(fruits)  # Output: {'apple', 'cherry', 'date'}
```

4. `pop()`: Removes and returns an arbitrary element from the set. Raises a `KeyError` if the set is empty.



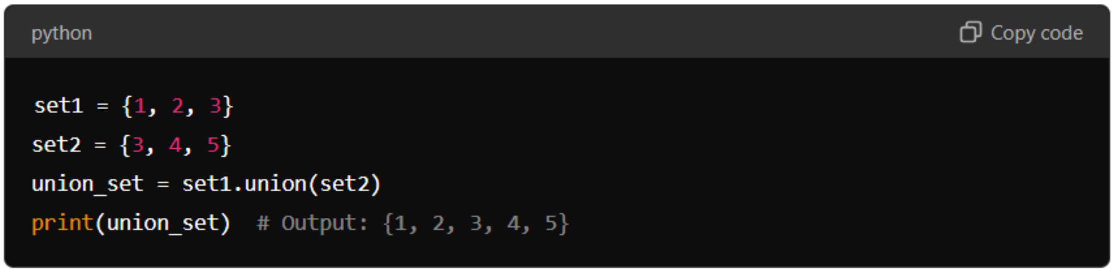
```python
removed_element = fruits.pop()
print(removed_element)  # Output: Random element, e.g., 'apple'
print(fruits)  # Output: Remaining elements in the set
```

5. `clear()`: Removes all elements from the set.

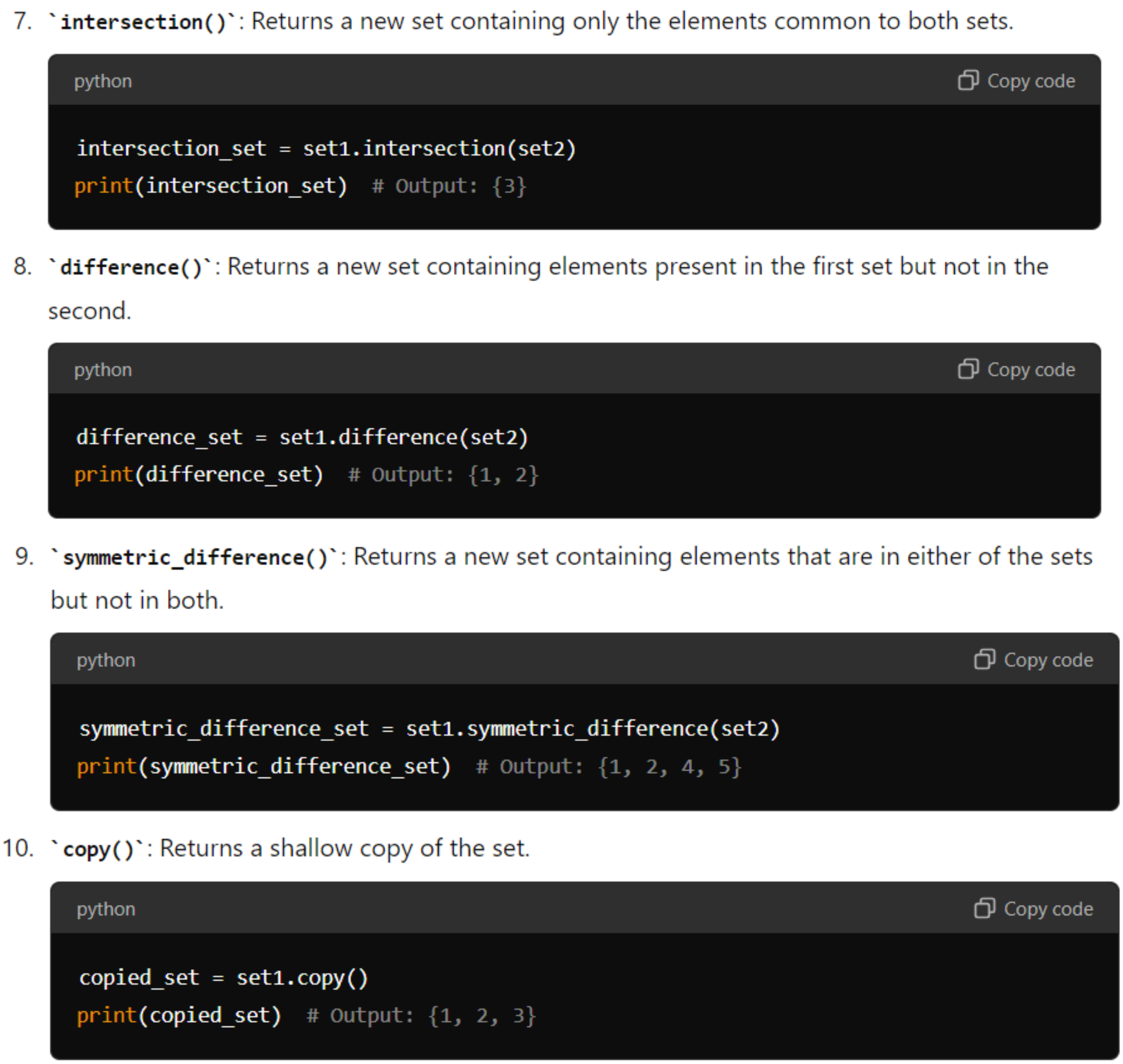


```python
fruits.clear()
print(fruits)  # Output: set()
```

6. `union()`: Returns a new set containing all elements from both sets.



```python
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1.union(set2)
print(union_set)  # Output: {1, 2, 3, 4, 5}
```



7. `intersection()`: Returns a new set containing only the elements common to both sets.

```python
intersection_set = set1.intersection(set2)
print(intersection_set)  # Output: {3}
```

8. `difference()`: Returns a new set containing elements present in the first set but not in the second.

```python
difference_set = set1.difference(set2)
print(difference_set)  # Output: {1, 2}
```

9. `symmetric_difference()`: Returns a new set containing elements that are in either of the sets but not in both.

```python
symmetric_difference_set = set1.symmetric_difference(set2)
print(symmetric_difference_set)  # Output: {1, 2, 4, 5}
```

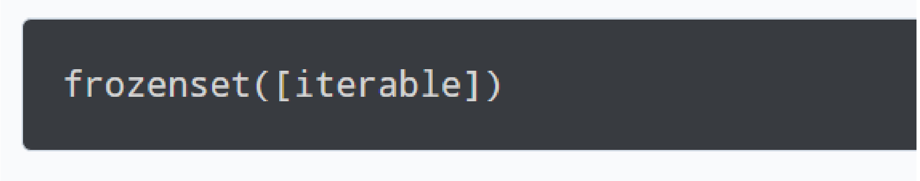
10. `copy()`: Returns a shallow copy of the set.

```python
copied_set = set1.copy()
print(copied_set)  # Output: {1, 2, 3}
```

Python frozenset()

Frozen set is just an immutable version of a Python set object. While elements of a set can be modified at any time, elements of the frozen set remain the same after creation. Due to this, frozen sets can be used as keys in Dictionary or as elements of another set. But like sets, it is not ordered (the elements can be set at any index).

The syntax of frozenset() function is:



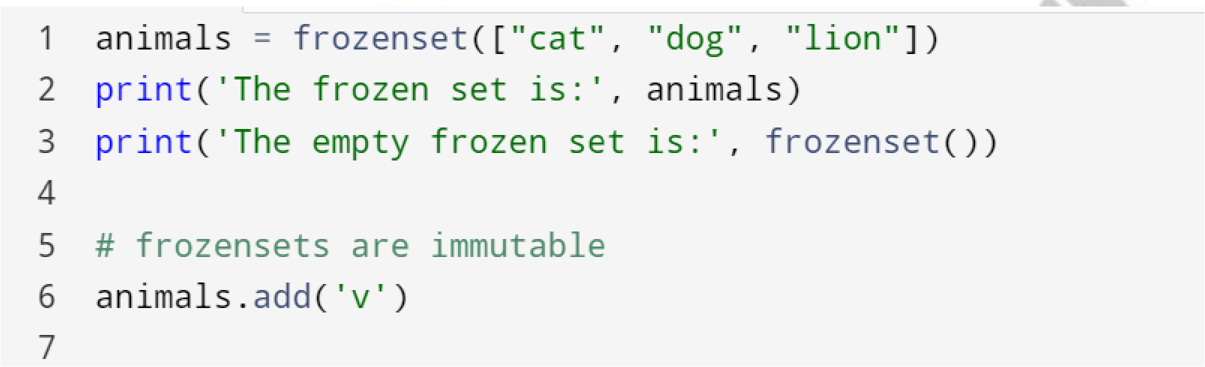
```
frozenset([iterable])
```

frozenset() Parameters

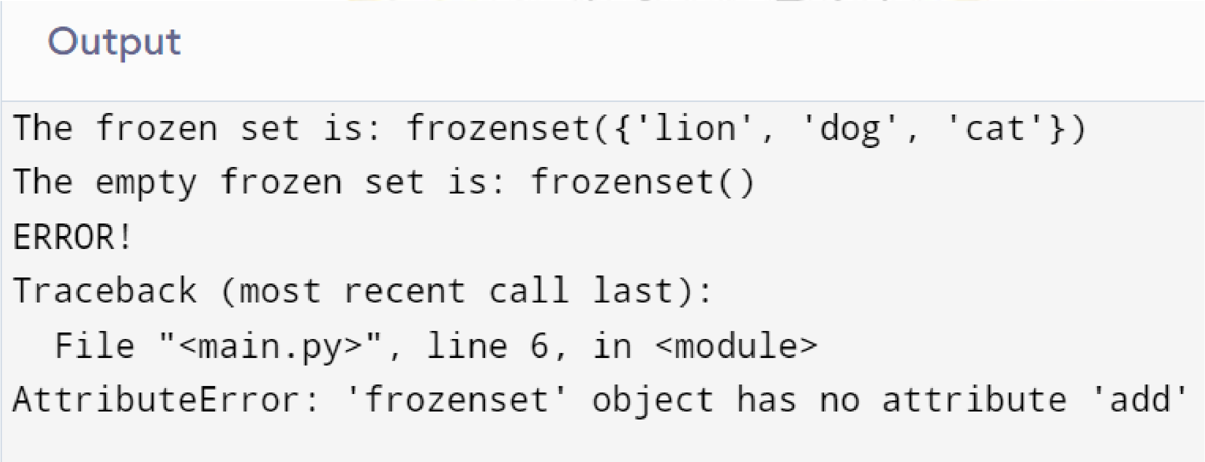
The frozenset() function takes a single parameter:

- **iterable (Optional)** - the iterable which contains elements to initialize the frozenset with.
  Iterable can be set, dictionary, tuple, etc.

Example : Working of Python frozenset()



```python
animals = frozenset(["cat", "dog", "lion"])
print('The frozen set is:', animals)
print('The empty frozen set is:', frozenset())

# frozensets are immutable
animals.add('v')

```

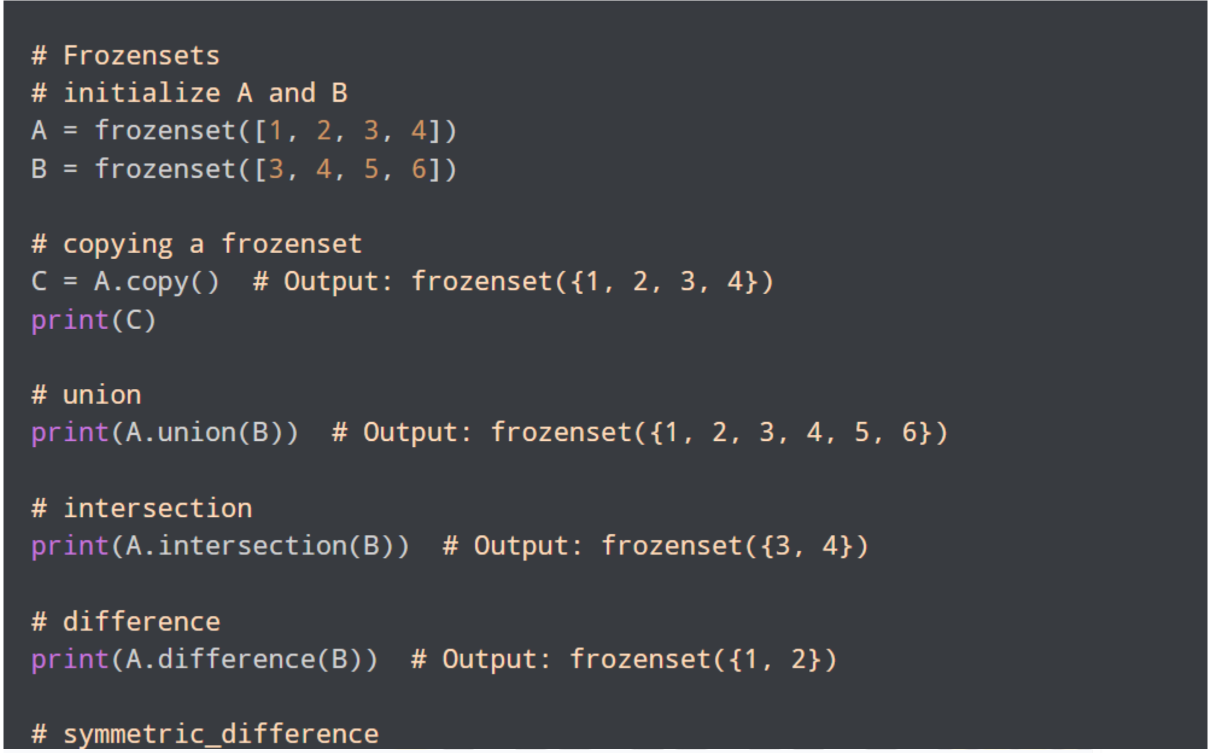


Output

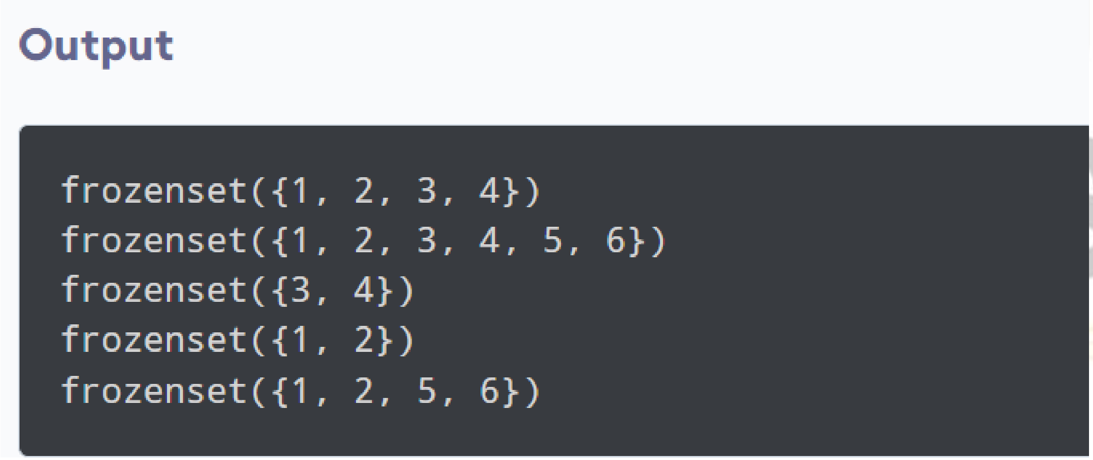
```
The frozen set is: frozenset({'lion', 'dog', 'cat'})
The empty frozen set is: frozenset()
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 6, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```

Frozenset operations

Like normal sets, frozenset can also perform different operations like copy(), difference(), intersection(), symmetric_difference(), and union().



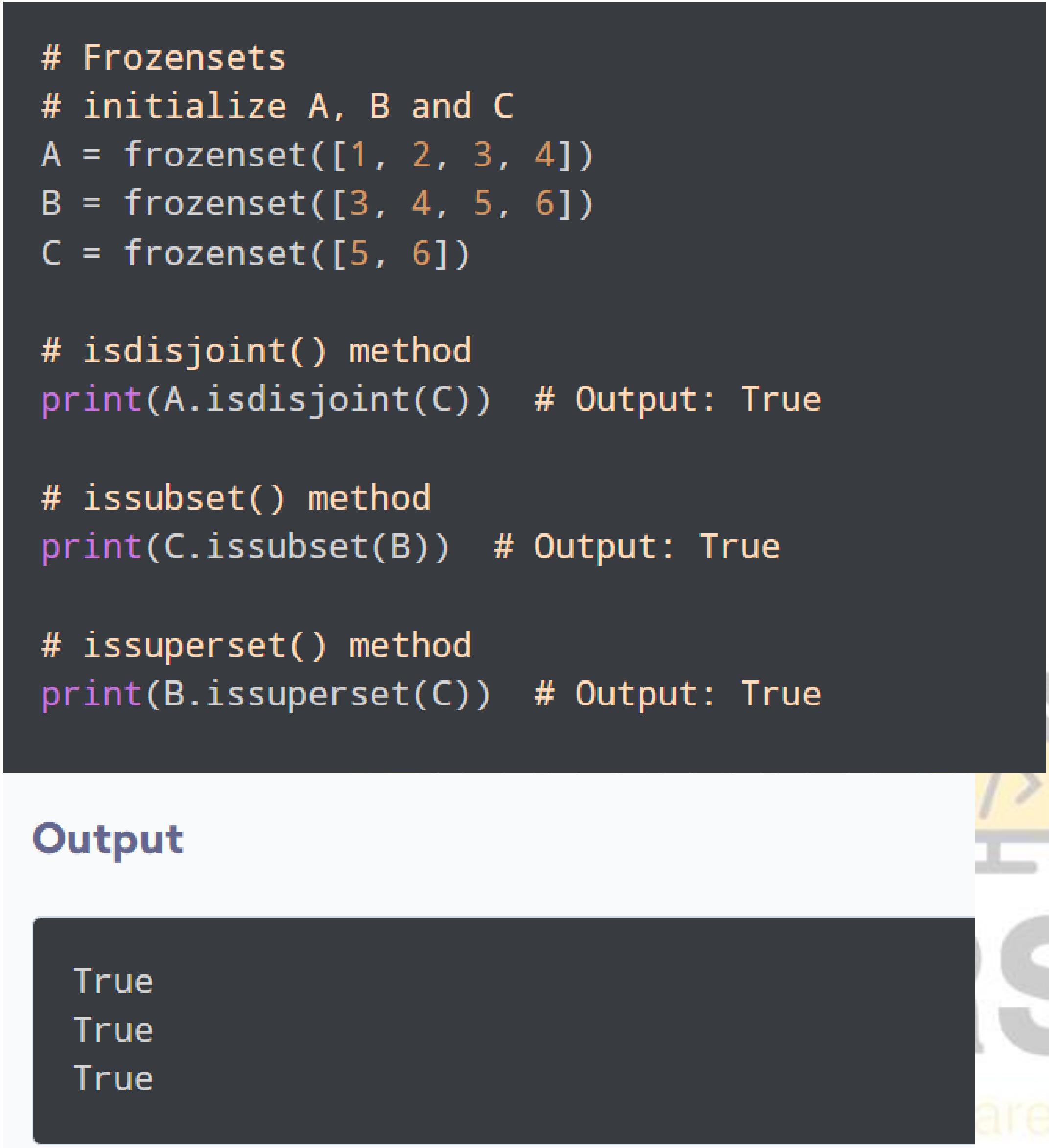
```
# Frozensets
# initialize A and B
A = frozenset([1, 2, 3, 4])
B = frozenset([3, 4, 5, 6])

# copying a frozenset
C = A.copy()  # Output: frozenset({1, 2, 3, 4})
print(C)

# union
print(A.union(B))  # Output: frozenset({1, 2, 3, 4, 5, 6})

# intersection
print(A.intersection(B))  # Output: frozenset({3, 4})

# difference
print(A.difference(B))  # Output: frozenset({1, 2})

# symmetric_difference
```



Output

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

Similarly, other set methods like isdisjoint(), issubset(), and issuperset() are also available.



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

# isdisjoint() method
print(A.isdisjoint(C))  # Output: True

# issubset() method
print(C.issubset(B))  # Output: True

# issuperset() method
print(B.issuperset(C))  # Output: True
```

Output

```
True
True
True
```