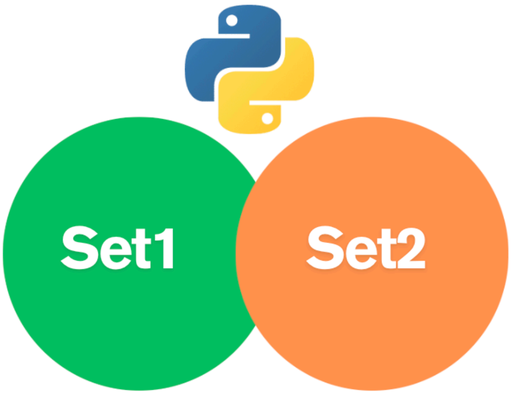
"Mastering Python Sets: A Beginner's Guide"

Learn Sets with Interactive Examples and Visuals



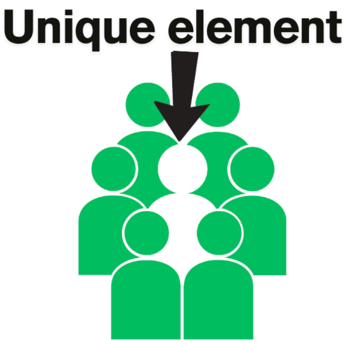
Created by Bhawana Saxena,

Date: May 30, 2025

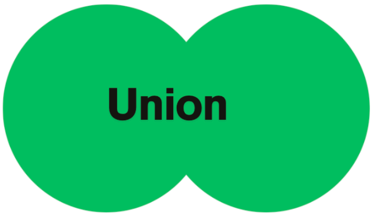
Introduction to Python Sets:

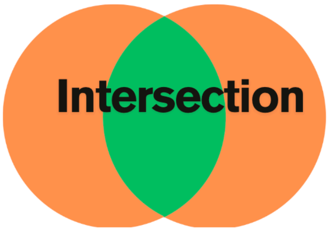
What is a Set?

- A set is an unordered, mutable collection of unique elements in Python.



- Used for storing distinct items, performing mathematical operations like union, intersection, etc.





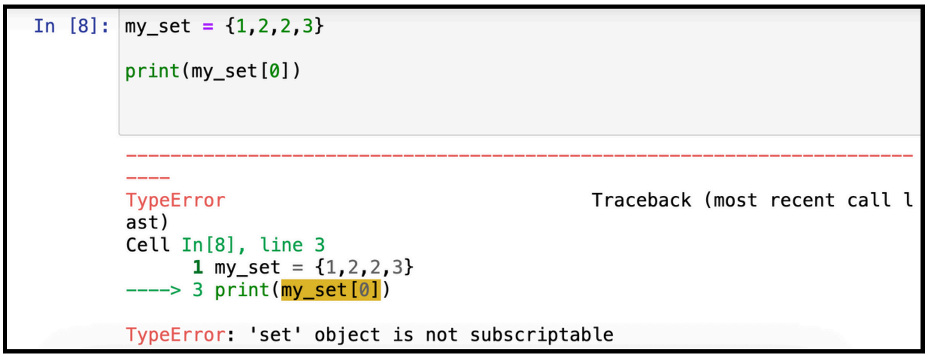
- ***Syntax:***

  my_set = {1, 2, 3}

  or

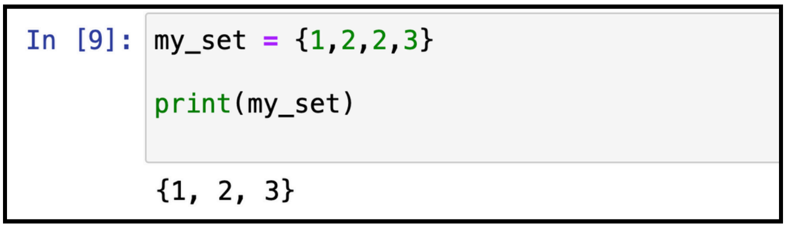
  my_set = set([1, 2, 3])

- ***Key Characteristics:***
  - Unordered: No indexing, e.g., my_set[0] raises an error.



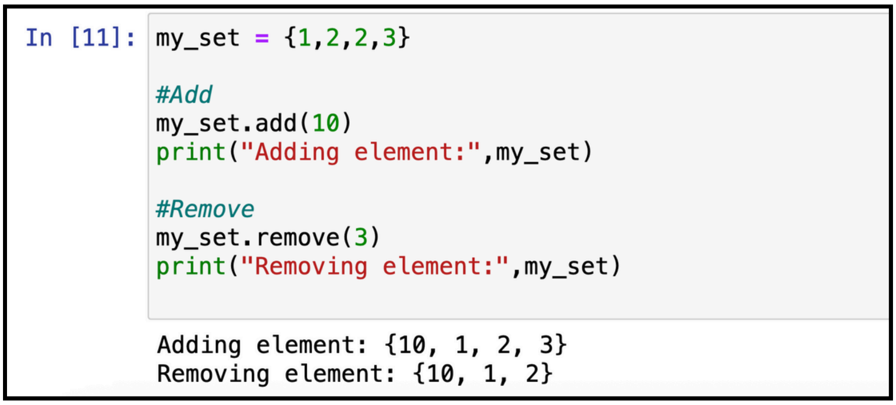
```
In [8]: my_set = {1,2,2,3}

        print(my_set[0])

        ---------------------------------------------------------------------------
        ----
        TypeError                                 Traceback (most recent call l
        ast)
        Cell In[8], line 3
              1 my_set = {1,2,2,3}
        ----> 3 print(my_set[0])

        TypeError: 'set' object is not subscriptable
```

  - Unique: Duplicates are automatically removed, e.g., {1, 2, 2, 3} becomes {1, 2, 3}.



```
In [9]: my_set = {1,2,2,3}

        print(my_set)

        {1, 2, 3}
```

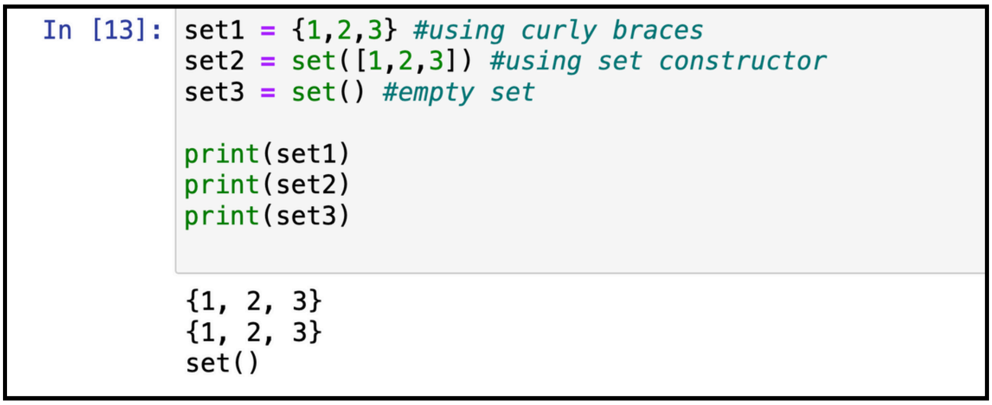
  - Mutable: Can add/remove elements, but elements themselves must be immutable **(e.g., numbers, strings, tuples).**



```
In [11]: my_set = {1,2,2,3}

         #Add
         my_set.add(10)
         print("Adding element:",my_set)

         #Remove
         my_set.remove(3)
         print("Removing element:",my_set)

         Adding element: {10, 1, 2, 3}
         Removing element: {10, 1, 2}
```

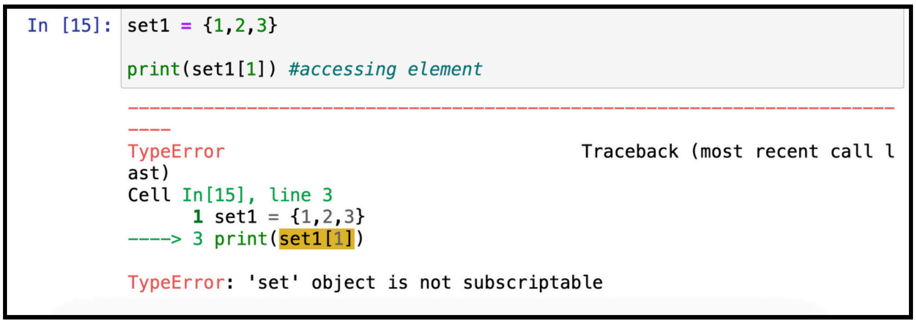
Page 2: Creating and Accessing Sets

- ***How to Create Sets:***
  - Using curly braces: {1, 2, 3}
  - Using set() constructor: set([1, 2, 3])
  - Empty set: set() (Note: {} creates an empty dictionary, not a set).



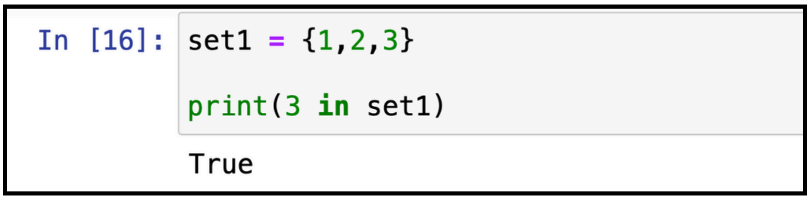
```
In [13]: set1 = {1,2,3} #using curly braces
         set2 = set([1,2,3]) #using set constructor
         set3 = set() #empty set

         print(set1)
         print(set2)
         print(set3)

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

- ***Accessing Elements:***
  - Cannot use indexing due to unordered nature.

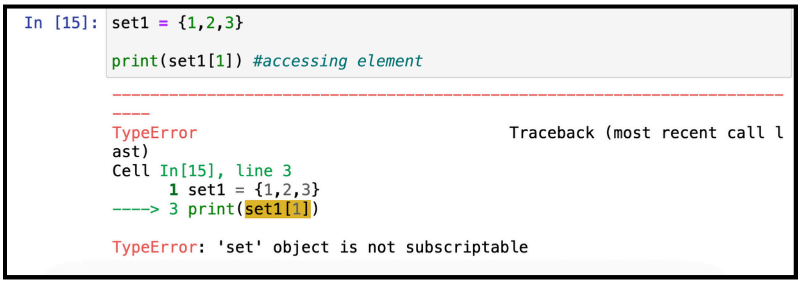


```
In [15]: set1 = {1,2,3}

         print(set1[1]) #accessing element

         ---------------------------------------------------------------------------
         ----
         TypeError                                 Traceback (most recent call l
         ast)
         Cell In[15], line 3
               1 set1 = {1,2,3}
         ----> 3 print(set1[1])

         TypeError: 'set' object is not subscriptable
```

  - Use loops or in operator to check membership.



```
In [16]: set1 = {1,2,3}

         print(3 in set1)

         True
```

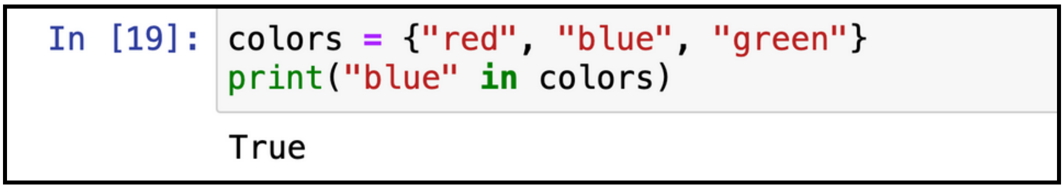


```
In [15]: set1 = {1,2,3}

         print(set1[1]) #accessing element

         ---------------------------------------------------------------------------
         ----
         TypeError                                 Traceback (most recent call l
         ast)
         Cell In[15], line 3
               1 set1 = {1,2,3}
         ----> 3 print(set1[1])

         TypeError: 'set' object is not subscriptable
```

- ***Interactive Element:***
  - A fill-in-the-blank exercise:

“Write code to create a set of your favorite colors and check if ‘blue’ is in it.”

  - Answer:

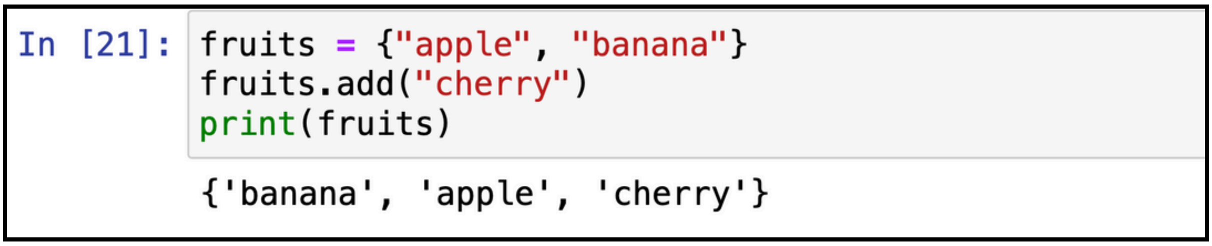


```
In [19]: colors = {"red", "blue", "green"}
         print("blue" in colors)

         True
```

Python Set Methods

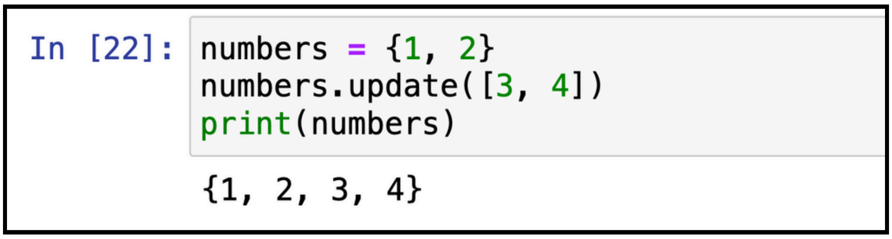
Below is a comprehensive list of Python set methods with beginner-friendly explanations, example code, and suggestions for visuals.

- **add(element): Adds a single element to the set.**



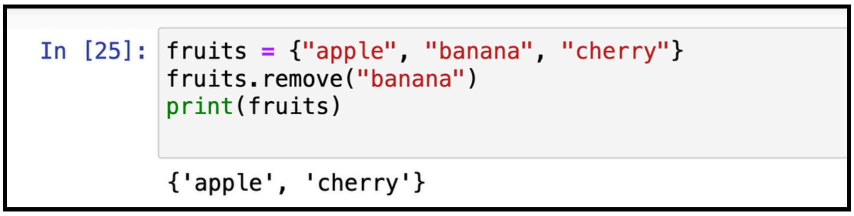
```
In [21]: fruits = {"apple", "banana"}
         fruits.add("cherry")
         print(fruits)

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

- **update(iterable): Adds multiple elements from an iterable (list, set, etc.).**



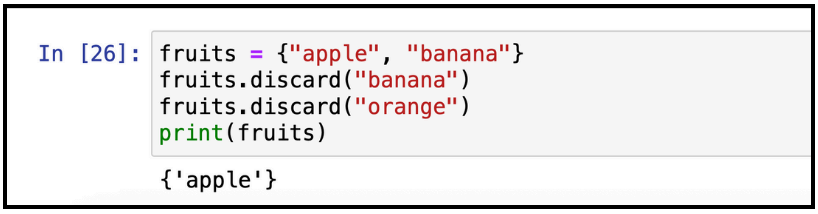
```
In [22]: numbers = {1, 2}
         numbers.update([3, 4])
         print(numbers)

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

- **remove(element): Removes an element; raises KeyError if not found.**



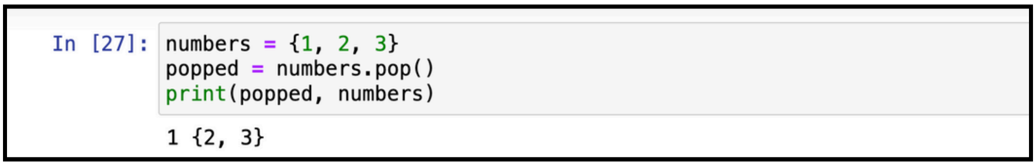
```
In [25]: fruits = {"apple", "banana", "cherry"}
         fruits.remove("banana")
         print(fruits)

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

- **discard(element): Removes an element; does not raise an error if not found**



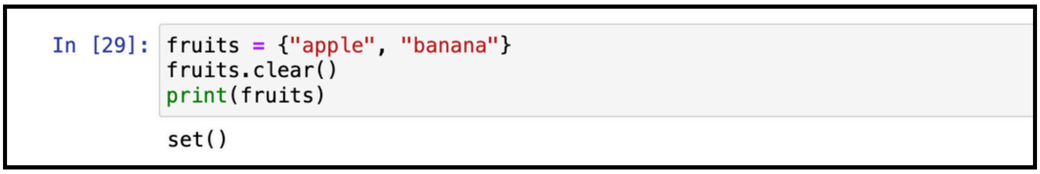
```
In [26]: fruits = {"apple", "banana"}
         fruits.discard("banana")
         fruits.discard("orange")
         print(fruits)
         {'apple'}
```

- **pop(): Removes and returns a random element; raises KeyError if set is empty.**



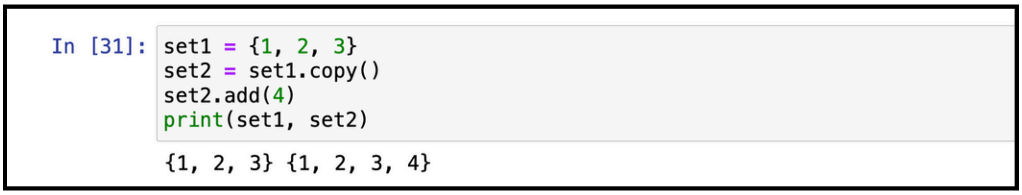
```
In [27]: numbers = {1, 2, 3}
         popped = numbers.pop()
         print(popped, numbers)
         1 {2, 3}
```

- **clear(): Removes all elements from the set.**



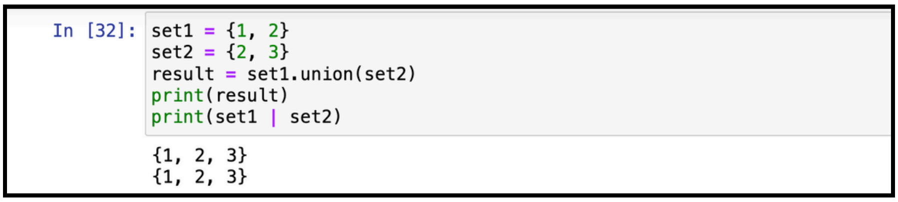
```
In [29]: fruits = {"apple", "banana"}
         fruits.clear()
         print(fruits)
         set()
```

- **copy(): Returns a shallow copy of the set.**



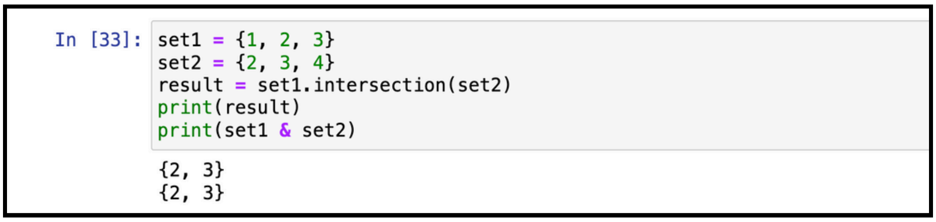
```
In [31]: set1 = {1, 2, 3}
         set2 = set1.copy()
         set2.add(4)
         print(set1, set2)
         {1, 2, 3} {1, 2, 3, 4}
```

- **union(*others) or |**: Returns a new set with elements from all sets.**



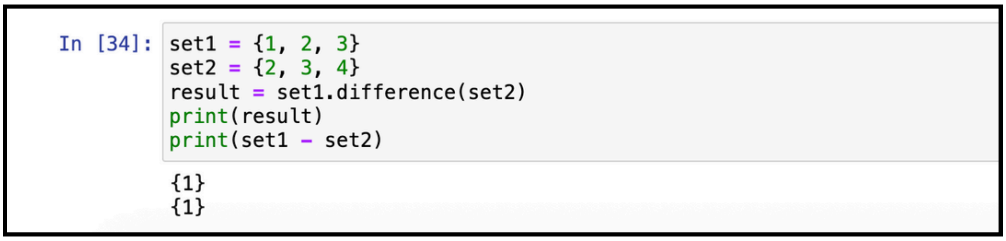
```
In [32]: set1 = {1, 2}
         set2 = {2, 3}
         result = set1.union(set2)
         print(result)
         print(set1 | set2)
         {1, 2, 3}
         {1, 2, 3}
```

- intersection(*others) or &**: Returns elements common to all sets.



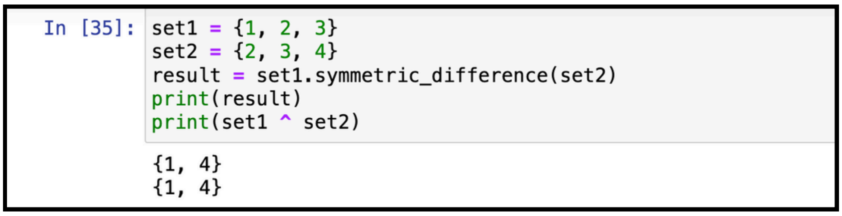
```
In [33]: set1 = {1, 2, 3}
         set2 = {2, 3, 4}
         result = set1.intersection(set2)
         print(result)
         print(set1 & set2)

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

- difference(*others) or -**: Returns elements in the first set but not in others.



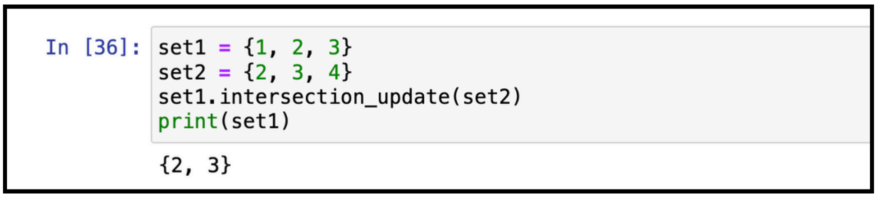
```
In [34]: set1 = {1, 2, 3}
         set2 = {2, 3, 4}
         result = set1.difference(set2)
         print(result)
         print(set1 - set2)

         {1}
         {1}
```

- symmetric_difference(other) or ^**: Returns elements in either set but not both.



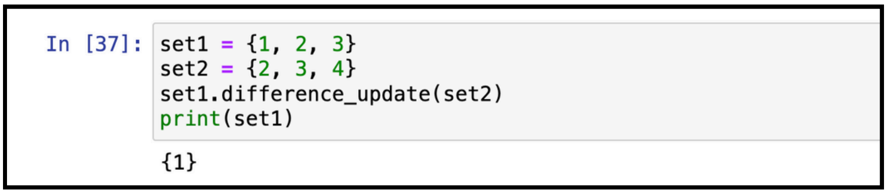
```
In [35]: set1 = {1, 2, 3}
         set2 = {2, 3, 4}
         result = set1.symmetric_difference(set2)
         print(result)
         print(set1 ^ set2)

         {1, 4}
         {1, 4}
```

- intersection_update(*others) or &=: Updates the set to keep only common elements.



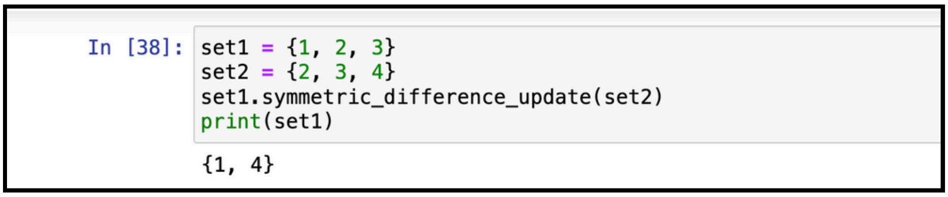
```
In [36]: set1 = {1, 2, 3}
         set2 = {2, 3, 4}
         set1.intersection_update(set2)
         print(set1)

         {2, 3}
```

- difference_update(*others) or -=: Updates the set to remove elements from others.



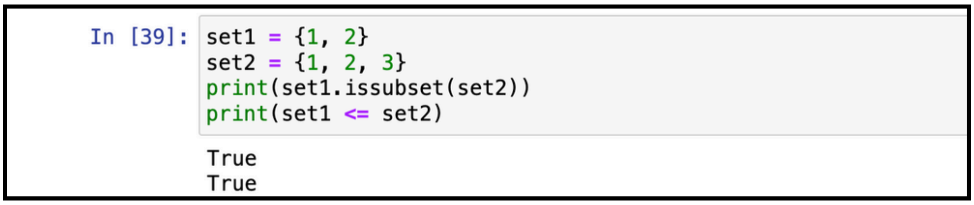
```
In [37]: set1 = {1, 2, 3}
         set2 = {2, 3, 4}
         set1.difference_update(set2)
         print(set1)

         {1}
```

- **symmetric_difference_update(other): Updates the set to keep elements in either but not both.**



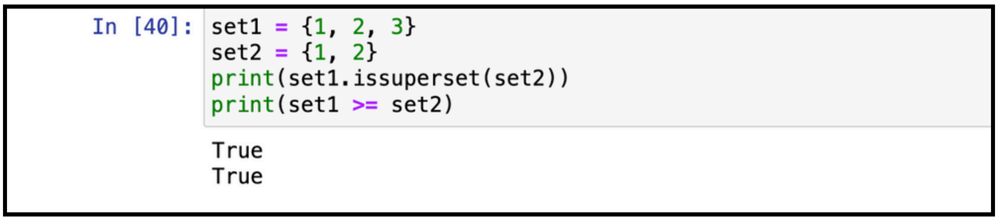
```
In [38]: set1 = {1, 2, 3}
         set2 = {2, 3, 4}
         set1.symmetric_difference_update(set2)
         print(set1)

         {1, 4}
```

- **issubset(other) or <=: Checks if the set is a subset of another.**



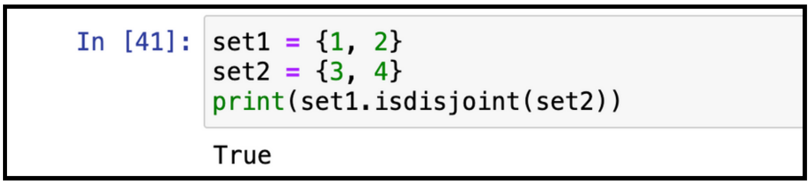
```
In [39]: set1 = {1, 2}
         set2 = {1, 2, 3}
         print(set1.issubset(set2))
         print(set1 <= set2)

         True
         True
```

- **issuperset(other) or >=: Checks if the set contains another set.**



```
In [40]: set1 = {1, 2, 3}
         set2 = {1, 2}
         print(set1.issuperset(set2))
         print(set1 >= set2)

         True
         True
```

- **isdisjoint(other): Checks if two sets have no common elements.**



```
In [41]: set1 = {1, 2}
         set2 = {3, 4}
         print(set1.isdisjoint(set2))

         True
```

Tips and Common Mistakes

- Tips:

  - Use sets for **membership testing** (faster than lists).
  - Use sets for **removing duplicates from a list**: unique = set(my_list).
  - Use **frozen sets (frozenset()) for immutable sets.**

- Common Mistakes:

  - **Trying to index a set**:

    my_set[0] (use loops instead).

  - Forgetting that **{} creates a dictionary**, not a set.
  - **Using mutable elements** like lists in sets (use tuples instead).

Conclusion and Further Resources

- **Summary**:

  Sets are powerful for handling unique elements and performing mathematical operations.

  Practice with the interactive examples to master them!

- Resources:
  - Python Official Docs: https://docs.python.org/3/tutorial/datastructures.html#sets