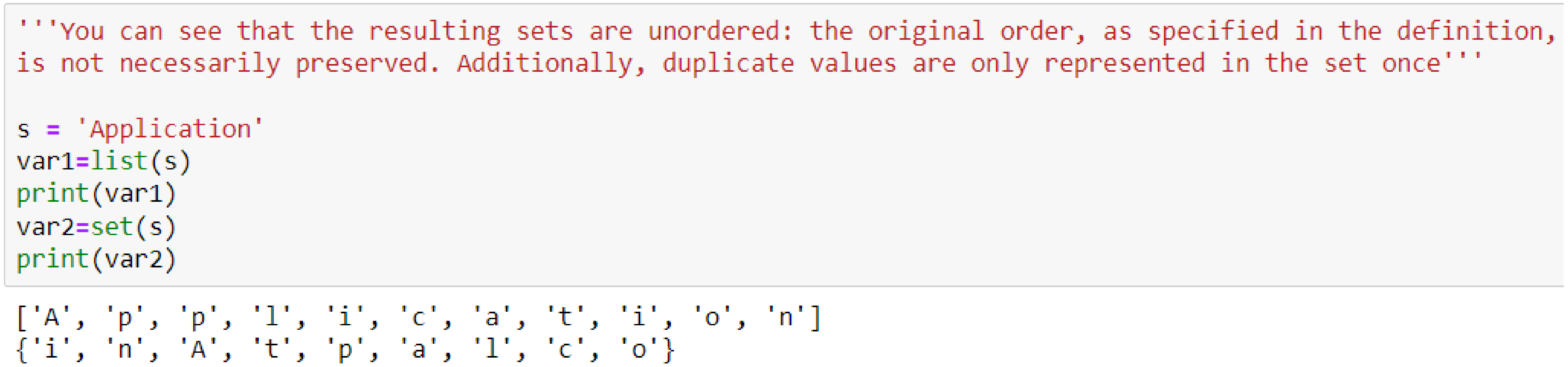
Set Data Structure

Anab Batool Kazmi

Set Data Structure

- A set in Python is a mutable, unchangeable, unordered collection of unique data elements.
- It is written with curly braces ({}) and its members can be of any type, including strings, numbers, and lists.
- Sets are useful for a variety of tasks, such as:
  - Removing duplicate elements from a list
  - Finding the union, intersection, and difference of two sets
  - Checking if an element is contained in a set
  - Iterating over the elements of a set

A set is an unordered collection

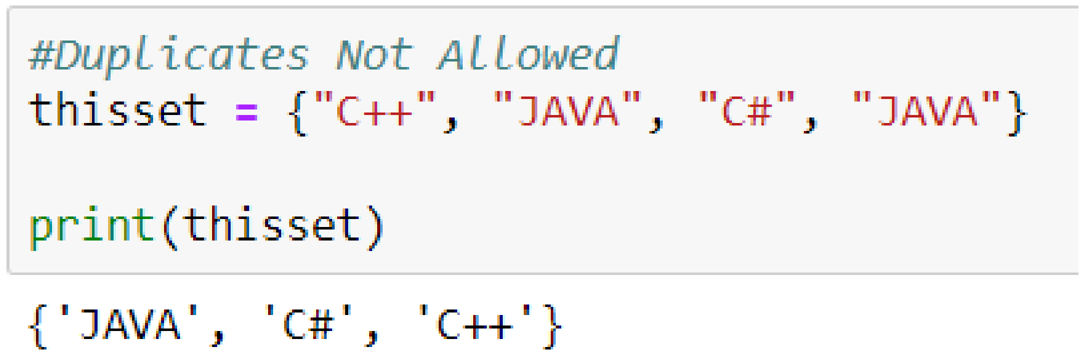


```python
'''You can see that the resulting sets are unordered: the original order, as specified in the definition,
is not necessarily preserved. Additionally, duplicate values are only represented in the set once'''

s = 'Application'
var1=list(s)
print(var1)
var2=set(s)
print(var2)
```

```
['A', 'p', 'p', 'l', 'i', 'c', 'a', 't', 'i', 'o', 'n']
{'i', 'n', 'A', 't', 'p', 'a', 'l', 'c', 'o'}
```

Duplicates Not Allowed

- In Python, adding a duplicate value to a set has no effect, and the set remains unchanged.
- Sets in Python are specifically designed to store unique elements.
- Sets automatically eliminate duplicates by not allowing multiple instances of the same element.
- If you try to add an element to a set that is already present, it won't raise an error, but the set's content will not change.
- Duplicates are ignored in sets to maintain the uniqueness of elements within the set.

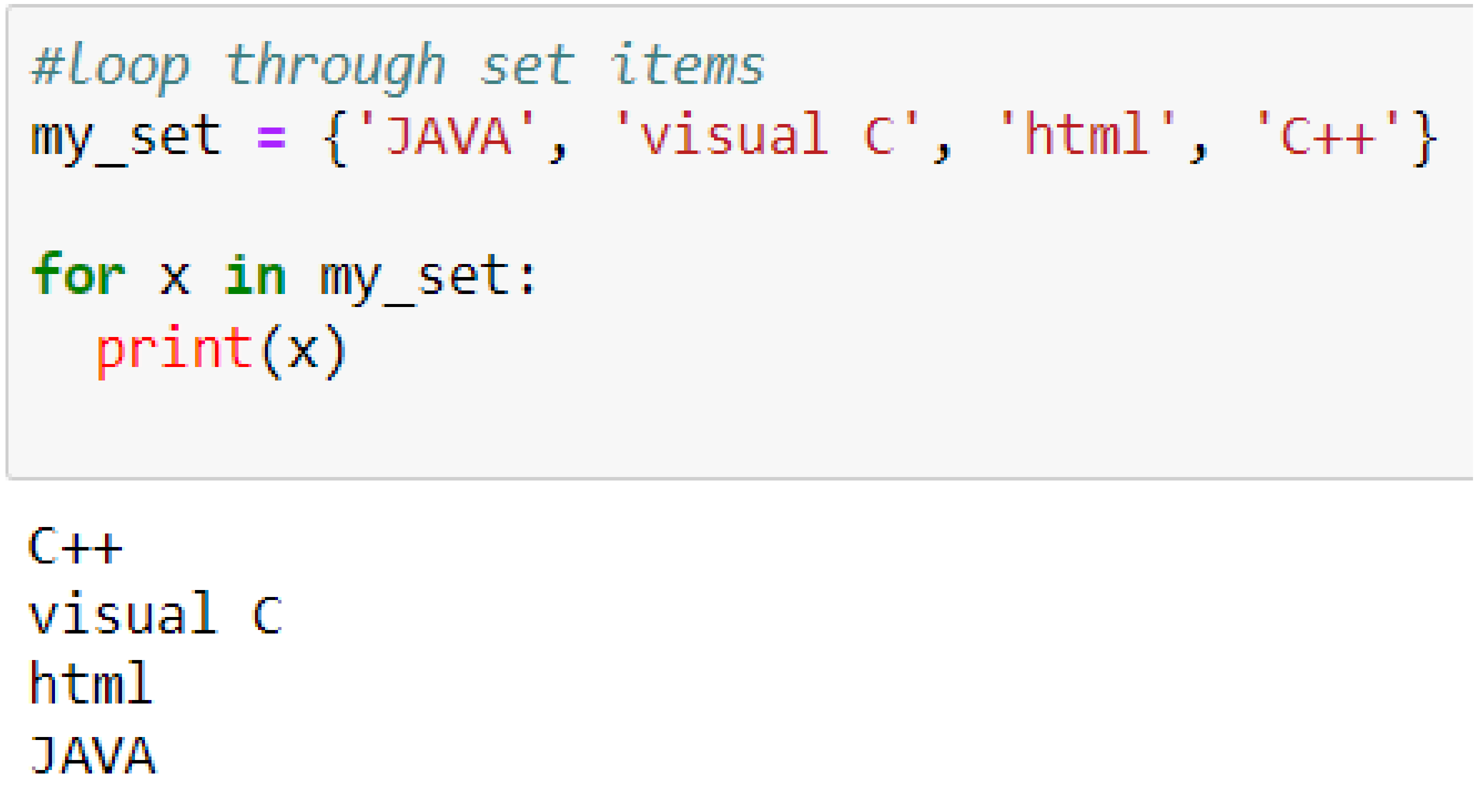


```
#Duplicates Not Allowed
thisset = {"C++", "JAVA", "C#", "JAVA"}

print(thisset)
```

```
{'JAVA', 'C#', 'C++'}
```

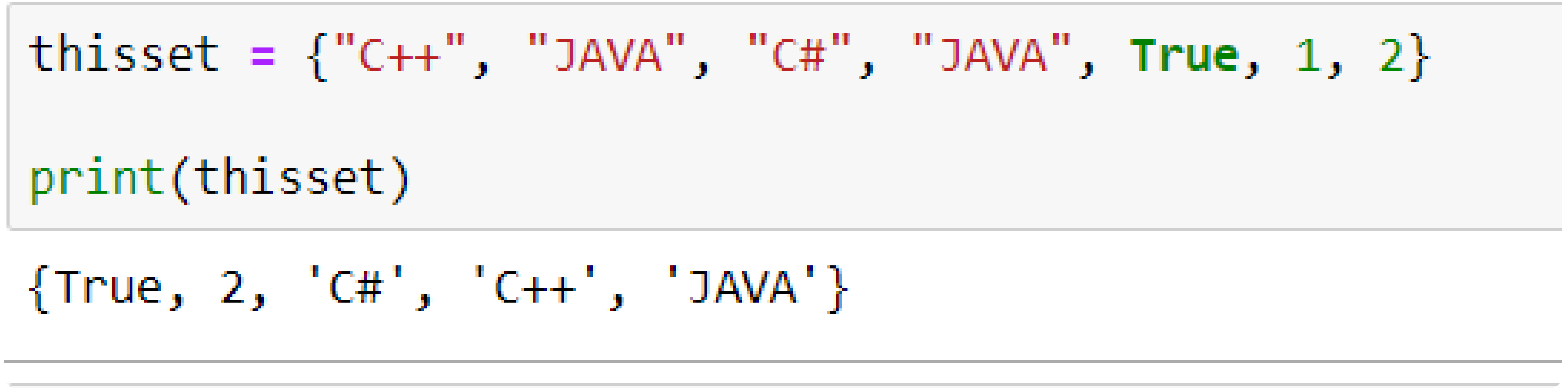
Python - Loop Sets

- You can loop through the set items by using a for loop



```python
#loop through set items
my_set = {'JAVA', 'visual C', 'html', 'C++'}

for x in my_set:
  print(x)
```

```
C++
visual C
html
JAVA
```



```python
thisset = {"C++", "JAVA", "C#", "JAVA", True, 1, 2}

print(thisset)
```

```
{True, 2, 'C#', 'C++', 'JAVA'}
```

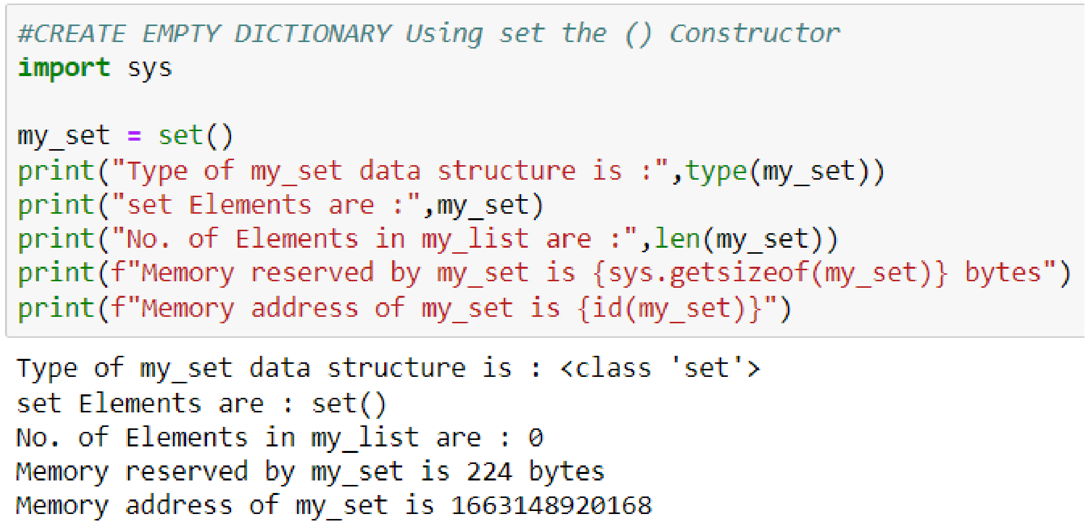
Duplicates Not Allowed

- The values True and 1 are considered the same value in sets, and are treated as duplicates:

how to create empty set in python

- Using the set() Constructor:

  empty_set = set()



```python
#CREATE EMPTY DICTIONARY Using set the () Constructor
import sys

my_set = set()
print("Type of my_set data structure is :",type(my_set))
print("set Elements are :",my_set)
print("No. of Elements in my_list are :",len(my_set))
print(f"Memory reserved by my_set is {sys.getsizeof(my_set)} bytes")
print(f"Memory address of my_set is {id(my_set)}")
```

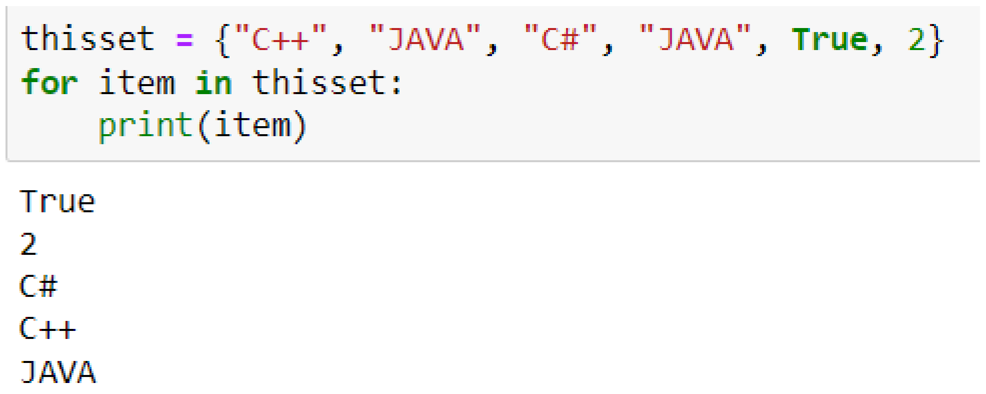
```
Type of my_set data structure is : <class 'set'>
set Elements are : set()
No. of Elements in my_list are : 0
Memory reserved by my_set is 224 bytes
Memory address of my_set is 1663148920168
```

Access Set Items

- In Python, you can access set items **through iteration or by checking for membership** of a specific element.
- Sets are unordered collections of unique elements, so **you cannot access set items by index** like you would with lists or tuples.

Access Set Items

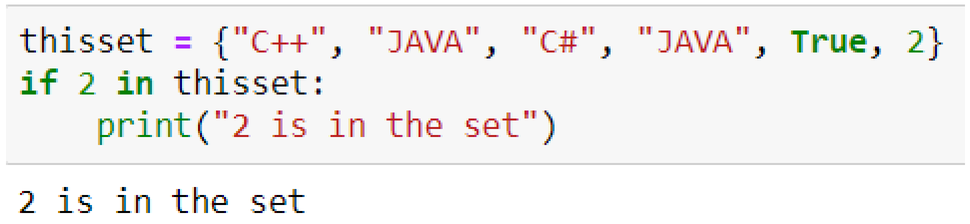
Iteration



```python
thisset = {"C++", "JAVA", "C#", "JAVA", True, 2}
for item in thisset:
    print(item)
```

```
True
2
C#
C++
JAVA
```

Membership Check



```python
thisset = {"C++", "JAVA", "C#", "JAVA", True, 2}
if 2 in thisset:
    print("2 is in the set")
```

```
2 is in the set
```

Set *items* are unchangeable

- Set *items* are unchangeable, but you can **remove items and add new items**.
- Set items are unchangeable, meaning that we cannot change the items after the set has been created.
- Once a set is created, you cannot change its items, but you can remove items and add new items.

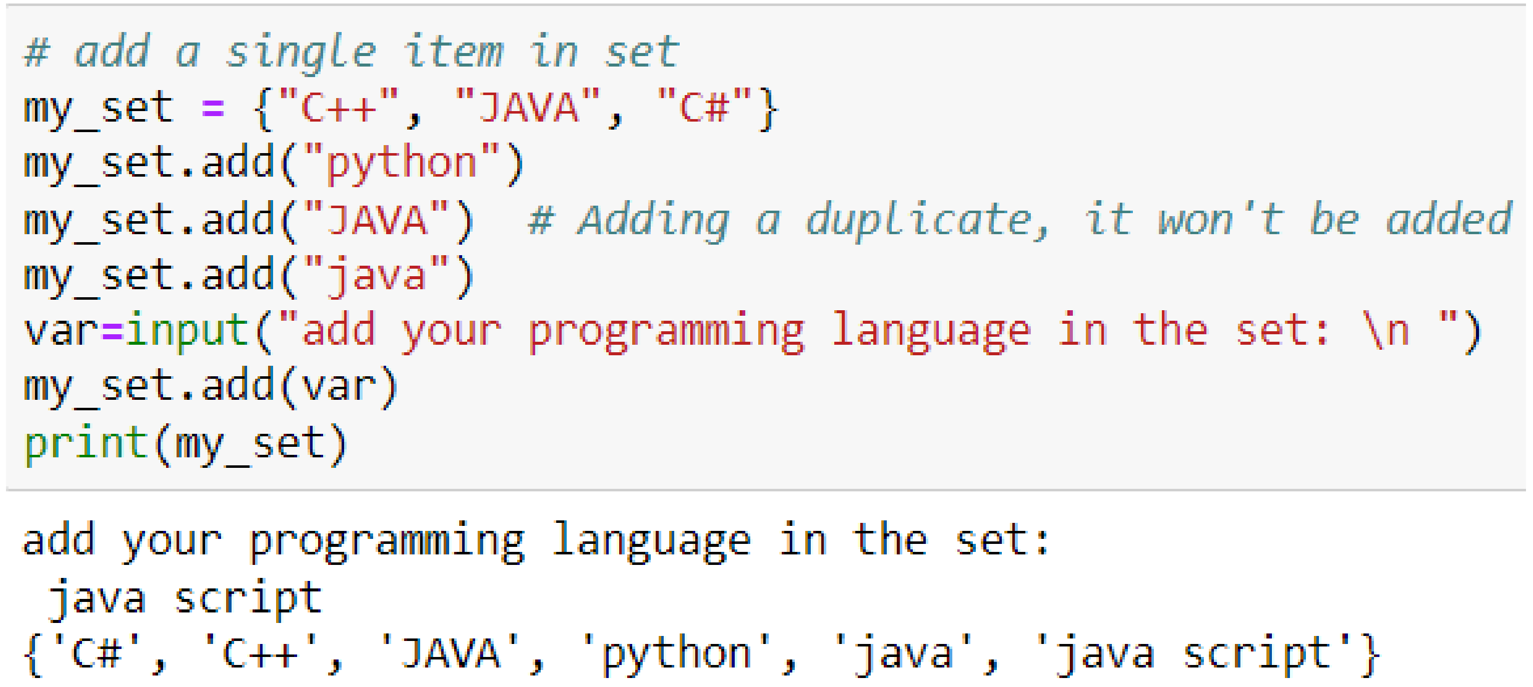
Add Set Items

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

- Using the add() Method
  - The add() method is used to add a single element to a set.
- Using the update() Method
  - The update() method is used to add multiple items to a set.
- Using Set Union
  - You can use set operations to add elements from one set to another. The union() method or the | operator can be used to combine two sets.
- Using Set Comprehension
  - You can create a new set by using a set comprehension. This allows you to add items conditionally or based on some logic.

Add Set Items- add() method

- The add() method is used to add a single element to a set. If the element is already in the set, it won't be added again (sets do not allow duplicate elements).

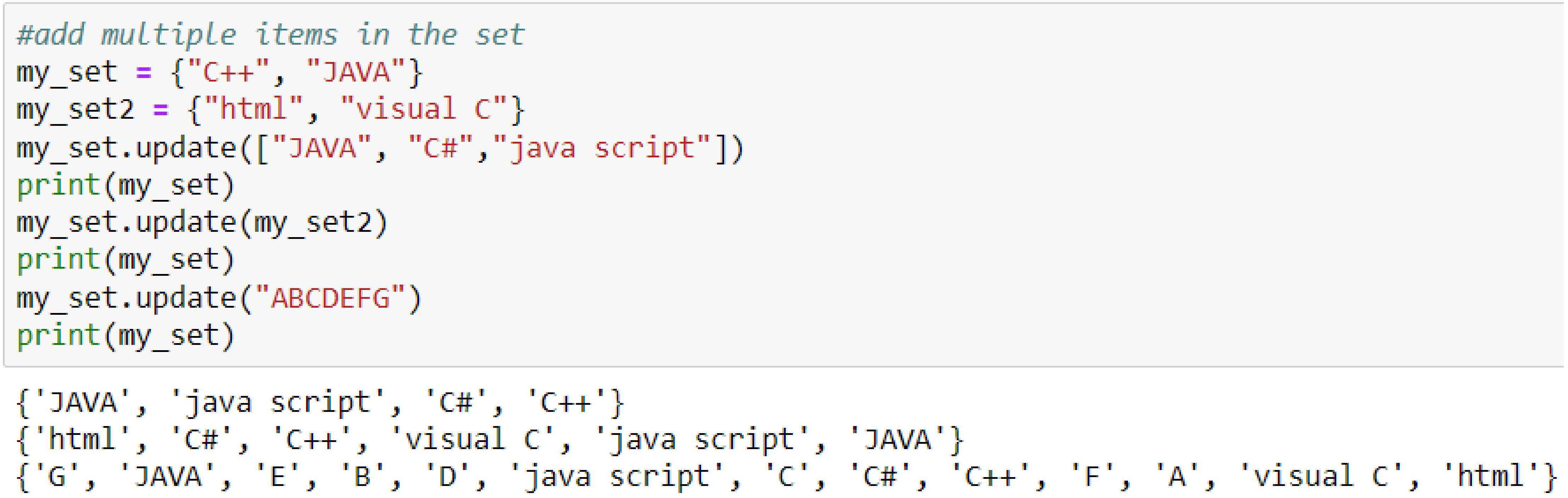


```python
# add a single item in set
my_set = {"C++", "JAVA", "C#"}
my_set.add("python")
my_set.add("JAVA")  # Adding a duplicate, it won't be added
my_set.add("java")
var=input("add your programming language in the set: \n ")
my_set.add(var)
print(my_set)
```

```
add your programming language in the set:
 java script
{'C#', 'C++', 'JAVA', 'python', 'java', 'java script'}
```

Add Set Items- update() method

- The update() method is used to add multiple items to a set. You can pass another iterable (e.g., a list, tuple, or another set) as an argument to update().

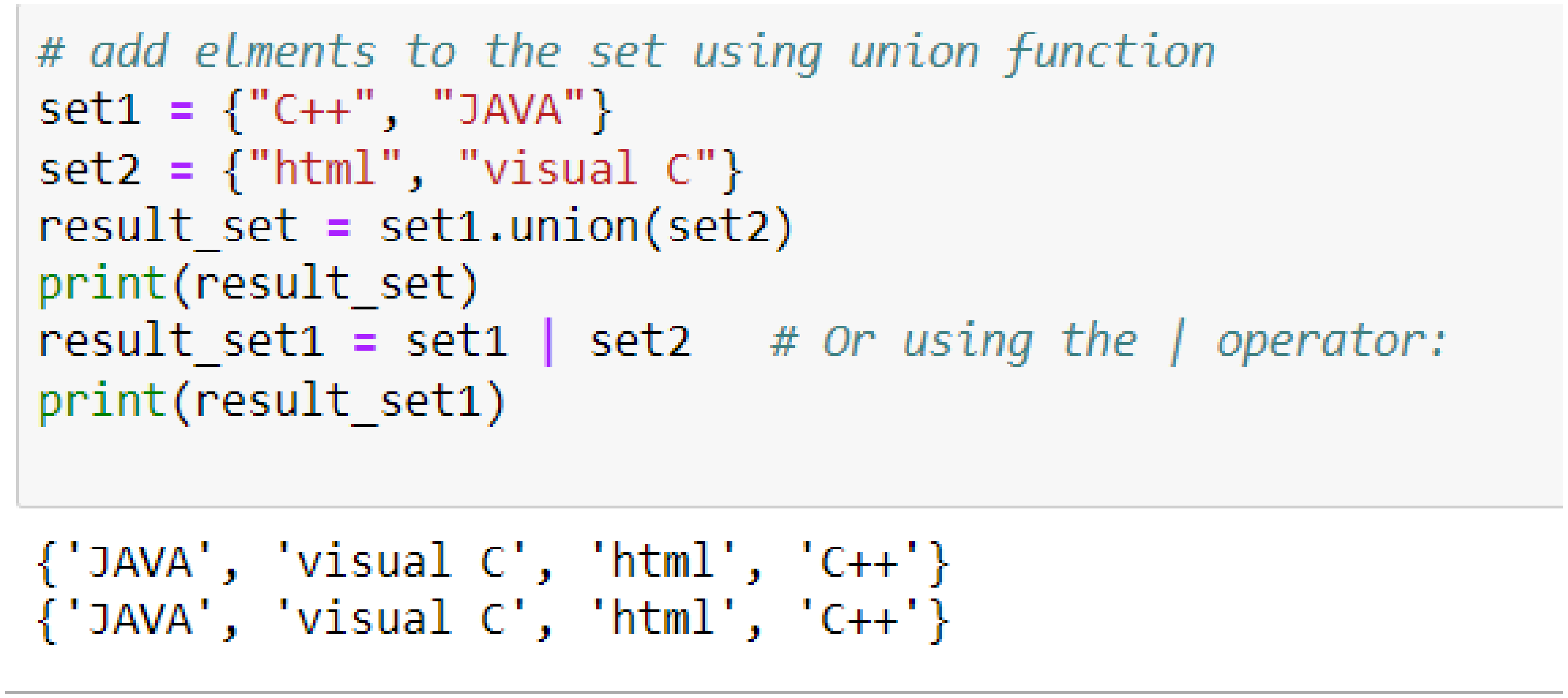


```
#add multiple items in the set
my_set = {"C++", "JAVA"}
my_set2 = {"html", "visual C"}
my_set.update(["JAVA", "C#","java script"])
print(my_set)
my_set.update(my_set2)
print(my_set)
my_set.update("ABCDEFG")
print(my_set)
```

```
{'JAVA', 'java script', 'C#', 'C++'}
{'html', 'C#', 'C++', 'visual C', 'java script', 'JAVA'}
{'G', 'JAVA', 'E', 'B', 'D', 'java script', 'C', 'C#', 'C++', 'F', 'A', 'visual C', 'html'}
```

Add Set Items- union() method

- You can use set operations to add elements from one set to another. The union() method or the | operator can be used to combine two sets.



```
# add elments to the set using union function
set1 = {"C++", "JAVA"}
set2 = {"html", "visual C"}
result_set = set1.union(set2)
print(result_set)
result_set1 = set1 | set2   # Or using the | operator:
print(result_set1)
```

```
{'JAVA', 'visual C', 'html', 'C++'}
{'JAVA', 'visual C', 'html', 'C++'}
```

Add Set Items- Using Set Comprehension

- Set comprehension is a concise way to create sets in Python by **applying an expression to each item in an iterable** (e.g., a list, tuple, or another iterable) and **optionally filtering the items based on a condition.**
- It is similar to list comprehension but results in a set instead of a list.
- Set comprehensions use curly braces {}.

Syntax:

```
new_set = {expression for item in iterable if condition}
```

Add Set Items- Using Set Comprehension

new_set = {expression for item in iterable if condition}

expression: The expression to apply to each item in the iterable.

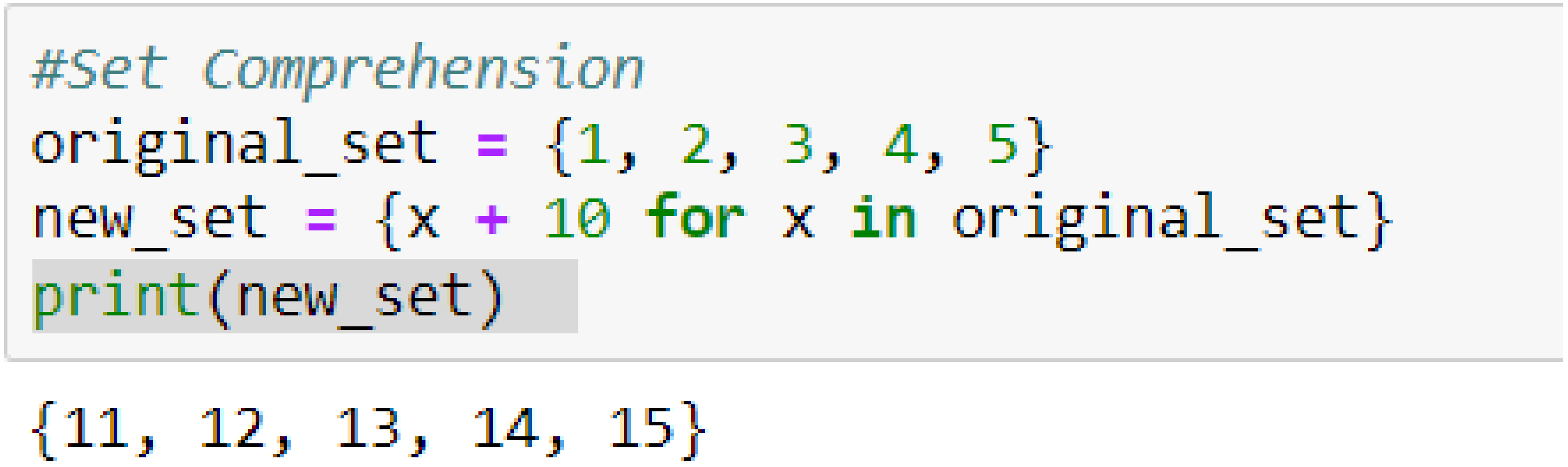
item: A variable that represents each item in the iterable.

iterable: The source iterable (e.g., list, tuple, or another iterable) from which items are taken.

condition (optional): A condition that filters items before they are included in the resulting set. This part is optional.

Add Set Items- Using Set Comprehension

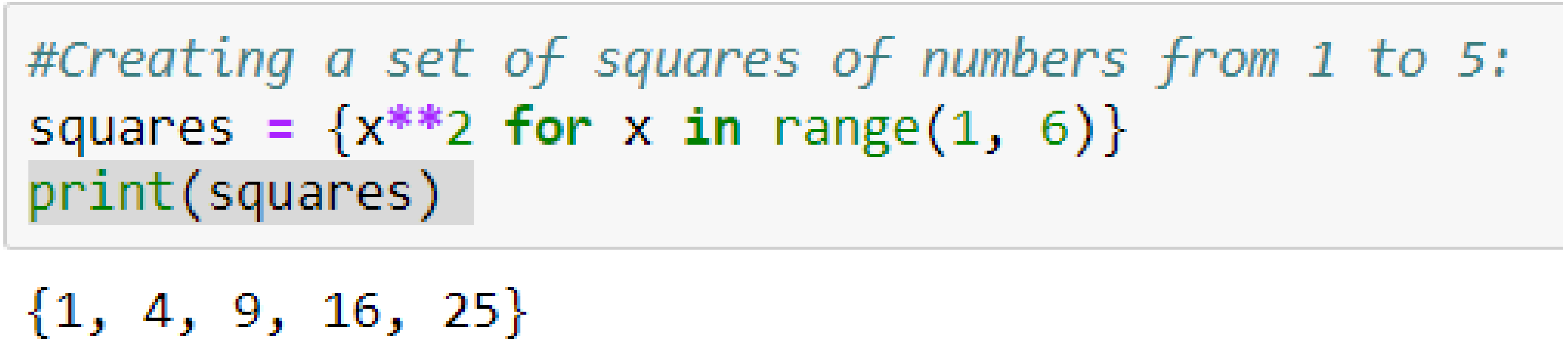
Creating a new set by adding 10 to every element of existing set:



```
#Set Comprehension
original_set = {1, 2, 3, 4, 5}
new_set = {x + 10 for x in original_set}
print(new_set)
```

```
{11, 12, 13, 14, 15}
```

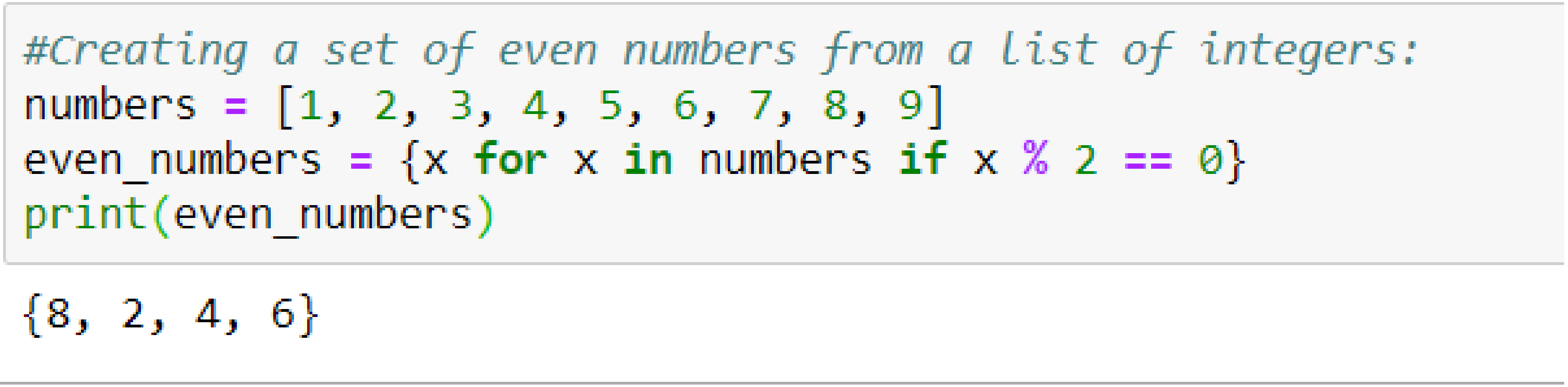
Add Set Items- Using Set Comprehension



```python
#Creating a set of squares of numbers from 1 to 5:
squares = {x**2 for x in range(1, 6)}
print(squares)
```

```
{1, 4, 9, 16, 25}
```

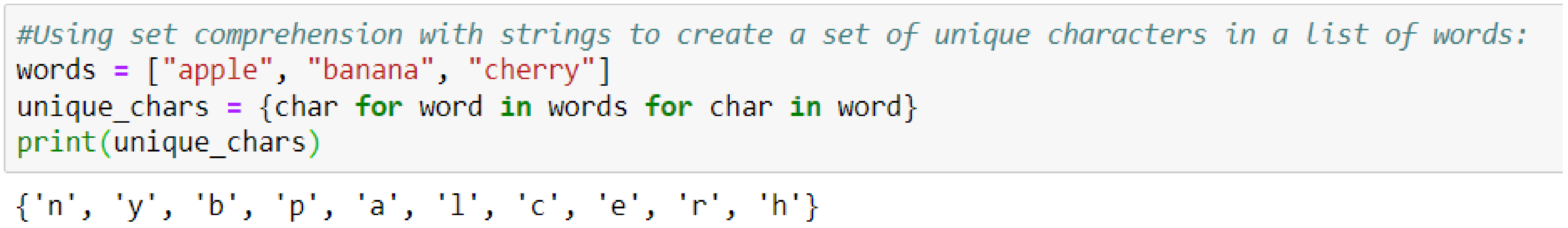
Add Set Items- Using Set Comprehension



```
#Creating a set of even numbers from a list of integers:
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
even_numbers = {x for x in numbers if x % 2 == 0}
print(even_numbers)
```

```
{8, 2, 4, 6}
```

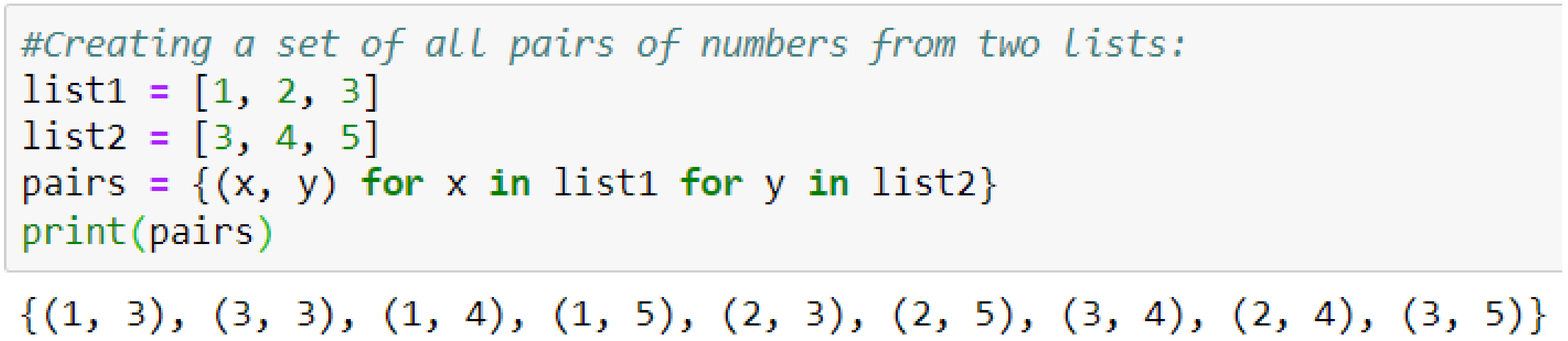
Add Set Items- Using Set Comprehension



```python
#Using set comprehension with strings to create a set of unique characters in a list of words:
words = ["apple", "banana", "cherry"]
unique_chars = {char for word in words for char in word}
print(unique_chars)
```

```
{'n', 'y', 'b', 'p', 'a', 'l', 'c', 'e', 'r', 'h'}
```

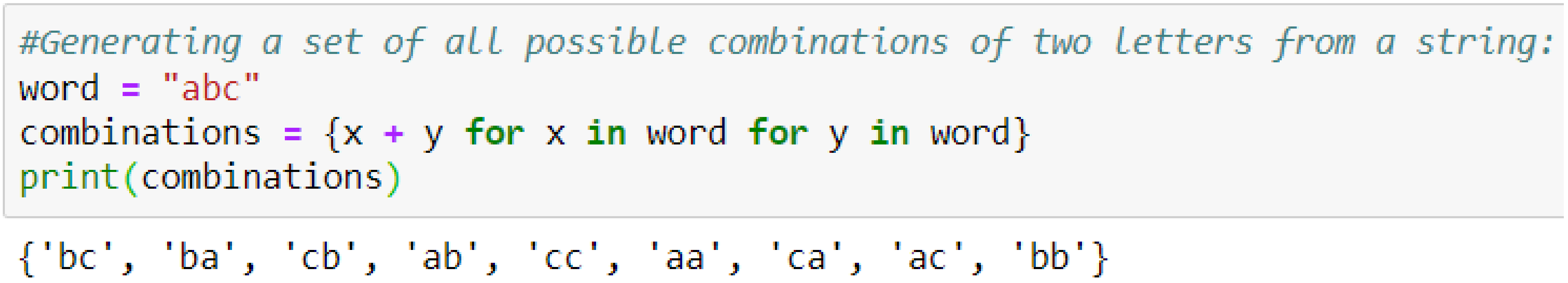
Add Set Items- Using Set Comprehension



```
#Creating a set of all pairs of numbers from two lists:
list1 = [1, 2, 3]
list2 = [3, 4, 5]
pairs = {(x, y) for x in list1 for y in list2}
print(pairs)
```

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

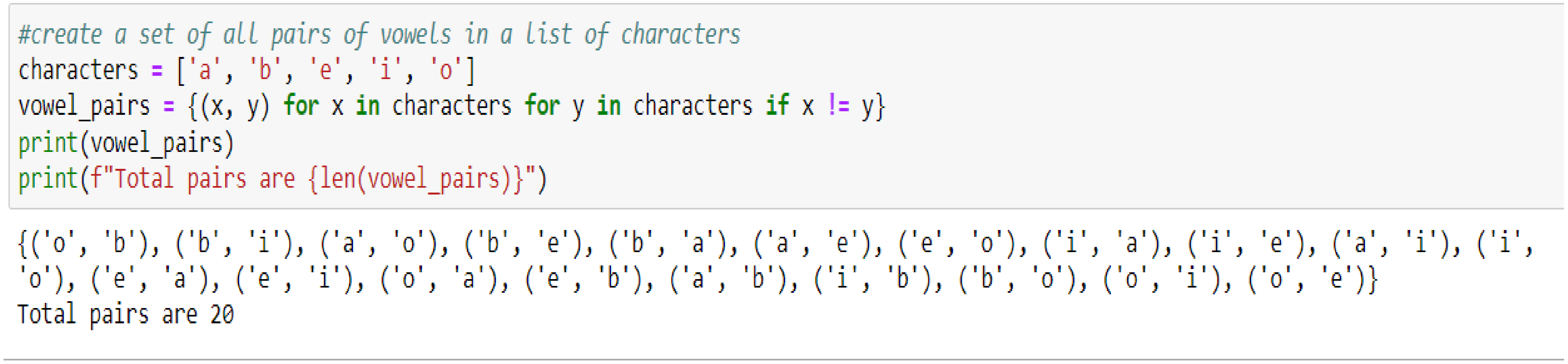
Add Set Items- Using Set Comprehension



```python
#Generating a set of all possible combinations of two letters from a string:
word = "abc"
combinations = {x + y for x in word for y in word}
print(combinations)
```

```
{'bc', 'ba', 'cb', 'ab', 'cc', 'aa', 'ca', 'ac', 'bb'}
```

Add Set Items- Using Set Comprehension



```python
#create a set of all pairs of vowels in a list of characters
characters = ['a', 'b', 'e', 'i', 'o']
vowel_pairs = {(x, y) for x in characters for y in characters if x != y}
print(vowel_pairs)
print(f"Total pairs are {len(vowel_pairs)}")
```

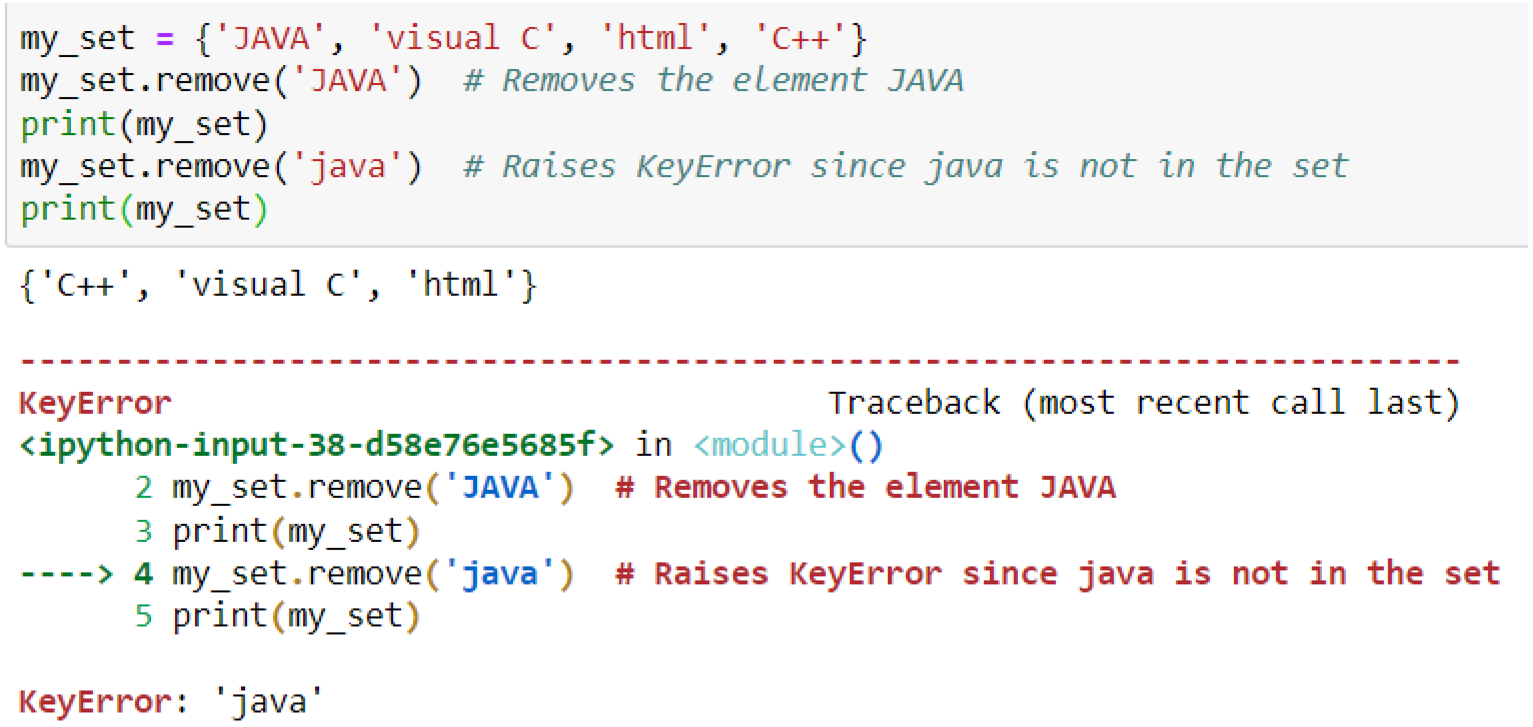
```
{('o', 'b'), ('b', 'i'), ('a', 'o'), ('b', 'e'), ('b', 'a'), ('a', 'e'), ('e', 'o'), ('i', 'a'), ('i', 'e'), ('a', 'i'), ('i',
'o'), ('e', 'a'), ('e', 'i'), ('o', 'a'), ('e', 'b'), ('a', 'b'), ('i', 'b'), ('b', 'o'), ('o', 'i'), ('o', 'e')}
Total pairs are 20
```

Python - Remove Set Items

- Using the remove() Method
  - remove a specific element from the set by specifying its value(raise exception)
- Using the discard() Method
  - remove a specific element from the set by specifying its value
- Using the pop() Method
  - removes and returns an arbitrary (random) element from the set
- Using the clear() method
  - empties the set
- Using the del keyword
  - will delete the set completely

Using the remove() Method:

- The remove() method is used to remove a specific element from the set by specifying its value.
- If the element is not found in the set, it raises a KeyError

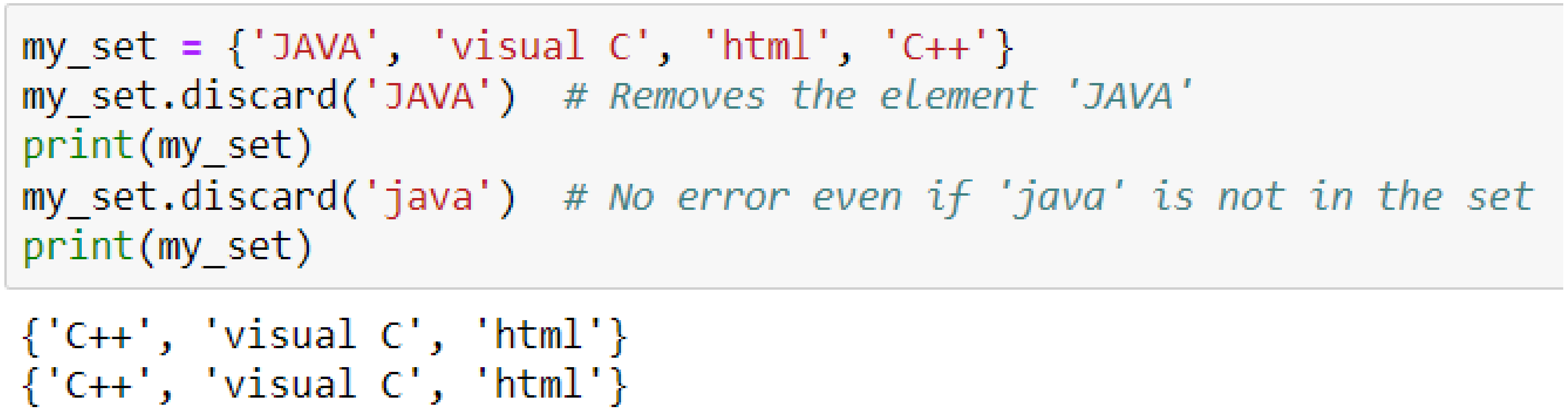


```
my_set = {'JAVA', 'visual C', 'html', 'C++'}
my_set.remove('JAVA')  # Removes the element JAVA
print(my_set)
my_set.remove('java')  # Raises KeyError since java is not in the set
print(my_set)
```

```
{'C++', 'visual C', 'html'}

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-38-d58e76e5685f> in <module>()
      2 my_set.remove('JAVA')  # Removes the element JAVA
      3 print(my_set)
----> 4 my_set.remove('java')  # Raises KeyError since java is not in the set
      5 print(my_set)

KeyError: 'java'
```

Using the discard() Method

- The discard() method is similar to remove(), but it does not raise an error if the element is not found in the set

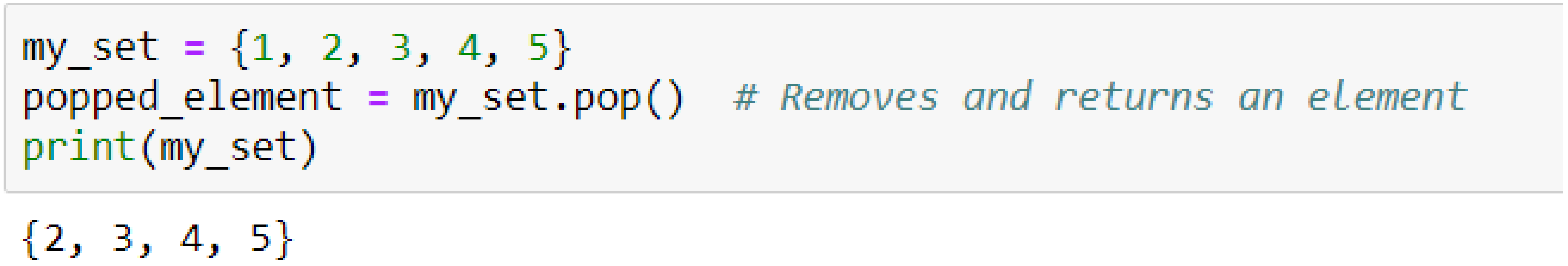


```
my_set = {'JAVA', 'visual C', 'html', 'C++'}
my_set.discard('JAVA')  # Removes the element 'JAVA'
print(my_set)
my_set.discard('java')  # No error even if 'java' is not in the set
print(my_set)
```

```
{'C++', 'visual C', 'html'}
{'C++', 'visual C', 'html'}
```

Using the pop() Method:

- The pop() method removes and returns an arbitrary (random) element from the set. Since sets are unordered, you won't know which element will be removed



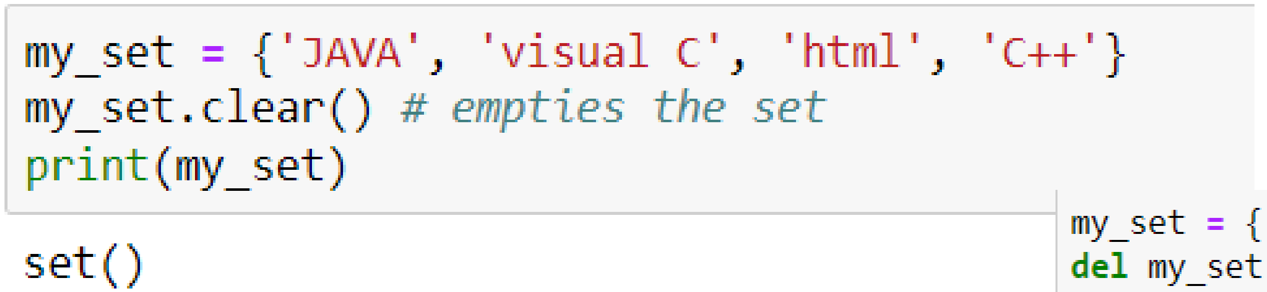
```python
my_set = {1, 2, 3, 4, 5}
popped_element = my_set.pop()  # Removes and returns an element
print(my_set)
```

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

clear() Method- the del keyword

Using the clear() Method

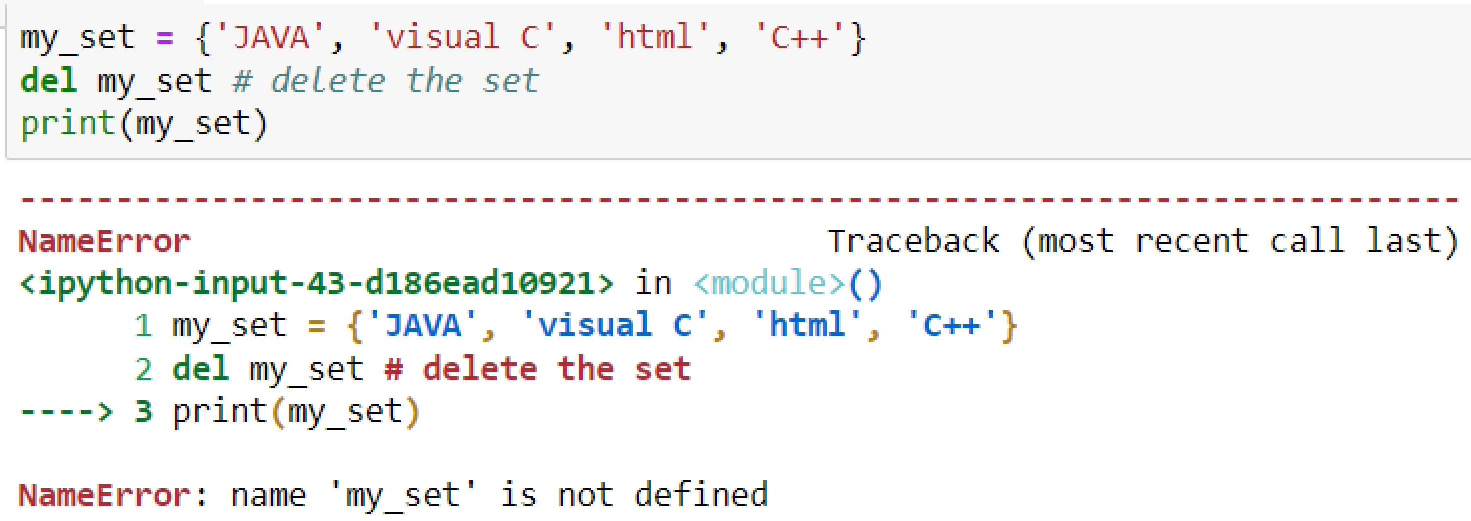
- Empty the set



```
my_set = {'JAVA', 'visual C', 'html', 'C++'}
my_set.clear() # empties the set
print(my_set)

set()
```

Using the del keyword

- Delete the set from memory

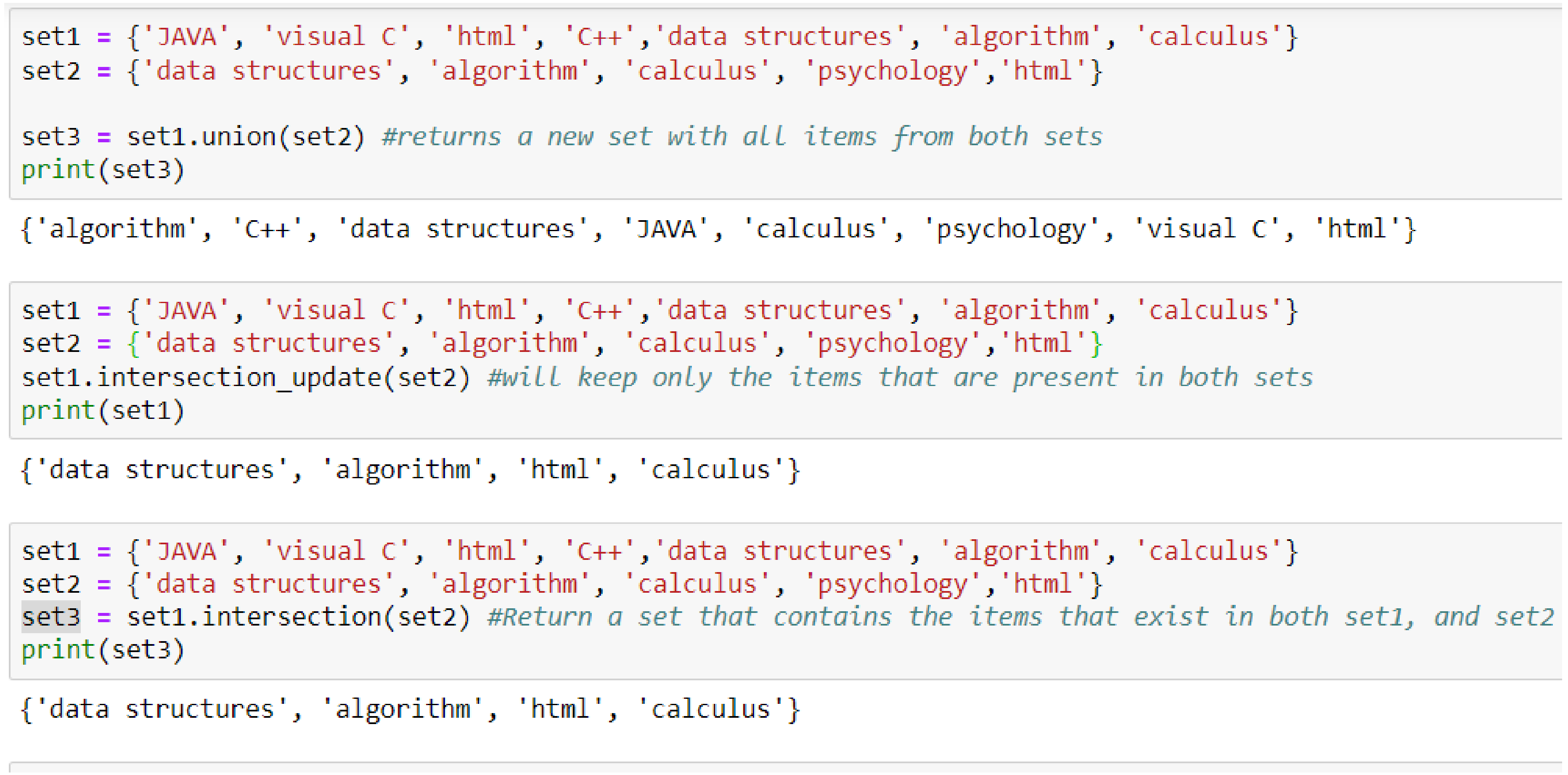


```
my_set = {'JAVA', 'visual C', 'html', 'C++'}
del my_set # delete the set
print(my_set)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-43-d186ead10921> in <module>()
      1 my_set = {'JAVA', 'visual C', 'html', 'C++'}
      2 del my_set # delete the set
----> 3 print(my_set)

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

Python - Join Sets using set operations

- The union() method
  - returns a new set containing all items from both sets
- intersection_update()
  - keep only the items that are present in both sets
- intersection()
  - return a new set, that only contains the items that are present in both sets
- symmetric_difference_update() method
  - will keep only the elements that are NOT present in both sets
- symmetric_difference()
  - will return a new set, that contains only the elements that are NOT present in both sets.

Python - Join Sets using set operations



```
set1 = {'JAVA', 'visual C', 'html', 'C++','data structures', 'algorithm', 'calculus'}
set2 = {'data structures', 'algorithm', 'calculus', 'psychology','html'}

set3 = set1.union(set2) #returns a new set with all items from both sets
print(set3)
```

```
{'algorithm', 'C++', 'data structures', 'JAVA', 'calculus', 'psychology', 'visual C', 'html'}
```

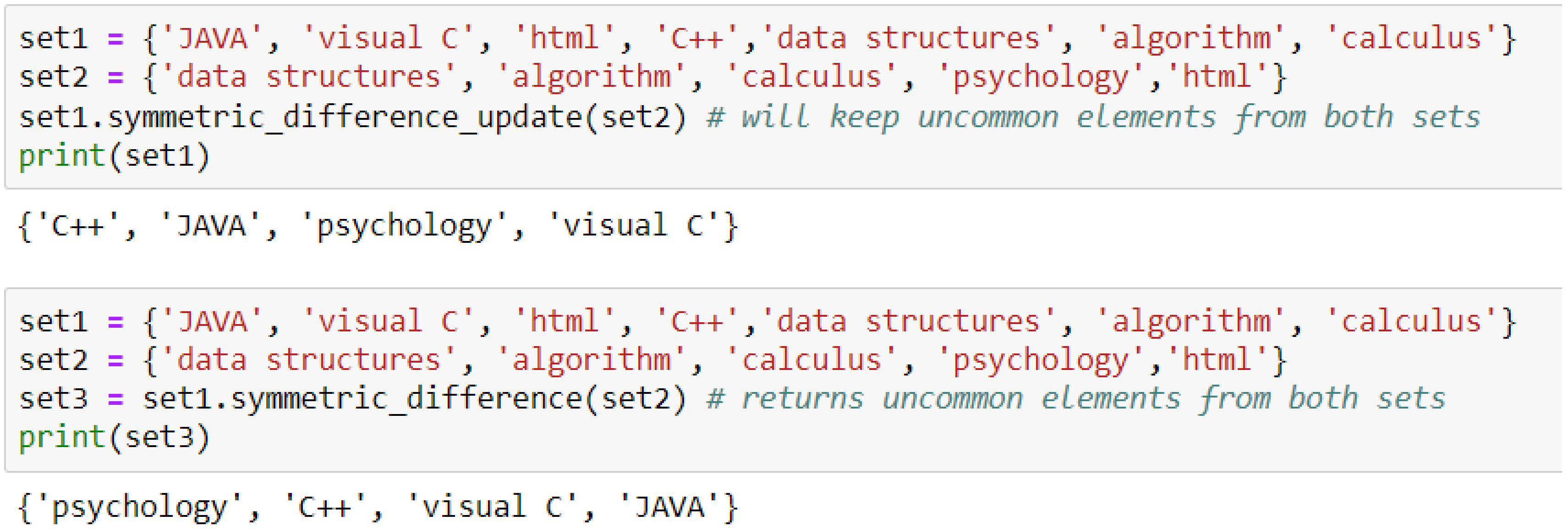
```
set1 = {'JAVA', 'visual C', 'html', 'C++','data structures', 'algorithm', 'calculus'}
set2 = {'data structures', 'algorithm', 'calculus', 'psychology','html'}
set1.intersection_update(set2) #will keep only the items that are present in both sets
print(set1)
```

```
{'data structures', 'algorithm', 'html', 'calculus'}
```

```
set1 = {'JAVA', 'visual C', 'html', 'C++','data structures', 'algorithm', 'calculus'}
set2 = {'data structures', 'algorithm', 'calculus', 'psychology','html'}
set3 = set1.intersection(set2) #Return a set that contains the items that exist in both set1, and set2
print(set3)
```

```
{'data structures', 'algorithm', 'html', 'calculus'}
```

Python - Join Sets using set operations



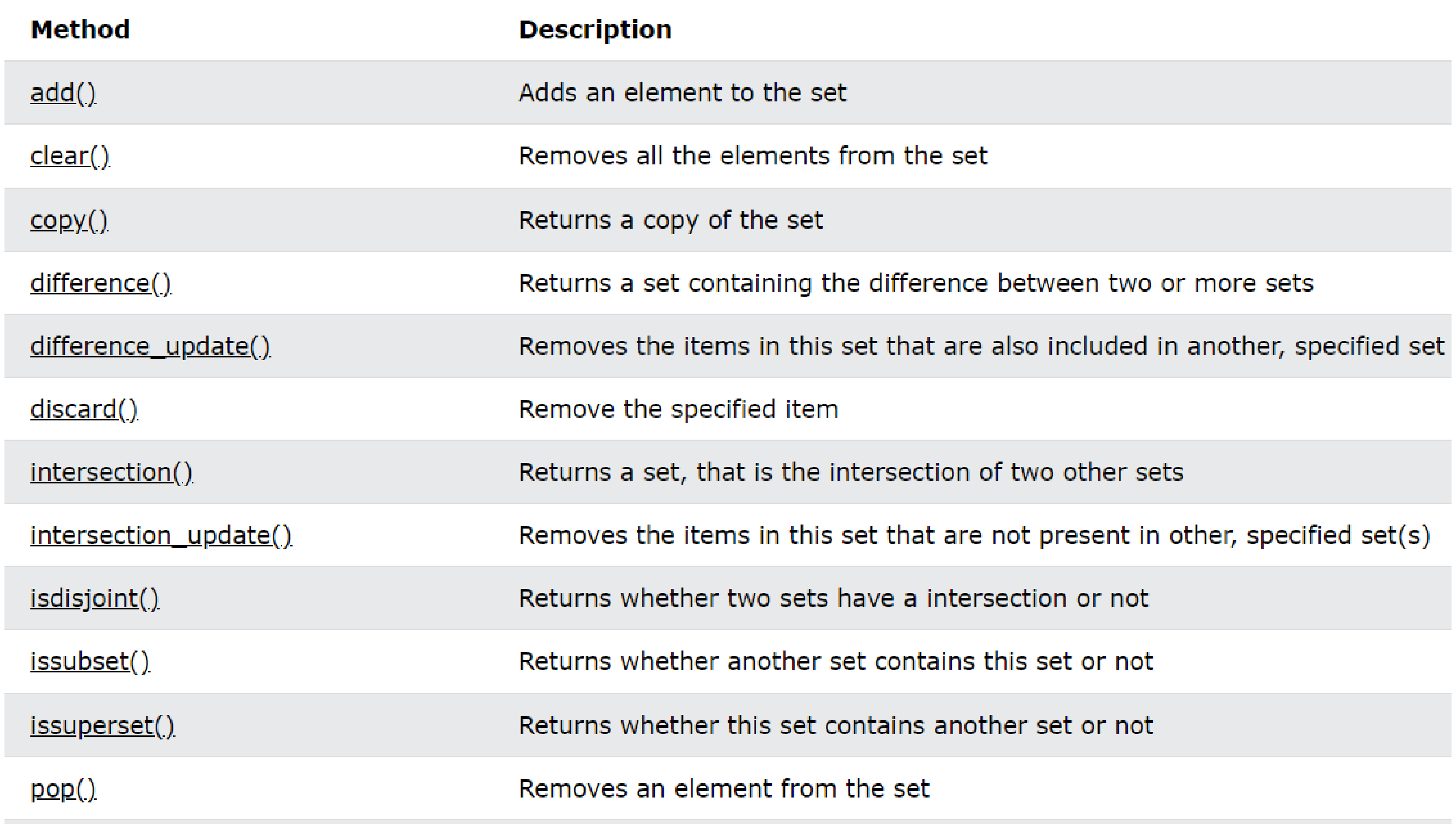
```
set1 = {'JAVA', 'visual C', 'html', 'C++','data structures', 'algorithm', 'calculus'}
set2 = {'data structures', 'algorithm', 'calculus', 'psychology','html'}
set1.symmetric_difference_update(set2) # will keep uncommon elements from both sets
print(set1)
```

```
{'C++', 'JAVA', 'psychology', 'visual C'}
```

```
set1 = {'JAVA', 'visual C', 'html', 'C++','data structures', 'algorithm', 'calculus'}
set2 = {'data structures', 'algorithm', 'calculus', 'psychology','html'}
set3 = set1.symmetric_difference(set2) # returns uncommon elements from both sets
print(set3)
```

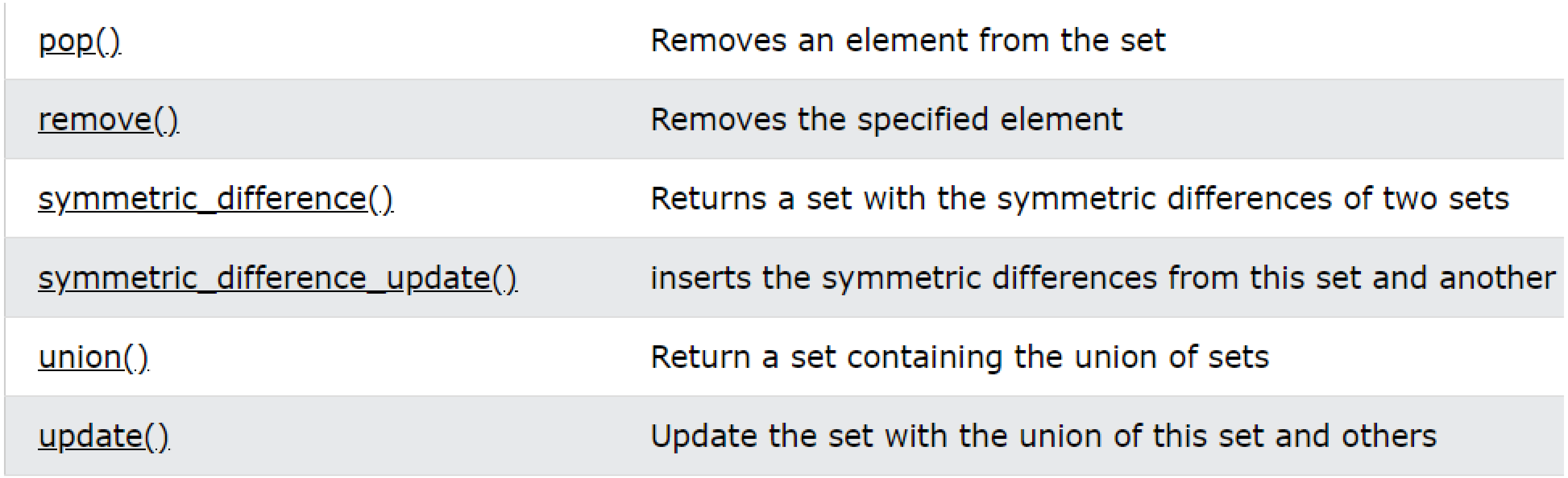
```
{'psychology', 'C++', 'visual C', 'JAVA'}
```

Python - Set Methods



Method	Description
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set

Python - Set Methods



pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

Reference

- W3school.com