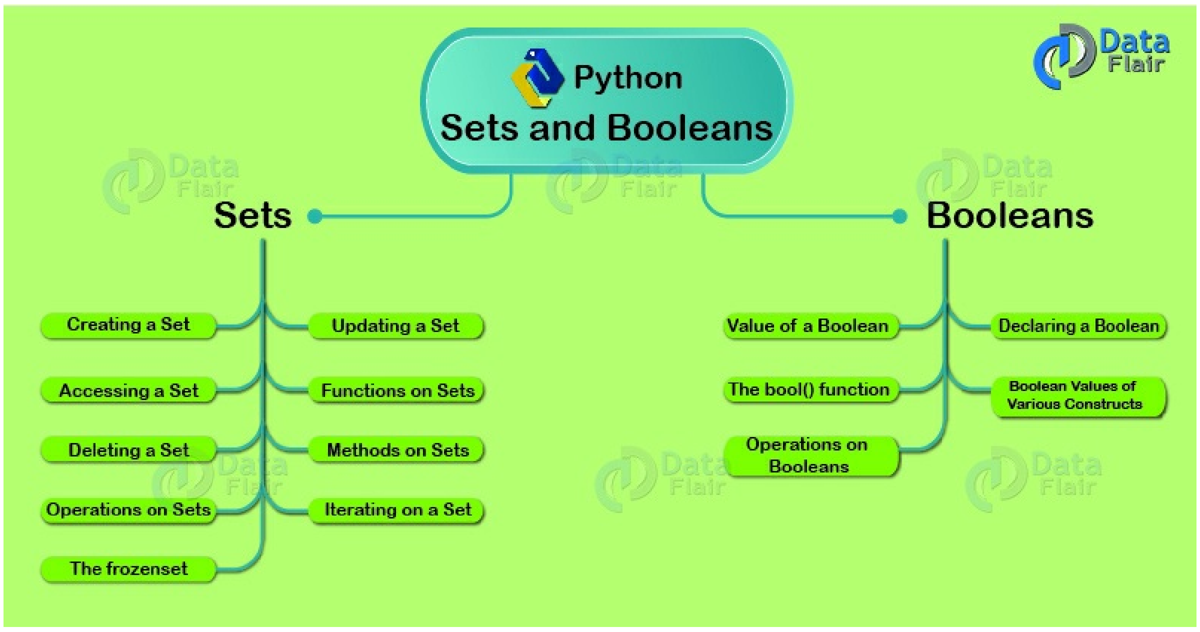
Python Set and Booleans with Syntax and Examples

Contents

1. Python Set and Python Booleans

So far, we have learned about various data types in Python. We dealt with strings, numbers, lists, tuples, and dictionaries. We've also learned that we don't need to declare the type of data while defining it. Today, we will talk about Python set examples and Python Booleans. After that, we will move onto python functions in further lessons.



Python Set and Python Booleans

2. Sets in Python

First, we focus on Python sets. A set in Python holds a sequence of values. It is sequenced, but does not support indexing. We will understand that as we get deeper into the article with the Python set Examples.

a. Creating a Python Set

To declare a set, you need to type a sequence of items separated by commas, inside curly braces. After that, assign it to a Python variable.

```
>>> a={1,3,2}
```

As you can see, we wrote it in the order 1, 3, 2. In point b, we will access this set and see what we get back.

A set may contain values of different types.

```
>>> c={1,2.0,'three'}
```

1. Duplicate Elements

A set also cannot contain duplicate elements. Let's try adding duplicate elements to another set, and then access it in point b.

```
>>> b={3,2,1,2}
```

2. Mutability

A set is mutable, but may not contain mutable items like a list, set, or even a dictionary.

```
>>> d={[1,2,3],4}
Traceback(most recent call last):
File "<pyshell#9>", line 1, in <module>
```

4. d={[1,2,3],4}

TypeError: unhashable type: ‘list’

As we discussed, there is no such thing as a nested Python set.

```
>>> d={{1,3,2},4}
Traceback(most recent call last):
File "<pyshell#10>", line 1, in <module>
d={{1,3,2},4}
```

TypeError: unhashable type: ‘set’

3. The Python set() function

You can also create a set with the set() function.

```
>>> d=set()
>>>type(d)
```

<class ‘set’>

This creates an empty set object. Remember that if you declare an empty set as the following code, it is an empty dictionary, not an empty set. We confirm this using the type() function.

```
>>> d={}
>>>type(d)
```

<class ‘dict’>

The set() function may also take one argument, however. It should be an iterable, like a list.

>>> d=set([1,3,2])

b. Accessing a Set in Python

Since sets in Python do not support indexing, it is only possible to access the entire set at once. Let’s try accessing the sets from point a.

>>> a

{1, 2, 3}

Did you see how it reordered the elements into an ascending order? Now let’s try accessing the set c.

>>> c

{1, 2.0, ‘three’}

Finally, let’s access set b.

>>> b

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

As you can see, we had two 2s when we declared the set, but now we have only one, and it automatically reordered the set.

Also, since sets do not support indexing, they cannot be sliced. Let’s try slicing one.

```
>>> b[1:]
Traceback(most recent call last):
File "<pyshell#26>", line 1, in <module>
b[1:]
```

TypeError: ‘set’ object is not subscriptable

As you can see in the error, a set object is not subscriptable.

c. Deleting a Set

Again, because a set isn’t indexed, you can’t delete an element using its index. So for this, you must use one the following methods. A method must be called on a set, and it may alter the set. For the following examples, let’s take a set called numbers.

```
>>> numbers={3,2,1,4,6,5}
>>> numbers
```

{1, 2, 3, 4, 5, 6}

1. discard()

This method takes the item to delete as an argument.

```
>>> numbers.discard(3)
>>> numbers
```

{1, 2, 4, 5, 6}

As you can see in the resulting set, the item 3 has been removed.

2. remove()

Like the discard() method, remove() deletes an item from the set.

```
>>> numbers.remove(5)
>>> numbers
```

{1, 2, 4, 6}

discard() vs remove()-

These two methods may appear the same to you, but there’s actually a difference. If you try deleting an item that doesn’t exist in the set, discard() ignores it, but remove() raises a KeyError.

```
>>> numbers.discard(7)
```

2. >>> numbers

{1, 2, 4, 6}

1. >>> numbers.remove(7)
2. Traceback(most recent call last):
3. File "<pyshell#37>", line 1, in <module>
4. numbers.remove(7)

KeyError: 7

3. pop()

Like on a dictionary, you can call the pop() method on a set. However, here, it does not take an argument. Because a set doesn't support indexing, there is absolutely no way to pass an index to the pop method. Hence, it pops out an arbitrary item. Furthermore, it prints out the item that was popped.

>>> numbers.pop()

1

Let's try popping anot/her element.

>>> numbers.pop()

2

Let's try it on another set as well.

>>>{2,1,3}.pop()

1

4. clear()

Like the pop method(), the clear() method for a dictionary can be applied to a Python set as well. It empties the set in Python.

1. >>> numbers.clear()
2. >>> numbers

set()

As you can see, it denoted an empty set as set(), not as {}.

d. Updating a Set

As we discussed, a Python set is mutable. But as we have seen earlier, we can't use indices to reassign it.

1. >>> numbers={3,1,2,4,6,5}
2. >>> numbers[3]

3. Traceback(most recent call last):
4. File "<pyshell#56>", line 1, in <module>
5. numbers[3]

TypeError: ‘set’ object does not support indexing

So, we use two methods for this purpose- add() and update(). We have seen the update() method on tuples, lists, and strings.

1. add()

It takes as argument the item to be added to the set.

1. >>> numbers.add(3.5)
2. >>> numbers

{1, 2, 3, 4, 5, 6, 3.5}

If you add an existing item in the set, the set remains unaffected.

1. >>> numbers.add(4)
2. >>> numbers

{1, 2, 3, 4, 5, 6, 3.5}

2. update()

This method can add multiple items to the set at once, which it takes as arguments.

1. >>> numbers.update([7,8],{1,2,9})
2. >>> numbers

{1, 2, 3, 4, 5, 6, 3.5, 7, 8, 9}

As is visible, we could provide a list and a set as arguments to this. This is because this is different than creating a set.

e. Functions on Sets

A function is something that you can apply to a Python set, and it performs operations on it and returns a value. Let’s talk about some of the functions that a set supports. We’ll take a new set for exemplary purposes.

>>> days={'Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'}

1. len()

The len() function returns the length of a set. This is the number of elements in it.

>>>len(days)

7

2. max()

This function returns the item from the set with the highest value.

>>>max({3,1,2})

3

We can make such a comparison on strings as well.

>>>max(days)

‘Wednesday’

The Python function returned ‘Wednesday’ because W has the highest ASCII value among M, T, W, F, and S.

But we cannot compare values of different types.

1. >>>max({1,2,'three','Three'})
2. Traceback(most recent call last):
3. File "<pyshell#69>", line 1, in <module>
4. max({1,2,'three','Three'})

TypeError: ‘>’ not supported between instances of ‘str’ and ‘int’

3. min()

Like the max() function, the min() function returns the item in the Python set with the lowest value.

>>>min(days)

‘Friday’

This is because F has the lowest ASCII value among M, T, W, F, and S.

4. sum()

The sum() functionin Python set returns the arithmetic sum of all the items in a set.

>>>sum({1,2,3})

6

However, you can’t apply it to a set that contains strings.

1. >>>sum(days)
2. Traceback(most recent call last):
3. File "<pyshell#72>", line 1, in <module>
4. sum(days)

TypeError: unsupported operand type(s) for +: ‘int’ and ‘str’

5. any()

This function returns True even if one item in the set has a Boolean value of True.

```
>>>any({0})
```

False

```
>>>any({0,'0'})
```

True

It returns True because the string ‘0’ has a Boolean value of True.

6. all()

Unlike the any() function, all() returns True only if all items in the Python set have a Boolean value of True. Otherwise, it returns False.

```
>>>all({0,'0'})
```

False

```
>>>all(days)
```

True

7. sorted()

The sorted() function returns a sorted python set to list. It is sorted in ascending order, but it doesn’t modify the original set.

1. >>> numbers={1, 2, 3, 4, 5, 6, 3.5}
2. >>>sorted(numbers)

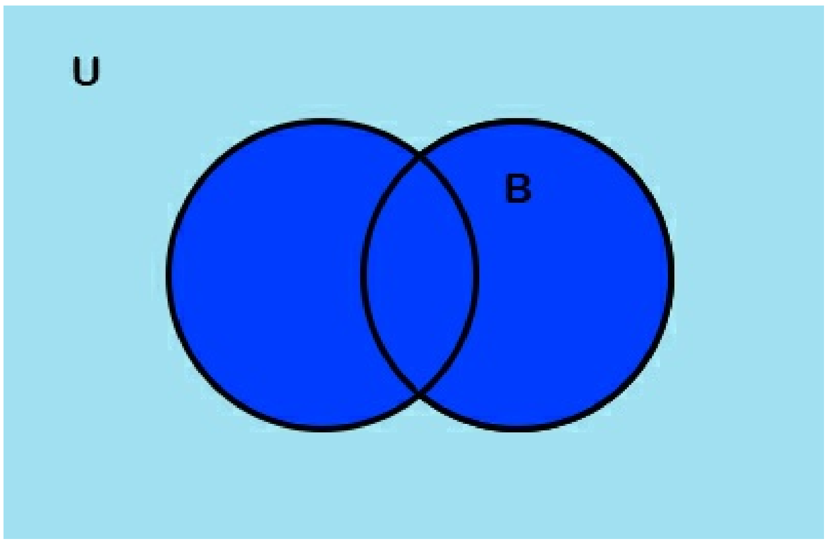
[1, 2, 3, 3.5, 4, 5, 6]

f. Methods on Sets

Unlike a function in Python set, a method may alter a set. It performs a sequence on operations on a set, and must be called on it. So far, we have learned about the methods add(), clear(), discard(), pop(), remove(), and update(). Now, we will see more methods from a more mathematical point of view.

1. union()

This method performs the union operation on two or more Python sets. What it does is it returns all the items that are in any of those sets.



Union in Python Set

```
1. >>> set1,set2,set3={1,2,3},{3,4,5},{5,6,7}
2. >>> set1.union(set2,set3)
```

```
{1, 2, 3, 4, 5, 6, 7}
```

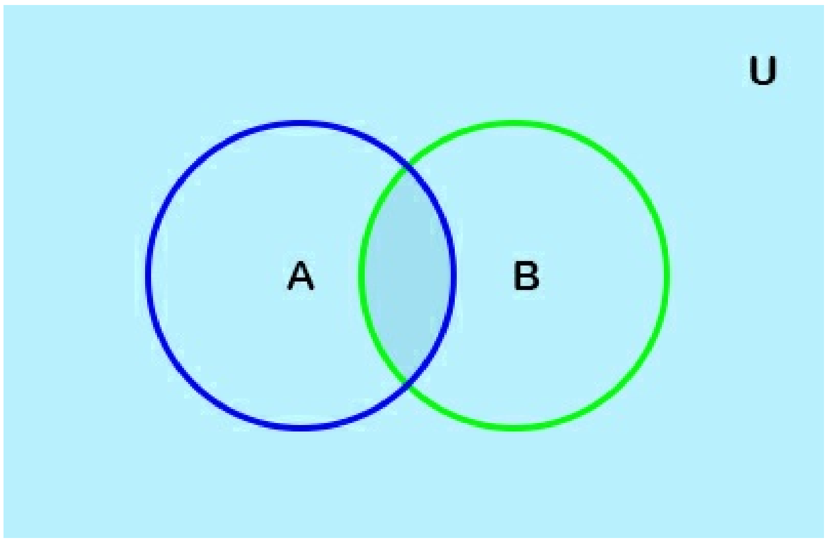
```
>>> set1
```

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

As you can see, it did not alter set1. A method does not always alter a set in Python.

2. intersection()

This method takes as argument sets, and returns the common items in all the sets.



Intersection Set in Python

```
>>> set2.intersection(set1)
```

```
{3}
```

Let’s intersect all three sets.

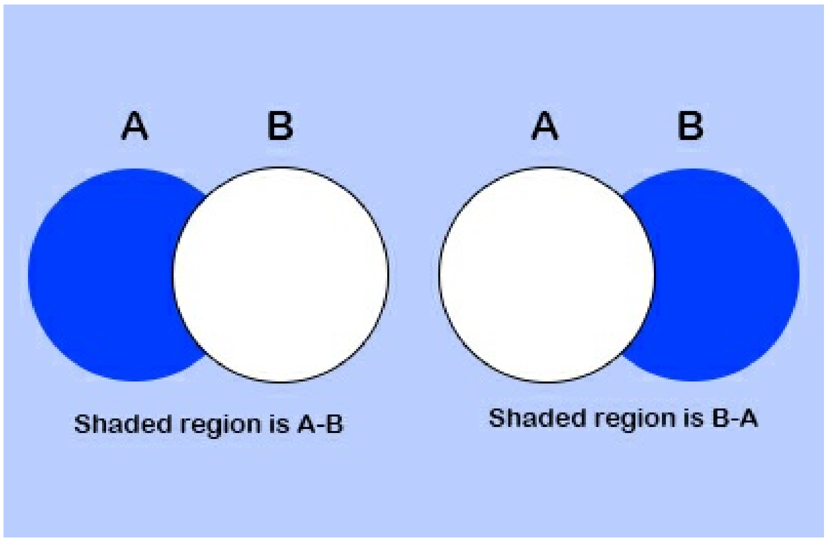
```
>>> set2.intersection(set1,set3)

set()
```

It returned an empty set because these three sets have nothing in common.

3. difference()

The difference() method returns the difference of two or more sets. It returns as a set.



Difference Python Set

```
>>> set1.difference(set2)

{1, 2}
```

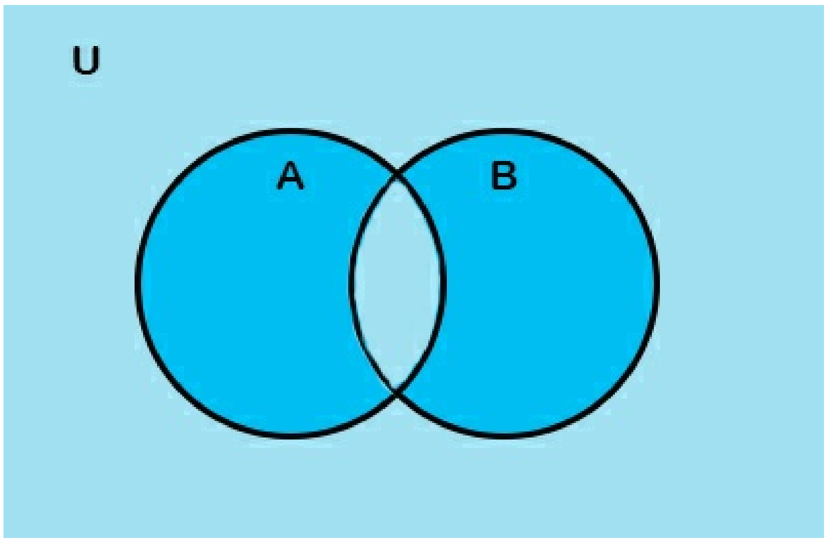
This returns the items that are in set1, but not in set2.

```
>>> set1.difference(set2,set3)

{1, 2}
```

4. symmetric_difference()

This method returns all the items that are unique to each set.



```
>>> set1.symmetric_difference(set2)
```

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

It returned 1 and 2 because they're in set1, but not in set2. It also returned 4 and 5 because they're in set2, but not in set1. It did not return 3 because it exists in both the sets.

5. intersection_update()

As we discussed in intersection(), it does not update the set on which it is called. For this, we have the intersection_update() method.

```
>>> set1.intersection_update(set2)
>>> set1
```

```
{3}
```

It stored 3 in set1, because only that was common in set1 and set2.

6. difference_update()

Like intersection-update(), this method updates the Python set with the difference.

```
>>> set1={1,2,3}
>>> set2={3,4,5}
>>> set1.difference_update(set2)
>>> set1
```

```
{1, 2}
```

7. symmetric_difference_update()

Like the two methods we discussed before this, it updates the set on which it is called with the symmetric difference.

```
>>> set1={1,2,3}
>>> set2={3,4,5}
>>> set1.symmetric_difference_update(set2)
>>> set1
```

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

8. copy()

The copy() method creates a shallow copy of the Python set.

```
>>> set4=set1.copy()
>>> set1,set4
```

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

9. isdisjoint()

This method returns True if two sets have a null intersection.

```
>>>{1,3,2}.isdisjoint({4,5,6})
```

```
True
```

However, it can take only one argument.

```
>>>{1,3,2}.isdisjoint({3,4,5},{6,7,8})
Traceback(most recent call last):
File "<pyshell#111>", line 1, in <module>
{1,3,2}.isdisjoint({3,4,5},{6,7,8})
```

```
TypeError: isdisjoint() takes exactly one argument (2 given)
```

10. issubset()

This method returns true if the set in the argument contains this set.

```
>>>{1,2}.issubset({1,2,3})
```

```
True
```

```
>>>{1,2}.issubset({1,2})
```

```
True
```

{1,2} is a proper subset of {1,2,3} and an improper subset of {1,2}.

11. issuperset()

Like the issubset() method, this one returns True if the set contains the set in the argument.

```
>>>{1,3,4}.issuperset({1,2})
```

```
False
```

```
>>>{1,3,4}.issuperset({1})
```

```
True
```

g. Operations on Sets

Now, we will look at the operations that we can perform on sets.

1. Membership

We can apply the ‘in’ and ‘not in’ python operators on items for a set. This tells us whether they belong to the set.

```
>>>'p' in {'a','p','p','l','e'}
```

```
True
```

>>>0 not in {'0','1'}

True

h. Iterating on a Set

Like we have seen with lists and tuples, we can also iterate on a set in a for-loop.

1. >>>for i in {1,3,2}:
2. print(i)

1

2

3

As you can see, even though we had the order as 1,3,2, it is printed in ascending order.

i. The frozenset

A frozen set is in-effect an immutable set. You cannot change its values. Also, a set can’t be used a key for a dictionary, but a frozenset can.

1. >>>{{1,2}:3}
2. Traceback(most recent call last):
3. File "<pyshell#123>", line 1, in <module>
4. {{1,2}:3}

TypeError: unhashable type: ‘set’

Now let’s try doing this with a frozenset.

1. >>>{frozenset(1,2):3}
2. Traceback(most recent call last):
3. File "<pyshell#124>", line 1, in <module>
4. {frozenset(1,2):3}

TypeError: frozenset expected at most 1 arguments, got 2

As you can see, it takes only one argument. Now let’s see the correct syntax.

>>>{frozenset([1,2]):3}

{frozenset({1, 2}): 3}

3. Booleans

Finally, let’s discuss Booleans. A Boolean is another data type that Python has to offer.

a. Value of a Boolean

As we have seen earlier, a Boolean value may either be True or be False. Some methods like isalpha() or issubset() return a Boolean value.

b. Declaring a Boolean

You can declare a Boolean just like you would declare an integer.

```
>>> days=True
```

As you can see here, we didn't need to delimit the True value by quotes. If you do that, it is a string, not a Boolean. Also note that what was once a set, we have reassigned a Boolean to it.

```
>>>type('True')

<class 'str'>
```

c. The bool() function

Like we have often seen earlier, the bool() function converts another value into the Boolean type.

```
>>>bool('Wisdom')

True

>>>bool([])

False
```

d. Boolean Values of Various Constructs

Different values have different equivalent Boolean values. In this example, we use the bool() Python set function to find the values.

For example, 0 has a Boolean value of False.

```
>>>bool(0)

False
```

1 has a Boolean value of True, and so does 0.00000000001.

```
>>>bool(0.000000000001)

True
```

A string has a Boolean value of True, but an empty string has False.

```
>>>bool(' ')

True

>>>bool('')
```

```
False
```

In fact, any empty construct has a Boolean value of False, and a non-empty one has True.

```
>>>bool(())

False

>>>bool((1,3,2))

True
```

e. Operations on Booleans

1. Arithmetic

You can apply some arithmetic operations to a set. It takes 0 for False, and 1 for True, and then applies the operator to them.

Addition

You can add two or more Booleans. Let’s see how that works.

```
>>>True+False #1+0

1

>>>True+True #1+1

2

>>>False+True #0+1

1

>>>False+False #0+0

0
```

Subtraction and Multiplication

The same strategy is adopted for subtraction and multiplication.

```
>>>False-True

-1
```

Division

Let’s try dividing Booleans.

```
>>>False/True

0.0
```

Remember that division results in a float.

1. >>>True/False
2. Traceback(most recent call last):
3. File "<pyshell#148>", line 1, in <module>
4. True/False

ZeroDivisionError: division by zero

This was an exception that raised. We will learn more about exception in a later lesson.

Modulus, Exponentiation, and Floor Division

The same rules apply for modulus, exponentiation, and floor division as well.

>>>False%True

0

>>>True**False

1

>>>False**False

1

>>>0//1

0

Try your own combinations like the one below.

>>>(True+True)*False+True

1

2. Relational

The relational operators we’ve learnt so far are >, <, >=, <=, !=, and ==. All of these apply to Boolean values. We will show you a few examples, you should try the rest of them.

>>>False>True

False

>>>False<=True

True

Again, this takes the value of False to be 0, and that of True to be 1.

3. Bitwise

Normally, the bitwise operators operate bit-by bit. For example, the following code ORs the bits of 2(010) and 5(101), and produces the result 7(111).

```
>>>2|5

7
```

But the bitwise operators also apply to Booleans. Let’s see how.

Bitwise &

It returns True only if both values are True.

```
>>>True&False

False

>>>True&True

True
```

Since Booleans are single-bit, it’s equivalent to applying these operations on 0 and/or 1.

Bitwise |

It returns False only if both values are False.

```
>>>False|True

True
```

Bitwise XOR (^)

This returns True only if one value is True and one is False.

```
>>>False^True

True

>>>False^False

False

>>>True^True

False
```

Binary 1’s Complement

This calculates 1’s complement for True(1) and False(0).

```
>>> ~True

-2

>>> ~False

-1
```

Left-shift(<<) and Right-shift(>>) Operators

As discussed earlier, these operators shift the value by specified number of bits left and right, respectively.

```
>>>False>>2

0

>>>True<<2

4
```

True is 1. When shifted two places two the left, it results in 100, which is binary for 4. Hence, it returns 4.

4. Identity

The identity operators ‘is’ and ‘is not’ apply to Booleans.

```
>>>False is False

True

>>>False is 0

False
```

5. Logical

Finally, even the logical operators apply on Booleans.

```
>>>FalseandTrue

False
```

This was all about the article on Python set and booleans.

4. Conclusion

In conclusion, we see that a Python Boolean value may be True or False. You may create it or use it when it’s returned by a method. We learned how to create, access, update, and delete a set. We saw how it is mutable and that is why we can’t use indexing to access, update, or delete it. So, we use certain Python set functions and methods for the same. Lastly, we learned about various operations that we can apply on a set. And we learned that some bitwise and logical operators mean the same thing on Booleans. See you tomorrow. Hope you liked our article on python sets and booleans.