Screenshot 1: Set Operations in Python

1. Creating a Set

This creates a set called setone. A **set** is an unordered collection of unique items. Here, it contains {1, 2, 3, 4}.

2. Intersection (Common Elements)

The & operator returns elements common to both sets.

Result: {1, 3}

3. Union (All Unique Elements)

```
python

Copy Copy Edit

setone | {1, 3}
```

The | operator returns all unique elements from both sets.

Result: {1, 2, 3, 4}

Union with a new element (7) included:

Result: {1, 2, 3, 4, 7}

4. Set Difference

Removes all elements present in {1, 2, 3, 4} from setone.

Result: set() (empty set)

5. Empty Set vs Empty Dictionary

In Python, {} is an **empty dictionary**, not a set.

To create an empty set, use set() instead.

Screenshot 2: Boolean Type in Python

1. Boolean Type

```
python

Copy * Edit

type(True) # <class 'bool'>
```

Python has a built-in bool type with two values: True and False.

2. Boolean and Integer Comparison

```
python

Copy Copy

True == 1 # True False == 0 # True
```

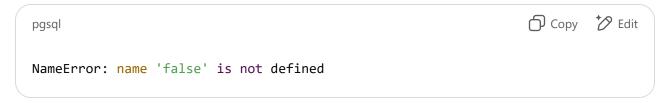
In Python:

- True is **equivalent** to 1
- False is equivalent to 0

3. Using Lowercase false

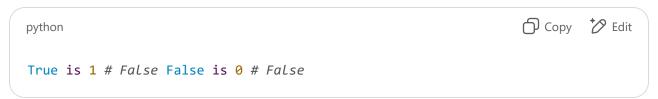


This raises an error:



Python is case-sensitive. The correct Boolean values are True and False.

4. **Using** is **vs** ==

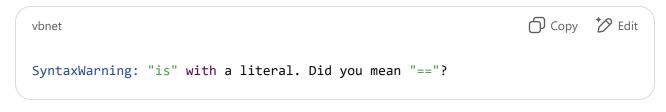


- == checks for value equality
- is checks for identity (same object in memory)

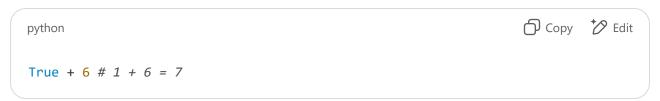
True == 1 is True (same value),

but True is 1 is False (different types/objects).

Python warns:



5. Boolean in Arithmetic



Since True is treated as 1, the result is 7.

Summary

Concept	Behavior
set()	Creates an empty set
{}	Creates an empty dictionary
True == 1	True (same value)
True is 1	False (not same object)
True + 6	7 (Boolean acts as integer)
& , `	, -`

Let me know if you want an example file or live code demonstration!

. Using is vs ==
python
Copy
Edit
True is 1 # False
False is 0 # False
== checks for value equality

is checks for identity (same object in memory)

True == 1 is True (same value), but True is 1 is False (different types/objects).

You're on the right track! Here's a clearer and deeper explanation of the difference between == and is in Python, specifically using the example with True, False, 1, and 0.

== vs is in Python

1. == (Equality Operator)

- Purpose: Checks if the values of two variables are equal.
- Result: True if the values are the same, regardless of type or memory location.

2. is (Identity Operator)

- **Purpose:** Checks if two variables point to the **same object** in memory.
- Result: True only if both variables refer to the exact same object.

Example with True, False, 1, and 0

```
python

True == 1 # True (same value) False == 0 # True (same value) True is 1 # False (not the same object) False is 0 # False (not the same object)
```

Why?

- True and 1 both behave like 1 in arithmetic, so True == 1 is True.
- But they are different types:

```
python

Copy Copy

type(True) # <class 'bool'> type(1) # <class 'int'>
```

Visualizing the Difference

Expression	Description	Result
True == 1	Values are equal	True
True is 1	Different objects in memory	False
False == 0	Values are equal	True
False is 0	Different objects in memory	False

Extra Insight

- Python optimizes memory by caching small integers and booleans.
- Even though True behaves like 1, Python stores them separately for type clarity.

Rule of Thumb

You want to compare values You want to compare identities (e.g., None, singleton objects, etc.)	Use == when	Use is when
	You want to compare values	



(a) Common Use of is

```
Copy Bedit
python
x = None if x is None: print("x is None") #  Correct if x == None: print("x
equals None") # Works, but less idiomatic
```

Let me know if you'd like a diagram showing memory identity vs value equality!



so as python treat it value as object

Yes, exactly! In **Python, **everything is treated as an object — including numbers, strings, functions, even classes themselves. This is a core concept in Python and helps explain the behavior of operators like is and == .



Breakdown: "Everything is an object"

In Python:

```
Copy Bedit
python
x = 5
```

x is a name (or reference) that points to an integer object with the value 5.

- Python stores 5 as an object of type int in memory.
- You can check:

```
python

Copy Copy

type(x) # <class 'int'> id(x) # Memory address of the object 5
```

P Difference Between Object and Value

Concept	Meaning in Python
Object	Everything in memory is an object
Value	The content inside the object (like 5)
Identity	The unique memory address of an object
Туре	What kind of object it is (int , bool , etc.)

A Example: Treating Values as Objects

python

a = 1000 b = 1000 print(a == b) # True, values are the same print(a is b) #

False, usually (separate objects in memory)

However:

Python Caches Some Objects

Python reuses small integers and strings for performance:

```
python

a = 3 b = 3 a is b # True (same cached object)

But not:

python

a = 1000 b = 1000 a is b # False (different memory)
```

Summary

- Python treats every value as an object.
- == checks if two objects have the same value.
- is checks if two variables point to the **same object** (same memory location).
- This is why True == 1 is True, but True is 1 is False.