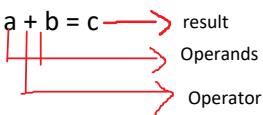


Operators

14 June 2025 12:17

Operators: They are the special symbols which is used to perform operations.

Example: $a + b = c$



Types:

- ◆ Arithmetic Operator
- ◆ Logical Operator
- ◆ Relational Operator
- ◆ Bitwise Operator
- ◆ Assignment Operator
- ◆ Membership Operator
- ◆ Identity Operator

Arithmetic Operators:-

Types of arithmetic operators:-

- ❖ 1.Addition(+)
- ❖ 2.Subtraction(-)
- ❖ 3.Multiplication(*)
- ❖ 4.Division(/)
 - True division(/)
 - Floor Division(//)
 - Modulus(%)
- ❖ 5.Power(**)

1.Addition(+):-

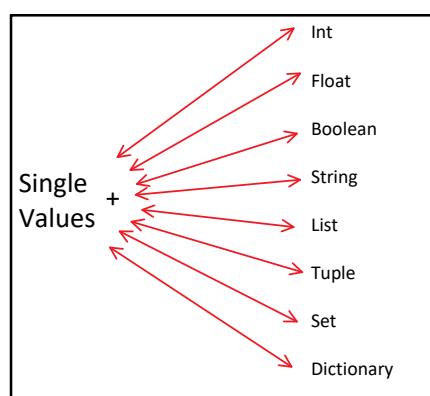
Definition: the Addition Operator is used to find the Sum of 2 or More Digits. the symbol is used for Addition '+'.
(Or)

Addition is used to perform add Multiple values to get sum of values at a time.

Syntax: Operand1 + Operand2 (or) Op1+Op2+.....+Operand n

Examples: For Single value Data Types

12+3
Output:-15
12+3.4
Output:-15.4
10+(3+4j)
Output:-(13+4j)
12+True
Output:-13
5+'hai'



Traceback (most recent call last):

File "<pyshell#5>", line 1, in <module>
5+'hai'

Type Error: unsupported operand type(s) for +: 'int' and 'str'

13+'45'

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>
13+'45'

```

TypeError: unsupported operand type(s) for +: 'int' and 'str'
5+[1]
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    5+[1]
TypeError: unsupported operand type(s) for +: 'int' and 'list'
(4+3j)+(3,)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    4+(3,)
TypeError: unsupported operand type(s) for +: 'int' and 'tuple'
5.5+{2}
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    5+{2}
TypeError: unsupported operand type(s) for +: 'int' and 'set'
True+{10:20}
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    6+{10:20}
TypeError: unsupported operand type(s) for +: 'int' and 'dict'

```

Note: We cannot add single-value data types (like int, float, complex, bool) with multi-value data types (like list, tuple, set, dictionary) using the + operator, as it will result in a Type Error.

Examples: For multi-value Data Types:-

```

'hai'+'hai'
Output:-'haihai'
[1,2,3]+[4,5,6]
Output:-[1, 2, 3, 4, 5, 6]
(10,20,30)+(40,50,60)
Output:-(10, 20, 30, 40, 50, 60)
{10,20}+{30,20}
Output:-Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    {10,20}+{30,20}
TypeError: unsupported operand type(s) for +: 'set' and 'set'
{'a':10}+{'b':30}
Output:-Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    {'a':10}+{'b':30}
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
'hi'+[1,2]
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'hi'+[1,2]
TypeError: can only concatenate str (not "list") to str

```

Collection + Collection

String + String
List + List
Tuple + Tuple
Set + Set
Dictionary + Dictionary

Note: In the case of a collection, the addition operation will not work as expected, it will merge the data instead. This process is known as "**concatenation**."

2. Subtraction (-):-

Definition: The subtraction operator (-) is used to find the difference between two or more operands.

Syntax: Operand1-Operand2

Supported Examples (with single-value data types like int, float, complex, and bool):

```
10 - 30      # Output: -20
10 - 3.5     # Output: 6.5
10 - (10 + 7j) # Output: -7j
4 - True     # Output: 3
```

Unsupported Examples (with incompatible data types):

```
10 - 'hai'    # Error: int - str not allowed
10 - '3'      # Error: int - str not allowed
2 - [1]       # Error: int - list not allowed
3 - (5,)      # Error: int - tuple not allowed
5 - {1, 2}    # Error: int - set not allowed
6 - {1: 2}    # Error: int - dict not allowed
```

Why do we get these errors?

Subtraction is only supported between compatible data types (like integers, floats, complex numbers, and Booleans).

You **cannot** subtract:

- Strings (str)
- Lists
- Tuples
- Dictionaries
- Sets (when used with a non-set)

These types represent collections or non-numeric data, and direct subtraction doesn't make sense for them.

Special Case: Subtraction in Set Data Type:

Python **does allow subtraction for sets**. This operation returns a new set containing elements that are in the first set but **not in the second**.

Example:

```
{1, 2, 3, 4, 5, 6} - {8, 9, 10, 20, 1, 2, 3, 4}
# Output: {5, 6}
{7, 3, 5, True, 98} - {10, 7, 99, 1, 18, 72} - {3, 95, 100}
# Output: {98, 5}
```

Note: In set subtraction, the result will include all values from the first set that are **not present** in the second set.

We **cannot** directly remove elements from a list, tuple, or dictionary using subtraction. For such collections, we use methods like `.pop()`, `.remove()`, or slicing. However, **sets in Python support mathematical operations**, including subtraction (`-`).

That's why subtraction works perfectly between sets, but not for other collections.

3.Multiplication(*):-

The multiplication operator `*` is used to calculate the product of two or more operands. It behaves differently based on the data types involved.

Syntax: Operand1 * Operand2

Supported Data Type Combinations:

1. Numeric Values (int, float, complex, bool):

You can multiply numbers directly.

```
8 * 2      → Output:- 16
```

$2.3 * 4 \rightarrow$ Output:- 9.2
 $5 * (10+5j) \rightarrow$ Output:- (50+25j)
 $(1+4j)*(2+6j) \rightarrow$ Output:- (-22+14j)
 $\text{True} * 3 \rightarrow$ Output:- 3 # True is treated as 1

2. String Repetition:

A string can be multiplied by an integer to repeat it.
 $'ji' * 6 \rightarrow$ 'jiijiijiiji'

3. List/Tuple Repetition:

Lists and tuples can be multiplied by an integer to repeat their elements.

$'abc' * 2 \rightarrow$ 'abcabc'
 $[1, 2] * 3 \rightarrow$ [1, 2, 1, 2, 1, 2]
 $(7,) * 1 \rightarrow$ (7,)

Unsupported Combinations (TypeErrors):

When you try to multiply unsupported data types, Python throws a `TypeError`.

- ✧ **String × String:**
`'abc' * 'abc'`
Error: can't multiply sequence by non-int of type 'str'
- ✧ **List × List:**
`[1, 2, 3] * [1, 2, 3]`
Error: can't multiply sequence by non-int of type 'list'
- ✧ **Tuple × Tuple:**
`(1, 2) * (1, 2)`
Error: can't multiply sequence by non-int of type 'tuple'
- ✧ **Set × Set or Set × Integer:**
`{1, 2} * {1, 2}`
`{1, 2} * 2`
Error: unsupported operand type(s) for *: 'set' and ...
- ✧ **Dictionary × Dictionary or Dictionary × Integer:**
`{1: 2} * {1: 2}`
`{1: 2} * 2`
Error: unsupported operand type(s) for *: 'dict' and ...

Note:-

The `*` operator in Python is used to perform multiplication. It works correctly with numbers (integers, floats, complex, and booleans), and it can also be used to repeat sequences like strings, lists, and tuples when multiplied by an integer. However, using `*` between incompatible types like `stringstring`, `listlist`, `tupletuple`, `setset`, or `dict*dict` will result in a `TypeError`.

4. Division(/):-

In Python, the division operator is classified into **three types**:

1. **True Division (/)** – This returns the **exact quotient** in float format, even if both operands are integers.
2. **Floor Division (//)** – This returns the **quotient after removing the decimal part** (i.e., the floor value).
3. **Modulus (%)** – This returns the **remainder** after division.

Examples of True Division (/):

Syntax: Operand1/Operand2

$10 / 2 \rightarrow$ 5.0 # Dividing two integers gives a float

```
7.6 / 3      → 2.533333 # Float divided by int gives precise result  
7 / (10 + 4j) → (0.6034 - 0.2413j) # Division also works with complex numbers  
5 / True     → 5.0      # True is treated as 1
```

Examples of Floor Division (//):

Syntax: Operand1//Operand2

```
10.5 // 3  
# Output: 3.0  
7.9 // 2.5  
# Output: 3.0  
-5.2 // 2  
# Output: -3.0  
5 // True  
# Output: 5
```

Returns a **float result**, but still the floor value.

Examples of Modulus (%):

Syntax: Operand1%Operand2

```
10 % 3  
# Output: 1  
Note:-The modulus operator returns the remainder after division.  
10.5 % 2  
# Output: 0.5  
Note: Works with float values, and returns floating-point remainders  
(4 + 5j) % 2  
# Error: TypeError: can't mod complex numbers.  
Note: Modulus does not work with complex numbers.  
5 % True  
# Output: 0  
Note: True is treated as 1.
```

Important Notes:

- **True Division (/), Floor Division (//), and Modulus (%) operators work on integers and floats.**
- Only the **True Division operator (/) works with complex numbers.**
- **Division operators do not work on collection data types** like list, tuple, set, or dictionary.

True Division (/), Floor Division (//), and Modulus (%) work on integers and floats:

Examples:

- ✧ True Division
10 / 3 # Output: 3.3333333333333335
- ✧ Floor Division
10 // 3 # Output: 3
- ✧ Modulus
10 % 3 # Output: 1
- ✧ With float values
7.5 / 2 # Output: 3.75
7.5 // 2 # Output: 3.0
7.5 % 2 # Output: 1.5

True Division (/) works with complex numbers:

Examples:

- ❖ Division of real number by complex number

$$5 / (2 + 3j) \quad \text{# Output: } (0.6153846153846154 - 0.9230769230769231j)$$

Note: // and % **do NOT work** with complex numbers. They will raise a TypeError

Division operators do NOT work on collection data types:

Examples:

- ❖ List division

$$[1, 2, 3] / 2$$

Error: TypeError: unsupported operand type(s) for /: 'list' and 'int'
- ❖ Tuple division

$$(1, 2, 3) // 2$$

Error: TypeError: unsupported operand type(s) for //: 'tuple' and 'int'
- ❖ Set modulus

$$\{1, 2, 3\} \% 2$$

Error: TypeError: unsupported operand type(s) for %: 'set' and 'int'
- ❖ Dictionary division

$$\{1: 'a'\} / 2$$

Error: TypeError: unsupported operand type(s) for /: 'dict' and 'int'

5.Power():-**

In Python, the ** operator is used to perform **exponentiation** (i.e., raising a number to the power of another number).

Syntax: base ** exponent Or Operand**n

> Integer to the power of Integer

```
3 ** 4
# Output: 81
Note: This means 3 raised to the power of 4 (i.e.,  $3 \times 3 \times 3 \times 3$ ).
```

> Float to the power of Integer

```
4.5 ** 1
# Output: 4.5
Note: A float can be raised to any integer power.
```

> Power with Complex Number

```
5 ** (10 + 6j)
# Output: (-9504326.976087175-2243925.215386333j)
Note: Python supports complex powers using built-in complex number handling.
```

> Boolean to the power of Integer

```
True ** 2
# Output: 1
Note: Python treats True as 1, so this becomes  $1^{** 2} = 1$ .
```

Invalid Examples (will raise Errors):-

❖ String to the power of Integer

```
'hi' ** 2
# TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

❖ List to the power of Integer

```
[1, 2] ** 2
# TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
✧ Tuple to the power of Integer

(1, 2) ** 2
# TypeError: unsupported operand type(s) for ** or pow(): 'tuple' and 'int'
✧ Set to the power of Integer

{1, 2} ** 2
# TypeError: unsupported operand type(s) for ** or pow(): 'set' and 'int'
✧ Dictionary to the power of Integer

{1: 2} ** 2
# TypeError: unsupported operand type(s) for ** or pow(): 'dict' and 'int'
```

In Python, the power operator `**` is used to perform exponentiation and works with integers, floats, booleans, and complex numbers, but **does not support** collection data types like strings, lists, tuples, sets, or dictionaries — attempting to use them will raise a **TypeError**.