Python Operators

- Operators are special symbols in Python that carry out arithmetic or logical computation.
- Understanding each kind of operator is essential for creating efficient code and solving complex problems.

Python divides the operators in the following groups:

- Arithmetic operators
- Comparison operators
- Logical operators
- Assignment operators
- Identity operators
- Membership operators

1. Arithmetic Operators

These operators are used for mathematical operations.

Operator	Description	Example (`a=10` , `b=3`)	Result
`+`	Addition	`a + b`	`13`
`-`	Subtraction	`a - b`	`7`
`*`	Multiplication	`a * b`	`30`
`/`	Division	`a / b`	`3.3333`
`//`	Floor Division	`a // b`	`3`
`%`	Modulus (Remainder)	`a % b`	`1`
`**`	Exponentiation	`a ** b`	`1000`

In [1]:
```
a = 10
b = 3

print(a + b)  # Addition: 10 + 3 = 13
print(a - b)  # Subtraction: 10 - 3 = 7
print(a * b)  # Multiplication: 10 * 3 = 30
print(a / b)  # Division: 10 / 3 = 3.3333
print(a // b) # Floor Division: 10 // 3 = 3
print(a % b)  # Modulus: 10 % 3 = 1
print(a ** b) # Exponentiation: 10^3 = 1000
```

```
13
7
30
3.3333333333333335
3
1
1000
```

2. Comparison (Relational) Operators

These operators compare two values and return a Boolean result (`True` or `False`).

Operator	Description	Example (`a=10` , `b=3`)	Result
`==`	Equal to	`a == b`	`False`
`!=`	Not equal to	`a != b`	`True`
`>`	Greater than	`a > b`	`True`
`<`	Less than	`a < b`	`False`
`>=`	Greater than or equal to	`a >= b`	`True`
`<=`	Less than or equal to	`a <= b`	`False`

In [2]:
```
a = 10
b = 3

print(a == b)  # False (10 is not equal to 3)
print(a != b)  # True (10 is not equal to 3)
print(a > b)   # True (10 is greater than 3)
print(a < b)   # False (10 is not less than 3)
print(a >= b)  # True (10 is greater than or equal to 3)
print(a <= b)  # False (10 is not less than or equal to 3)
```

```
False
True
True
False
True
False
```

3. Logical Operators

These operators are used to combine conditional statements.

Operator	Description	Example (`a=10` , `b=3`)	Result
`and`	Returns `True` if both are `True`	`(a > 5 and b < 5)`	`True`
`or`	Returns `True` if at least one is `True`	`(a < 5 or b < 5)`	`True`
`not`	Reverses the Boolean value	`not(a > 5)`	`False`

In [3]:
```
x = True
y = False
```

```
print(x and y) # False (Both must be True)
print(x or y)  # True (At least one must be True)
print(not x)   # False (Reverses True to False)
```

```
False
True
False
```

4. Assignment Operators

These operators are used to assign values to variables.

Operator	Example (`a=10`)	Equivalent To	Result		
`=`	`a = 10`	`a = 10`	`10`		
`+=`	`a += 5`	`a = a + 5`	`15`		
`-=`	`a -= 5`	`a = a - 5`	`5`		
`*=`	`a *= 2`	`a = a * 2`	`20`		
`/=`	`a /= 2`	`a = a / 2`	`5.0`		
`//=`	`a //= 3`	`a = a // 3`	`3`		
`%=`	`a %= 3`	`a = a % 3`	`1`		
`**=`	`a **= 2`	`a = a ** 2`	`100`		
`&=`	`a &= 3`	`a = a & 3`	`2`		
`	=`	`a	= 3`	`a = a 3`	`11`
`^=`	`a ^= 3`	`a = a ^ 3`	`9`		
`<<=`	`a <<= 2`	`a = a << 2`	`40`		
`>>=`	`a >>= 2`	`a = a >> 2`	`2`		

In [4]:
```
a = 10
b = 5

a += b  # a = 10 + 5 = 15
print(a)  # 15

a -= b  # a = 15 - 5 = 10
print(a)  # 10

a *= b  # a = 10 * 5 = 50
print(a)  # 50

a /= b  # a = 50 / 5 = 10.0
print(a)  # 10.0

a = int(a)  # Convert back to integer for bitwise operations

a //= b  # a = 10 // 5 = 2
print(a)  # 2
```

```
a **= b  # a = 2 ** 5 = 32
print(a)  # 32
```

```
15
10
50
10.0
2
32
```

5. Identity Operators

These operators check whether two objects refer to the same memory location.

Operator	Description	Example (`a = 10` , `b = 10` , `c = [1, 2, 3]` , `d = [1, 2, 3]`)	Result
`is`	Returns `True` if both refer to the same object	`a is b`	`True`
`is not`	Returns `True` if they refer to different objects	`c is d`	`False`

In [5]:
```
x = [1, 2, 3]
y = [1, 2, 3]
z = x  # z is assigned the same reference as x

print(x is y)   # False (Different objects)
print(x is z)   # True (Same object reference)
print(x is not y)  # True (Not the same object)
```

```
False
True
True
```

6. Membership Operators

These operators check if a value is present in a sequence (string, list, tuple, dictionary, etc.).

Operator	Description	Example (`x = [1, 2, 3, 4, 5]`)	Result
`in`	Returns `True` if the value exists	`2 in x`	`True`
`not in`	Returns `True` if the value does not exist	`10 not in x`	`True`

In [6]:
```
my_list = [1, 2, 3, 4, 5]

print(3 in my_list)    # True (3 exists in the list)
print(10 not in my_list)  # True (10 is not in the list)
```

```
True
True
```

7. Operator Precedence

Operator precedence determines the order in which expressions are evaluated. Higher precedence operators are evaluated first.

Precedence Order (Highest to Lowest)

1. `()` - Parentheses
2. `**` - Exponentiation
3. `+x, -x, ~x` - Unary plus, minus, bitwise NOT
4. `*, /, //, %` - Multiplication, division, floor division, modulus
5. `+, -` - Addition, subtraction
6. `<<, >>` - Bitwise shift
7. `&` - Bitwise AND
8. `^` - Bitwise XOR
9. `|` - Bitwise OR
10. `==, !=, >, <, >=, <=` - Comparison
11. `is, is not, in, not in` - Identity and membership operators
12. `not` - Logical NOT
13. `and` - Logical AND
14. `or` - Logical OR
15. `=` and assignment operators (`+=` , `-=` , etc.)

In [7]:
```python
x = 5 + 3 * 2  # Multiplication first, then addition
print(x)  # Output: 11

y = (5 + 3) * 2  # Parentheses first, then multiplication
print(y)  # Output: 16
```

```
11
16
```

Python `input()` and String Formatting (`format()` & f-strings)

This covers:

- How to use `input()` for user input
- `format()` method for string formatting
- Using indexed, named, and formatted placeholders
- Modern `f-strings` for clean formatting

1. `input()` Function

The `input()` function is used to take user input as a string. By default, it returns a string value.

In [8]:
```
# Example: Basic input()
name = input("Enter your name: ")
print("Hello,", name)
```

```
Hello, ss
```

Taking Integer Input

Since `input()` returns a string, we need to convert it using `int()` if we expect an integer.

In [9]:
```
# Example: Taking integer input
age = int(input("Enter your age: "))
print("You are", age, "years old.")
```

```
You are 25 years old.
```

Taking Multiple Inputs

You can take multiple space-separated inputs using `split()`. If numeric input is required, use `map()` to convert them.

In [11]:
```
# Example: Taking multiple inputs
x, y = map(int, input("Enter two numbers: ").split())
print("Sum:", x + y)
```

```
Sum: 3
```

2. `format()` Method

The `format()` method is used to format strings dynamically by inserting values into placeholders `{}`.

In [12]:
```
# Example: Basic format() usage
name = "Alice"
age = 25
print("My name is {} and I am {} years old.".format(name, age))
```

```
My name is Alice and I am 25 years old.
```

Using Indexed Placeholders

You can use index numbers `{0}, {1}` to control the order of values.

In [13]:
```
# Example: Indexed placeholders
print("I love {1} more than {0}!".format("Tea", "Coffee"))
```

```
I love Coffee more than Tea!
```

Using Named Placeholders

Instead of positional arguments, you can use named arguments for better readability.

In [14]:
```python
# Example: Named placeholders
print("Name: {name}, Age: {age}".format(name="Alice", age=25))
```

```
Name: Alice, Age: 25
```

Formatting Numbers

- You can format decimal places using `{:.2f}`.
- Use `{:,}` to format large numbers with commas.

In [15]:
```python
# Example: Formatting numbers
pi = 3.1415926535
print("Value of pi: {:.2f}".format(pi))  # Rounds to 2 decimal places

number = 1000000
print("Formatted Number: {:,}".format(number))
```

```
Value of pi: 3.14
Formatted Number: 1,000,000
```

3. Modern `f-strings` (Python 3.6+)

An easier and more readable way to format strings is by using **f-strings**.

In [17]:
```python
# Example: Using f-strings
name = "Alice"
age = 25
print(f"My name is {name} and I am {age} years old.")
```

```
My name is Alice and I am 25 years old.
```

Formatting Numbers with f-strings

You can also format numbers using f-strings similar to `format()`.

In [18]:
```python
# Example: f-strings with formatting
pi = 3.14159
print(f"Value of pi: {pi:.2f}")
```

```
Value of pi: 3.14
```