What are the Operators in the Python

In Python, operators are special symbols or keywords that perform operations on values or variables. Operators are used to manipulate data and variables, enabling you to perform calculations, comparisons, and more. Python supports several types of operators, each serving different purposes.

Types of operators

- Arthimatic operators
- Assignment operators
- Logical operators
- Comparison operators
- Identity operators
- Membership operators
- Conditional operators
- Bitwise operators

Arthimatic operators

- Perform basic mathematical operations.

+(Addition): Adds two values.

-(Subtraction): Subtracts the second value from the first

*(Multiplication): Multiplies two values.

/ (Division): Divides the first value by the second (returns a float).

// (Floor Division): Divides and returns the largest integer less than or equal to the result.

% (Modulus): Returns the remainder of the division.

** (Exponentiation): Raises the first value to the power of the second.

Examples Arthimatic operators

```
a = 2
b = 5
result = a + b
print(result)
```

```
7
```

```
a = 2
b = 5
result = a - b
print(result)
```

```
-3
```

```
a = 2
b = 5
result = a * b
print(result)
```

```
10
```

```
a = 2
b = 5
result = a / b
print(result)
```

```
0.4
```

```
a = 2
b = 5
result = a // b
print(result)
```

```
0
```

```
a = 2
b = 5
result = a % b
print(result)
```

```
2
```

```
a = 2
b = 5
result = a ** b
print(result)
```

```
32
```

```
a = 2
result = +a
print(result)
```

```
2
```

```
a = 2
result = -a
print(result)
```

```
-2
```

```
a = 2
b = 5
result = (a + b) * (b - a) / a
print(result)
```

```
10.5
```

Assignment operators:

Assignment Operators: Used to assign values to variables and perform operations simultaneously.

=: Assigns a value to a variable.

+=: Adds and assigns.

-=: Subtracts and assigns.

=: Multiplies and assigns.

/=: Divides and assigns.

//=: Performs floor division and assigns.

%=: Performs modulus and assigns.

**=: Performs exponentiation and assigns # Example Assignment operators

```
x = 10
print(x)
```

```
10
```

```
x = 10
x += 5  # Equivalent to x = x + 5
print(x)
```

```
15
```

```python
x = 10
x -= 3  # Equivalent to x = x - 3
print(x)
```

```
7
```

```python
x = 4
x *= 6  # Equivalent to x = x * 6
print(x)
```

```
24
```

```python
x = 20
x /= 4  # Equivalent to x = x / 4
print(x)
```

```
5.0
```

```python
x = 20
x //= 3  # Equivalent to x = x // 3
print(x)
```

```
6
```

```python
x = 10
x %= 4  # Equivalent to x = x % 4
print(x)
```

```
2
```

```python
x = 2
x **= 3  # Equivalent to x = x ** 3
print(x)
```

```
8
```

```python
x = 12  # Binary: 1100
x &= 7   # Binary: 0111
print(x)
```

```
4
```

```python
x = 6  # Binary: 0110
x |= 3  # Binary: 0011
print(x)
```

```
7
```

Comparison Operators:

Compare two values and return a boolean result (True or False).

== (Equal to): Checks if two values are equal.

!= (Not equal to): Checks if two values are not equal.

(Greater than): Checks if the first value is greater than the second.

< (Less than): Checks if the first value is less than the second.

= (Greater than or equal to): Checks if the first value is greater than or equal to the second.

<= (Less than or equal to): Checks if the first value is less than or equal to the second. # Examples for Comparison Operators

```
a = 5
b = 5
print(a == b)
```

```
True
```

```
a = 5
b = 3
print(a != b)
```

```
True
```

```
a = 7
b = 3
print(a > b)
```

```
True
```

```
a = 4
b = 10
print(a < b)
```

```
True
```

```
a = 5
b = 5
print(a >= b)
```

```
True
```

```
a = 8
b = 10
print(a <= b)
```

```
True
```

```
str1 = "hello"
str2 = "world"
print(str1 == str2)
```

```
False
```

```
list1 = [1, 2, 3]
list2 = [1, 2, 3]
list3 = [4, 5, 6]
print(list1 != list2)
print(list1 != list3)
```

```
False
True
```

```
x = [1, 2, 3]
y = [1, 2, 3]
print(x is y)
```

```
False
```

```
a = None
b = None
print(a is not b)
```

```
False
```

Logical Operators

These operators are used to combine conditional statements

- (and) : Returns True if both statements are true (e.g., a and b)
- (or) : Returns True if at least one statement is true (e.g., a or b)
- (not) : Reverses the result, returns False if the statement is true (e.g., not a)

Example for and,or,not

```
a = 5
b = 10
c = 15
result = (a < b) and (b < c)
print(result)
```

```
True
```

```
a = 5
b = 20
result = (a > 10) or (b > 10)
print(result)
```

```
True
```

```
a = True
result = not a
print(result)
```

```
False
```

```
x = 7
y = 3
z = 12
result = (x > y) and (y < z) or (z > 20)
print(result)
```

```
True
```

```
x = 5
y = 10
result = not (x > 10 and y < 20)
print(result)
```

```
True
```

```
a = False
b = True
result = not (a or b)
print(result)
```

```
False
```

```
a = 10
b = 20
c = 30
result = (a < b < c) and (c > b)
print(result)
```

```
True
```

```
s1 = "hello"
s2 = "world"
result = (s1 == "hello") and (s2 == "world")
print(result)
```

```
True
```

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
result = (1 in list1) or (7 in list2)
print(result)
```

```
True
```

```
age = 25
has_license = True
result = (age >= 18) and (has_license)
if result:
    print("You can drive.")
else:
    print("You cannot drive.")
```

```
You can drive.
```

Identity Operators

Identity operators in Python are used to compare the memory locations of two objects to check if they refer to the same object in memory. There are two identity operators:

- is: Evaluates to True if both variables point to the same object (i.e., have the same memory location).
- is not: Evaluates to True if the variables point to different objects (i.e., have different memory locations)

```
a = 1000
b = 1000
print(a is b)
```

```
False
```

```
a = 1000
b = 1000
print(a is not b)
```

```
True
```

```
a = "hello"
b = "hello"
print(a is b)
```

```
True
```

```
a = "hello"
b = "hello"
print(a is not b)
```

```
False
```

```
list1 = [1, 2, 3]
list2 = [1, 2, 3]
print(list1 is list2)
```

```
False
```

```
list1 = [1, 2, 3]
list2 = [1, 2, 3]
print(list1 is not list2)
```

```
True
```

```
tuple1 = (1, 2, 3)
tuple2 = (1, 2, 3)
print(tuple1 is tuple2)
```

```
False
```

```
tuple1 = (1, 2, 3)
tuple2 = (1, 2, 3)
print(tuple1 is not tuple2)
```

```
True
```

```
a = None
b = None
print(a is b)
```

```
True
```

```
a = None
b = None
print(a is not b)
```

```
False
```

Membership Operator

Membership operators in Python are used to test whether a value is found within a sequence, such as a string, list, tuple, or set. There are two membership operators:

- in: Evaluates to True if the specified value is present in the sequence.
- not in: Evaluates to True if the specified value is not present in the sequence.

```
fruits = ['apple', 'banana', 'cherry']
print('apple' in fruits)
```

```
True
```

```
fruits = ['apple', 'banana', 'cherry']
print('orange' not in fruits)
```

```
True
```

```
sentence = "The quick brown fox"
print('quick' in sentence)
```

```
True
```

```
sentence = "The quick brown fox"
print('slow' not in sentence)
```

```
True
```

```
numbers = (1, 2, 3, 4, 5)
print(3 in numbers)
```

```
True
```

```
numbers = (1, 2, 3, 4, 5)
print(6 not in numbers)
```

```
True
```

```
student = {'name': 'John', 'age': 25}
print('name' in student)
```

```
True
```

```
student = {'name': 'John', 'age': 25}
print('grade' not in student)
```

```
True
```

```
unique_numbers = {1, 2, 3, 4, 5}
print(2 in unique_numbers)
```

```
True
```

```
unique_numbers = {1, 2, 3, 4, 5}
print(7 not in unique_numbers)
```

```
True
```

Bitwise operators

Bitwise operators in Python operate at the bit level, meaning they perform operations on the binary representations of integers. These operators are used to manipulate bits and are commonly used in low-level programming tasks.

Here are the bitwise operators:

- AND (&): Compares each bit, returns 1 if both bits are 1.
- OR (|): Compares each bit, returns 1 if any bit is 1.

- XOR (^): Compares each bit, returns 1 if bits are different.
- NOT (~): Inverts all bits.
- Left Shift (<<): Shifts bits to the left.
- Right Shift (>>): Shifts bits to the right.

```
a = 12  # Binary: 1100
b = 7   # Binary: 0111
result = a & b
print(result)
```

```
4
```

```
a = 12  # Binary: 1100
b = 7   # Binary: 0111
result = a | b
print(result)
```

```
15
```

```
a = 12  # Binary: 1100
b = 7   # Binary: 0111
result = a ^ b
print(result)
```

```
11
```

```
a = 5   # Binary: 0101
result = a << 2
print(result)
```

```
20
```

```
a = 12  # Binary: 0000 1100
result = ~a
print(result)
```

```
-13
```

```
a = 20  # Binary: 10100
result = a >> 2
print(result)
```

```
5
```