Python

Operators in Python

Operators are the symbols that are used to perform operations on variables and values.

The types of operators in python are as follows:

1. Arithmetic Operators
2. Relational Operators / Comparison Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Equality Operators
7. Shift Operators
8. Ternary Operator / Conditional Operator
9. Special Operators

- Identity Operator
- Membership Operator

Now, Let's study all Operators one by one:

1. **Arithmetic Operators:**

Arithmetic operators are used with the numeric values to perform mathematical operations, however '+' and * can be used in string also under correct conditions.

Operator	**Name**	**Example**
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Examples in Code:

```python
# Arithmetic operators in Python

# Addition (+)
a = 10
b = 5
addition = a + b
print("Addition:", a, "+", b, "=", addition)

# Subtraction (-)
subtraction = a - b
print("Subtraction:", a, "-", b, "=", subtraction)

# Multiplication (*)
multiplication = a * b
print("Multiplication:", a, "*", b, "=", multiplication)

# Division (/)
division = a / b
print("Division:", a, "/", b, "=", division)

# Floor Division (//)
floor_division = a // b
print("Floor Division:", a, "//", b, "=", floor_division)

# Modulus (%)
modulus = a % b
print("Modulus:", a, "%", b, "=", modulus)

# Exponentiation (**)
exponentiation = a ** b
print("Exponentiation:", a, "**", b, "=", exponentiation)
```

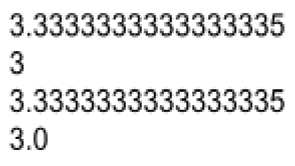
Output:

```
Addition: 10 + 5 = 15
Subtraction: 10 - 5 = 5
Multiplication: 10 * 5 = 50
Division: 10 / 5 = 2.0
Floor Division: 10 // 5 = 2
```

```
Modulus: 10 % 5 = 0
Exponentiation: 10 ** 5 = 100000
```

Different Cases of Division and Floor Division

```
print(10/3)
print(10//3)
print(10.0/3)
print(10.0//3)
```



```
3.3333333333333335
3
3.3333333333333335
3.0
```

2. **Relational Operators:**

The relational operators or comparison operator can be used with various data types, including numbers, strings, booleans, and more. In Python, comparison operators are used to compare the values of two operands (elements being compared). When comparing strings, the comparison is based on the alphabetical order of their characters (lexicographic order).

The available relational operators are: `<, >, >=` **`and`** `<=`

Examples:

```
# Integers comparison
a = 99
b = 58

print("Integer comparison:")
print(a > b)    # True
print(a < b)    # False
print(a >= b)   # True
print(a <= b)   # False

# Float comparison
x = 10.5
y = 20.3

print("\nFloat comparison:")
print(x > y)    # False
print(x < y)    # True
print(x >= y)   # False
print(x <= y)   # True
```

```
# String comparison
str1 = "apple"
str2 = "banana"

print("\nString comparison:")
print(str1 > str2)   # False (because 'a' comes before 'b'
lexicographically)
print(str1 < str2)   # True
print(str1 >= str2)  # False
print(str1 <= str2)  # True

# Boolean comparison
bool1 = True
bool2 = False

print("\nBoolean comparison:")
print(bool1 > bool2)   # True (True is considered as 1 and False as 0)
print(bool1 < bool2)   # False
print(bool1 >= bool2)  # True
print(bool1 <= bool2)  # False
```

Chaining of relational operator:

Chaining is left to right, so in a < b < c, the expression a < b is evaluated before b < c, and if a < b is falsey, b < c is not evaluated.

(2 < 1 < f()) gives the value False without calling the function f, because 2 < 1 evaluates to false, so the second comparison does not need to be performed.

Examples:

```
print(10>20>30)
print(10<20<30<40)
print(10<20<30<40>40)
print(10>20<30<40>40)
```

Output:

```
False
True
False
False
```

3. Equality Operator:

The equality operators in the python are: == and !=, they are used to compare whether two variables or values are equal or not. The result of the evaluation is either True or False.

Examples:
(Outputs are mentioned as comment)

```
# Integer comparison
a = 10
b = 50
print("Integer comparison:")
print(a == b)  # False
print(a != b)  # True

x = 2
y = 2
print(a == b)  # True
print(a != b)  # False

# Float comparison
x = 10.5
y = 20.5
print("\nFloat comparison:")
print(x == y)  # False
print(x != y)  # True

# String comparison
str1 = "hello"
str2 = "world"
print("\nString comparison:")
print(str1 == str2)  # False
print(str1 != str2)  # True

# Boolean comparison
bool1 = True
bool2 = False
print("\nBoolean comparison:")
print(bool1 == bool2)  # False
print(bool1 != bool2)  # True

# Comparison between boolean and integer
bool_int1 = True  # True is treated as 1
bool_int2 = False  # False is treated as 0
print("\nBoolean and Integer comparison:")
print(bool_int1 == 1)  # True (True is equivalent to 1)
print(bool_int2 == 0)  # True (False is equivalent to 0)
print(bool_int1 != 0)  # True (True is not 0)
print(bool_int2 != 1)  # True (False is not 1)
```

4. **Logical Operators**

   Python logical operators are used to form compound Boolean expressions. Each operand for these logical operators is itself a Boolean expression. There are three logical operators in python, they are: and, or, and not.

Example:

```
age > 16 and marks > 80
percentage < 50 or attendance < 75
```

Logical "and" operator:

For the compound Boolean expression to be True, both the operands must be True. If any or both operands evaluate to False, the expression returns False.

Logical or Operator

In contrast, the or operator returns True if any of the operands is True. For the compound Boolean expression to be False, both the operands have to be False.

not operator

This is a unary operator. The state of Boolean operand that follows, is reversed. As a result, not True becomes False and not False becomes True.

For non boolean types:

For and:

The expression "x and y" first evaluates "x". If "x" is false, its value is returned; otherwise, "y" is evaluated and the resulting value is returned.

For or:

The expression "x or y" first evaluates "x"; if "x" is true, its value is returned; otherwise, "y" is evaluated and the resulting value is returned.

Examples:

```
x = 10
y = 20
print("x > 0 and x < 10:",x > 0 and x < 10)
print("x > 0 and y > 10:",x > 0 and y > 10)
print("x > 10 or y > 10:",x > 10 or y > 10)
print("x%2 == 0 and y%2 == 0:",x%2 == 0 and y%2 == 0)
print ("not (x+y>15):", not (x+y)>15)
```

Output:

```
x > 0 and x < 10: False
x > 0 and y > 10: True
x > 10 or y > 10: True
x%2 == 0 and y%2 == 0: True
not (x+y>15): False
```

5. Bitwise Operator and Shift Operators

Computers store all kinds of information as a stream of binary digits called bits. Whether you're working with text, images, or videos, they all boil down to ones and zeros. Python's bitwise operators let you manipulate those individual bits of data at the most granular level.

Here is a summary of the common bitwise operators in Python:

AND (&): Compares each bit of two integers and returns 1 if both bits are 1, otherwise returns 0.

Example: 5 & 3 results in 1 (binary: 101 & 011 = 001).

OR (|): Compares each bit of two integers and returns 1 if at least one of the bits is 1, otherwise returns 0.

Example: 5 | 3 results in 7 (binary: 101 | 011 = 111).

XOR (^): Compares each bit of two integers and returns 1 if the bits are different, otherwise returns 0.

Example: 5 ^ 3 results in 6 (binary: 101 ^ 011 = 110).

NOT (~): Flips all bits of the number (i.e., inverts the bits, changing 1 to 0 and 0 to 1). This is a unary operator.

Example: ~5 results in -6 (binary: ~101 becomes ...111111010 in two's complement).

Left Shift (<<): Shifts the bits of a number to the left by a specified number of positions, effectively multiplying the number by 2 for each shift.

Example: 5 << 1 results in 10 (binary: 101 << 1 = 1010).

Right Shift (>>): Shifts the bits of a number to the right by a specified number of positions, effectively dividing the number by 2 for each shift.

Example: 5 >> 1 results in 2 (binary: 101 >> 1 = 10).

Code Example:

```python
a = 5  # binary: 101
b = 3  # binary: 011

# Bitwise AND
and_result = a & b  # Output: 1 (binary: 001)

# Bitwise OR
or_result = a | b  # Output: 7 (binary: 111)

# Bitwise XOR
xor_result = a ^ b  # Output: 6 (binary: 110)

# Bitwise NOT
not_result = ~a  # Output: -6 (binary: ~101 = ...111111010)

# Left Shift
left_shift = a << 1  # Output: 10 (binary: 1010)

# Right Shift
right_shift = a >> 1  # Output: 2 (binary: 10)

print(f"AND: {and_result}, OR: {or_result}, XOR: {xor_result}, NOT: {not_result}, "
      f"Left Shift: {left_shift}, Right Shift: {right_shift}")
```

Output:

```
AND: 1, OR: 7, XOR: 6, NOT: -6, Left Shift: 10, Right Shift: 2
```

6. Assignment Operator

The Python Operators are used to perform operations on values and variables. These are the special symbols that carry out arithmetic, logical, and bitwise computations. Assignment operators are used to assign values to variables.

Example:

```
#assigns 10 to x
X = 10
```

Operator	Name
=	Assignment Operator
+=	Addition Assignment
-=	Subtraction Assignment
*=	Multiplication Assignment
/=	Division Assignment
%=	Remainder Assignment
**=	Exponent Assignment

Examples:

```
# Integer and float operations
a = 15          # integer
b = 2.5         # float

print(f"Initial a (int): {a}, b (float): {b}")
a += b          # addition assignment (int + float)
print(f"After a += b: {a}")

a -= 5.5        # subtraction assignment (int - float)
print(f"After a -= 5.5: {a}")
```

```
a *= 2          # multiplication assignment (int * int)
print(f"After a *= 2: {a}")

b /= 2          # division assignment (float / int)
print(f"After b /= 2: {b}\n")

# String operations
str1 = "Hello"
str2 = " World"

print(f"Initial str1: {str1}, str2: {str2}")
str1 += str2    # string concatenation
print(f"After str1 += str2: {str1}")

# str1 *= 2      # string repetition
print(f"After str1 *= 2: {str1 * 2}\n")

# List operations
list1 = [1, 2, 3]
list2 = [4, 5]

print(f"Initial list1: {list1}, list2: {list2}")
list1 += list2  # list concatenation
print(f"After list1 += list2: {list1}")

list1 *= 2      # list repetition
print(f"After list1 *= 2: {list1}\n")

# Modulus operation on integers
x = 20
print(f"Initial x: {x}")
x %= 6  # remainder assignment
print(f"After x %= 6: {x}\n")

# Exponentiation
y = 3
print(f"Initial y: {y}")
y **= 4  # exponentiation assignment
print(f"After y **= 4: {y}\n")
```

Output:

```
Initial a (int): 15, b (float): 2.5
After a += b: 17.5
After a -= 5.5: 12.0
After a *= 2: 24.0
After b /= 2: 1.25

Initial str1: Hello, str2:  World
After str1 += str2: Hello World
After str1 *= 2: Hello WorldHello World

Initial list1: [1, 2, 3], list2: [4, 5]
After list1 += list2: [1, 2, 3, 4, 5]
After list1 *= 2: [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]

Initial x: 20
After x %= 6: 2

Initial y: 3
After y **= 4: 81
```

7. Ternary Operator in Python

   The word “ternary” means “composed of three parts.” The ternary operator was named because it involves three operands:

   - A condition
   - An expression to execute if the condition is True
   - Another expression to execute if the condition is false.

   Simple if-else condition:

```
if weather=="sunny":
      print("Go to the beach")
else:
      print("Stay home")
```

   Now let’s translate this decision-making process into a ternary operation:

```
result = "Go to beach" if weather = "sunny" else "stay home"
```

Examples:

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))

min = a if a<b else b
```

```
print(min)


Output:
Enter first number: 10
Enter second number: 15
10
```

Nesting of Ternary Operator:

Nested ternary operators are a way to write more complex conditional statements using the ternary operator. A nested ternary operator is a ternary operator within another ternary operator. It allows us to write multiple conditions in a single line of code.

The syntax for a nested ternary operator is as follows:

```
 [on_true] if [expression] else ([on_true] if [expression] else
[on_false])
```

Example:

To find maximum number between three numbers:

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
c = int(input("Enter third number: "))

max = a if a>b and a > c else b if b > c else c
print(max)
```

Output:

```
Enter first number: 10
Enter second number: 80
Enter third number: 20
80
```

Identity Operator:

is and **is not**

In Python, the is and is not operators are used to compare the identity of two objects, not their values. These operators check whether two variables point to the same object in

memory.
is Operator:
The is operator checks if two variables refer to the same object in memory. It returns True if both variables point to the same object and False otherwise.

is not Operator:
The is not operator checks if two variables refer to different objects in memory. It returns True if they point to different objects and False otherwise.

In the case of fundamental data types it checks the content because they are immutable whereas for other data types like list, dictionary it doesn't check content because they are mutable.

Examples:

```
x = 100
y = 200
print(id(x))
print(id(y))
print(x is y)

x = 100
y = 100
print(id(x))
print(id(y))
print(x is y)

a = [1, 2, 3]
b = [1, 2, 3]
c = a

print(a is b)
print(a is c)

# Example of `is not` operator
print(a is not b)
print(a is not c)

# Example with `None`
x = None
y = None

print(x is None)  # True, both x and None refer to the same None object
print(y is not None)  # False, y is None
```

```
x = 1000
y = 1000

print(x == y)
print(x is y)
```

Output:

```
140727518758424
140727518761624
False
140727518758424
140727518758424
True
False
True
True
False
True
False
True
True
```

Membership Operator in Python

in and **not in** operator

The in and not in operators in Python are used to check whether an element exists in a sequence (such as a list, tuple, string, or dictionary).

in Operator:
The in operator checks if a value exists in a sequence or collection. It returns True if the value is found, otherwise False.

not in Operator:
The not in operator checks if a value does not exist in a sequence or collection. It returns True if the value is not found, otherwise False.

Example:

```
a = "My country name is Nepal"
```

```
print("country" in a)
print("India" in a)


fruits = ['apple', 'banana', 'orange']

print('apple' in fruits)  # True, 'apple' is in the list
print('grape' in fruits)  # False, 'grape' is not in the list
```

Output:

```
True
False
True
False
```