Python Bitwise Operators

geeksforgeeks.org/python-bitwise-operators

November 22, 2019

- Difficulty Level : Medium
- Last Updated : 25 Mar, 2022

Operators are used to perform operations on values and variables. These are the special symbols that carry out arithmetic and logical computations. The value the operator operates on is known as Operand.

Bitwise operators

In Python, bitwise operators are used to performing bitwise calculations on integers. The integers are first converted into binary and then operations are performed on bit by bit, hence the name bitwise operators. Then the result is returned in decimal format.

Note: Python bitwise operators work only on integers.

OPERATOR	DESCRIPTION	SYNTAX
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

Let's understand each operator one by one.
Bitwise AND operator: Returns 1 if both the bits are 1 else 0.
Example:

```
a = 10 = 1010 (Binary)
b = 4 =  0100 (Binary)

a & b = 1010
         &
        0100
      = 0000
      = 0 (Decimal)
```

Bitwise or operator: Returns 1 if either of the bit is 1 else 0.
Example:

```
a = 10 = 1010 (Binary)
b = 4 =  0100 (Binary)

a | b = 1010
         |
        0100
      = 1110
      = 14 (Decimal)
```

Bitwise not operator: Returns one's complement of the number.
Example:

```
a = 10 = 1010 (Binary)

~a = ~1010
   = -(1010 + 1)
   = -(1011)
   = -11 (Decimal)
```

Bitwise xor operator: Returns 1 if one of the bits is 1 and the other is 0 else returns false.
Example:

```
a = 10 = 1010 (Binary)
b = 4 =  0100 (Binary)

a ^ b = 1010
         ^
        0100
      = 1110
      = 14 (Decimal)
```

Python3

```
# Python program to show
# bitwise operators
a  = 10
b  = 4
# Print bitwise AND operation
print ( "a & b =" , a & b)
# Print bitwise OR operation
print ( "a | b =" , a | b)
# Print bitwise NOT operation
print ( "~a =" , ~a)
# print bitwise XOR operation
print ( "a ^ b =" , a ^ b)
```

Output:

```
a & b = 0
a | b = 14
~a = -11
a ^ b = 14
```

Shift Operators

These operators are used to shift the bits of a number left or right thereby multiplying or dividing the number by two respectively. They can be used when we have to multiply or divide a number by two.

Bitwise right shift: Shifts the bits of the number to the right and fills 0 on voids left(fills 1 in the case of a negative number) as a result. Similar effect as of dividing the number with some power of two.

Example:

```
Example 1:
a = 10 = 0000 1010 (Binary)
a >> 1 = 0000 0101 = 5

Example 2:
a = -10 = 1111 0110 (Binary)
a >> 1 = 1111 1011 = -5
```

Bitwise left shift: Shifts the bits of the number to the left and fills 0 on voids right as a result. Similar effect as of multiplying the number with some power of two.
Example:

```
Example 1:
a = 5 = 0000 0101 (Binary)
a << 1 = 0000 1010 = 10
a << 2 = 0001 0100 = 20

Example 2:
b = -10 = 1111 0110 (Binary)
b << 1 = 1110 1100 = -20
b << 2 = 1101 1000 = -40
```

Python3

```
# Python program to show
# shift operators
a = 10
b = -10
# print bitwise right shift operator
print("a >> 1 =", a >> 1)
print("b >> 1 =", b >> 1)
a = 5
b = -10
# print bitwise left shift operator
print("a << 1 =", a << 1)
print("b << 1 =", b << 1)
```

Output:

```
a >> 1 = 5
b >> 1 = -5
a << 1 = 10
b << 1 = -20
```

Bitwise Operator Overloading

Operator Overloading means giving extended meaning beyond their predefined operational meaning. For example operator + is used to add two integers as well as join two strings and merge two lists. It is achievable because the '+' operator is overloaded by int class and str class. You might have noticed that the same built-in operator or function shows different behavior for objects of different classes, this is called **Operator Overloading**.
Below is a simple example of Bitwise operator overloading.

Python3

```
# Python program to demonstrate
# operator overloading
class Geek():
def __init__( self , value):
self .value = value
def __and__( self , obj):
print ( "And operator overloaded" )
if isinstance (obj, Geek):
return self .value & obj.value
else :
raise ValueError( "Must be a object of class Geek" )
def __or__( self , obj):
print ( "Or operator overloaded" )
if isinstance (obj, Geek):
return self .value | obj.value
else :
raise ValueError( "Must be a object of class Geek" )
def __xor__( self , obj):
print ( "Xor operator overloaded" )
if isinstance (obj, Geek):
return self .value ^ obj.value
else :
```

```
        raise ValueError( "Must be a object of class Geek" )
    def __lshift__( self , obj):
        print ( "lshift operator overloaded" )
        if isinstance (obj, Geek):
            return self .value << obj.value
        else :
            raise ValueError( "Must be a object of class Geek" )
    def __rshift__( self , obj):
        print ( "rshift operator overloaded" )
        if isinstance (obj, Geek):
            return self .value & obj.value
        else :
            raise ValueError( "Must be a object of class Geek" )
    def __invert__( self ):
        print ( "Invert operator overloaded" )
        return ~ self .value
# Driver's code
if __name__ = = "__main__" :
    a = Geek( 10 )
    b = Geek( 12 )
    print (a & b)
    print (a | b)
    print (a ^ b)
    print (a << b)
    print (a >> b)
    print (~a)
```

Output:

```
And operator overloaded
8
Or operator overloaded
14
Xor operator overloaded
8
lshift operator overloaded
40960
rshift operator overloaded
8
Invert operator overloaded
-11
```

Note: To know more about operator overloading click here.