

# Python - Bitwise Operators

## Python Bitwise Operators

**Python bitwise operators** are normally used to perform bitwise operations on integer-type objects. However, instead of treating the **object** as a whole, it is treated as a string of bits. Different operations are done on each bit in the **string**.

Python has six bitwise operators - `&`, `|`, `^`, `~`, `<<` and `>>`. All these **operators** (except `~`) are binary in nature, in the sense they operate on two operands. Each operand is a binary digit (bit) 1 or 0.

The following are the bitwise operators in Python -

- Bitwise AND Operator
- Bitwise OR Operator
- Bitwise XOR Operator
- Bitwise NOT Operator
- Bitwise Left Shift Operator
- Bitwise Right Shift Operator

## Python Bitwise AND Operator (`&`)

Bitwise AND operator is somewhat similar to logical and operator. It returns True only if both the bit operands are 1 (i.e. True). All the combinations are –

0 & 0 is 0  
1 & 0 is 0  
0 & 1 is 0  
1 & 1 is 1

When you use integers as the operands, both are converted in equivalent binary, the `&` operation is done on corresponding bit from each number, starting from the least significant bit and going towards most significant bit.

## Example of Bitwise AND Operator in Python

Let us take two integers 60 and 13, and assign them to **variables** `a` and `b` respectively.

```
</>
```

Open Compiler

```
a=60  
b=13  
print ("a:",a, "b:",b, "a&b:",a&b)
```

It will produce the following **output** –

```
a: 60 b: 13 a&b: 12
```

To understand how Python performs the operation, obtain the binary equivalent of each variable.

```
print ("a:", bin(a))  
print ("b:", bin(b))
```

It will produce the following **output** –

```
a: 0b111100  
b: 0b1101
```

For the sake of convenience, use the standard 8-bit format for each number, so that "a" is 00111100 and "b" is 00001101. Let us manually perform and operation on each corresponding bits of these two numbers.

```
0011 1100  
&  
0000 1101  
-----  
0000 1100
```

Convert the resultant binary back to integer. You'll get 12, which was the result obtained earlier.

```
>>> int('00001100',2)  
12
```

Learn **Python** in-depth with real-world projects through our **Python certification course**. Enroll and become a certified expert to boost your career.

## Python Bitwise OR Operator (|)

The "|" symbol (called **pipe**) is the bitwise OR operator. If any bit operand is 1, the result is 1 otherwise it is 0.

```
0 | 0 is 0  
0 | 1 is 1  
1 | 0 is 1  
1 | 1 is 1
```

### Example of Bitwise OR Operator in Python

Take the same values of  $a=60$ ,  $b=13$ . The "|" operation results in 61. Obtain their binary equivalents.

```
</> Open Compiler  
  
a=60  
b=13  
print ("a:",a, "b:",b, "a|b:",a|b)  
print ("a:", bin(a))  
print ("b:", bin(b))
```

It will produce the following **output** –

```
a: 60 b: 13 a|b: 61  
a: 0b111100  
b: 0b1101
```

To perform the "|" operation manually, use the 8-bit format.

```
0011 1100  
|  
0000 1101  
-----  
0011 1101
```

Convert the binary number back to integer to tally the result –

```
>>> int('00111101',2)
61
```

## Python Bitwise XOR Operator (^)

The term XOR stands for exclusive OR. It means that the result of OR operation on two bits will be 1 if only one of the bits is 1.

```
0 ^ 0 is 0
0 ^ 1 is 1
1 ^ 0 is 1
1 ^ 1 is 0
```

## Example of Bitwise XOR Operator in Python

Let us perform XOR operation on a=60 and b=13.

```
</> Open Compiler
a=60
b=13
print ("a:",a, "b:",b, "a^b:",a^b)
```

It will produce the following **output** –

```
a: 60 b: 13 a^b: 49
```

We now perform the bitwise XOR manually.

```
0011 1100
 ^
0000 1101
-----
0011 0001
```

The int() function shows 00110001 to be 49.

```
>>> int('00110001',2)
49
```

## Python Bitwise NOT Operator (~)

This operator is the binary equivalent of logical NOT operator. It flips each bit so that 1 is replaced by 0, and 0 by 1, and returns the complement of the original number. Python uses 2's complement method. For positive integers, it is obtained simply by reversing the bits. For negative number,  $-x$ , it is written using the bit pattern for  $(x-1)$  with all of the bits complemented (switched from 1 to 0 or 0 to 1). Hence: (for 8 bit representation)

```
-1 is complement(1 - 1) = complement(0) = "11111111"
-10 is complement(10 - 1) = complement(9) = complement("00001001") =
"11110110".
```

## Example of Bitwise NOT Operator in Python

For  $a=60$ , its complement is –

```
</>
Open Compiler

a=60
print ("a:",a, ">>> ~a:", ~a)
```

It will produce the following **output** –

```
a: 60 ~a: -61
```

## Python Bitwise Left Shift Operator (<<)

Left shift operator shifts most significant bits to right by the number on the right side of the "<<" symbol. Hence, " $x << 2$ " causes two bits of the binary representation of  $x$  to shift right.

## Example of Bitwise Left Shift Operator in Python

Let us perform left shift on 60.

```
</>
```

```
Open Compiler
```

```
a=60  
print ("a:",a, "a<<2:", a<<2)
```

It will produce the following **output** –

```
a: 60 a<<2: 240
```

How does this take place? Let us use the binary equivalent of 60, and perform the left shift by 2.

```
0011 1100  
<<  
2  
-----  
1111 0000
```

Convert the binary to integer. It is 240.

```
>>> int('11110000',2)  
240
```

## Python Bitwise Right Shift Operator (>>)

Right shift operator shifts least significant bits to left by the number on the right side of the ">>" symbol. Hence, "x >> 2" causes two bits of the binary representation of to left.

### Example of Bitwise Right Shift Operator in Python

Let us perform right shift on 60.

```
</>  
  
a=60  
print ("a:",a, "a>>2:", a>>2)
```

Open Compiler

It will produce the following **output** –

```
a: 60 a>>2: 15
```

Manual right shift operation on 60 is shown below –

```
0011 1100  
>>  
2  
-----  
0000 1111
```

Use int() function to convert the above binary number to integer. It is 15.

```
>>> int('00001111',2)  
15
```