Bitwise Operator

- The Bitwise operator are AND, OR, XOR, Complement, Left shift , Right Shift.
- These operators work on integral type of data and they perform operation on Binary representation of data I.e, 0's and 1's.
- Bitwise operators are used in applications of networking , Encryption and Decryption etc.
- Bitwise calculations starts from right hand side.

Operator	
&	AND
|	OR
^	XOR
~	Complement
<<	Left Shift
>>	Right Shift

Understanding Binary numbers system :

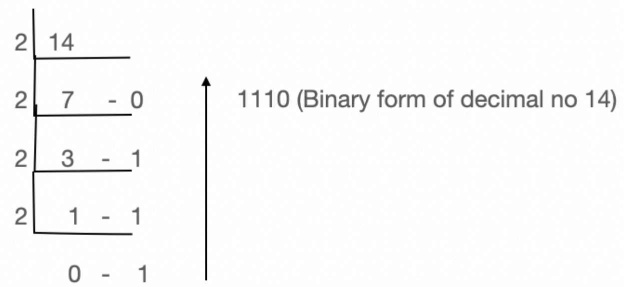
- To understand bitwise we first need to understand binary number system . Binary number system uses only 0's and 1's
- Decimal number system uses numbers from 0 - 9
- To convert decimal to binary we can do it in either 2 ways

```
0  →  0    # 0 is 0

1  →  1    # 1 is 1

   +   1   # adding 1 to 1 to make it 2
-----------
2  → 10    # binary form of decimal no 2

     + 1   # adding 1 to binary of 2 to make it 3
-----------
3  → 11    # binary form of decimal no 3   so on…

     + 1
-----------
4  → 1 0 0

      + 1
-----------
5  → 1 0 1
```

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Table for decimal to binary

Consider decimal number as - 14



Let us check this in IDLE

```
Python 3.9.7 (v3.9.7:1016ef3790, Aug 30 2021, 16:39:15)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> a=10
>>> format(a, 'b')
'1010'
>>> a=14
>>> format(a, 'b')
'1110'
>>> format(25,'b')
'11001'
>>> a=25
>>> bin(a)
'0b11001'
>>> a.bit_length()
5
>>>
```

Bitwise Operations

- Let us understand all the bitwise operators with an example
- Consider a = 10 (binary of 10 is 1010)

  b = 13 (binary of 13 is 1101)

AND operations (&)

- AND works on multiplication
- Working of AND -

- 1 * 1 = 1
  1 * 0 = 0
  0 * 1 = 0
  0 * 0 = 0

Ex : a - 1010

b - 1101

---

a & b - 1000 = 8 (Decimal form of binary number)

OR operations (|)

- OR works on addition
- Working of OR - 1 + 1 = 1
  1 + 0 = 1
  0 + 1 = 1
  0 + 0 = 1

Ex : a - 1010
b - 1101

---

a | b - 1111 = 15 (decimal form of binary number)

XOR operations (^)

- Working of XOR - 1 ^ 1 = 0
  1 ^ 0 = 1
  1 ^ 1 = 1
  1 ^ 0 = 0

Ex: a - 1010
b - 1101

---

a ^ b - 0111 = 7 (decimal form of binary number)

- Leading zero doesn't make sense
- Hence , we consider 111 whose decimal form is 7

Left shift (<<) and right shift (>>)

- a = 10 (binary of 10 = 1010)
- If we left shift ‘ a ’ by one place I.e, a << 1 then

1 0 1 0

1 0 1 0 0 ← Left shift by one place

Extra space filled with value zero

- After left shift by one place the bit that is freed will be taken as zero.
- Now 10100 = 20 = 2* 10
- If we left shift 20 again then,

1 0 1 0 0

1 0 1 0 0 0 = 40 = 2 * 20

- We see that when we left shift the number will double for one place I.e; a << n , a * 2 (power n)
- If we double the left shift I.e , a. << 2 then

a << 2 = 2(power n) * a = 4*10 = 40

- similarly , if we do a << 5 , then the value gets double for 5 times

a << 5 = 2(power 5) * a = 32 * 10 = 320

- If we RIGHT SHIFT the number will become half.

a = 10 ->   1 0 1 0

__ 1 0 1 0 (this 0 gets discarded)

This leading empty space has no value (it becomes zero)

- Right shift operator divide by 2 I.e; a >> n a / 2 (power n)