Bitwise Operators
These operators are used to perform operations on binary data.

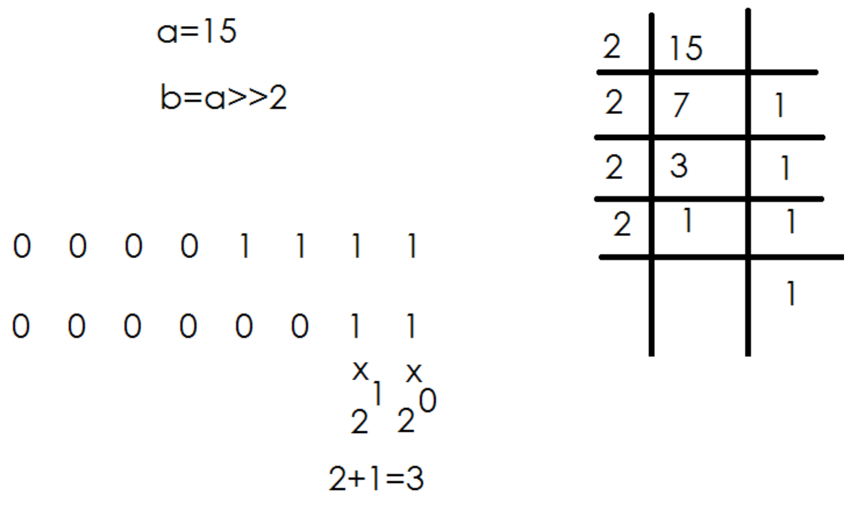
1. Left shift operator <<
2. Right shift operator >>
3. Bitwise & (and) operator
4. Bitwise | (or) operator
5. Bitwise ^ (XOR) operator
6. Bitwise ~ (not) operator

Bitwise Right Shift Operator >>
This operator is used to shift number of bits towards right side.
This shift operators are used in memory management or runtime environment (OS, PVM, JVM,..)

Syntax: opr>>n

Note: bitwise operators are applied only on integer data type.

opr → integer value
n → integer value, which define how many bits are shifted towards right side.



```
>>> a=15
>>> bin(a)
```

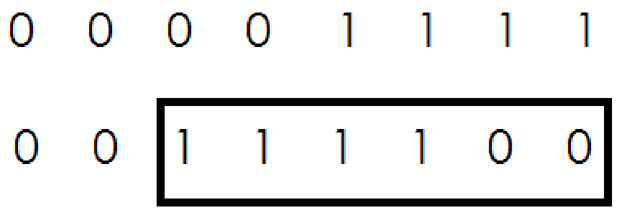
```
'0b1111'
>>> b=a>>2
>>> bin(b)
'0b11'
>>> print(a)
15
>>> print(b)
3
```

Formula: opr//2 pow n

Left shift operator
This operator is used to shift number of bits towards left side.

Syntax: opr<<n

```
a=15
b=a<<2
```



```
>>> a=15
>>> b=a<<2
>>> bin(a)
'0b1111'
>>> bin(b)
'0b111100'
>>> print(a)
15
```

```
>>> print(b)
60
```

Formula: opr*2 pow n

Logic Gates

Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic. Based on this, logic gates are named as AND gate, OR gate, NOT gate etc.

1. Bitwise & (and) operator
2. Bitwise | (or) operator
3. Bitwise ^ (XOR) operator
4. Bitwise ~ (not) operator

Bitwise (&) and operator
This operator is used to apply and gate
Truth table

Opr1	**Opr2**	**Opr1&Opr2**
1	1	1
1	0	0
0	1	0
0	0	0

```
>>> a=0b1010
>>> b=0b1100
>>> c=a&b
>>> print(bin(a))
0b1010
>>> print(bin(b))
0b1100
>>> print(bin(c))
0b1000
>>> print(a,b,c)
10 12 8
```

Bitwise | (or) operator
This operator is used to apply or gate
Truth table of | operator

Opr1	**Opr2**	**Opr1 | Opr2**
1	0	1
0	1	1
0	0	0
1	1	1

```
>>> a=0b1010
>>> b=0b1100
>>> c=a|b
>>> print(bin(a),bin(b),bin(c),sep="\n")
0b1010
0b1100
0b1110
>>> print(a,b,c)
10 12 14
>>> a=10
>>> b=12
>>> c=a|b
>>> print(bin(a),bin(b),bin(c))
0b1010 0b1100 0b1110
>>> print(a,b,c)
10 12 14
```

Bitwise XOR(^) operator
The XOR gate is also known as the exclusive OR gate. The output of an XOR gate is 1 if either input is 1 but not both, and 0 if both inputs are the same.

Opr1	Opr2	Opr1^Opr2
1	0	1
0	1	1
0	0	0

1	1	0

```
>>> a=0b1110
>>> b=0b1101
>>> c=a^b
>>> print(bin(a),bin(b),bin(c),sep="\n")
0b1110
0b1101
0b11
>>> print(a,b,c)
14 13 3
```

```
n1=14
n2=13
n3=n1^n2
print(bin(n1),bin(n2),bin(n3))
0b1110 0b1101 0b11
```

Bitwise (~) not operator

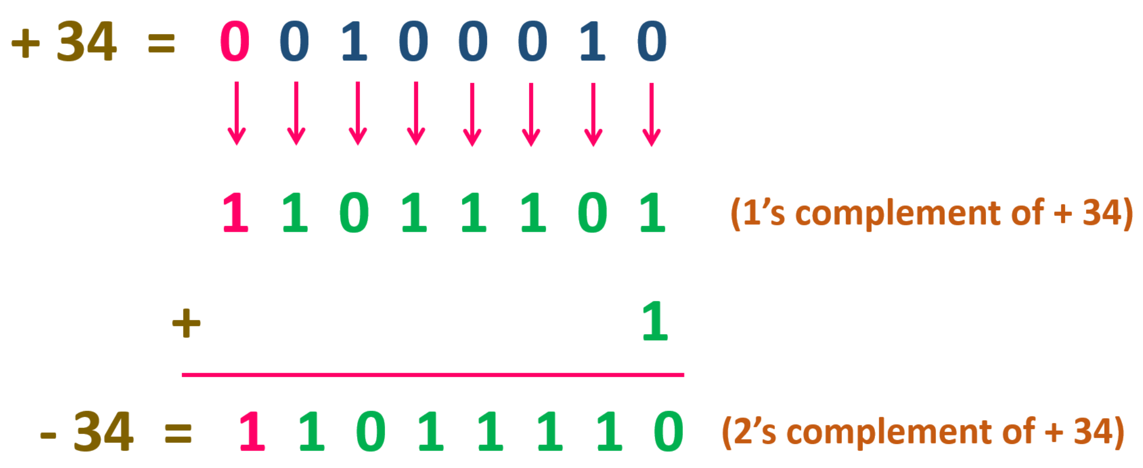
Python's ***bitwise NOT*** operator ~x inverts each bit from the binary representation of integer x so that 0 becomes 1 and 1 becomes 0. This is semantically the same as calculating ~x == -x-1. For example, the bitwise NOT expression ~0 becomes -1, ~9 becomes -10, and ~32 becomes -33.

```
>>> ~32
-33
```

Example

```
>>> a=0b0
>>> b=~a
>>> print(bin(a),bin(b))
0b0 -0b1
>>> n1=0b1100
>>> n2=~n1
>>> print(bin(n1),bin(n2))
0b1100 -0b1101
```

To represent -34 in 2’s complement form



The expression ~32 operates on the bit representations "0100000" (decimal 32) and performs ***bitwise NOT*** resulting in binary "1011111". This corresponds to the negative decimal number -33.

The NOT gate takes one input and provides the opposite output. If the input is 1, the output is 0, and if the input is 0, the output is 1.

not gate is applied using “not” keyword (logical operator)

```
>>> b=not a
>>> print(bin(a),bin(b))
0b1 0b0
>>> c=not b
>>> print(bin(b),bin(c))
0b0 0b1
```

Walrus Operator (:=)
Walrus operator is introduced in python 3.8 version.
Walrus operator is also called assignment expression operator.

=	-- assignment operator
:=	-- walrus operator or assignment expression
==	-- relational operator or equal

Example:

```
>>> a=10+20+b=10-5
SyntaxError: cannot assign to expression
>>> a=10
>>> b:=20
SyntaxError: invalid syntax
```