Bitwise Operations in Python

Bitwise operators operate on the binary (bit-level) representation of integers. They're commonly used for:

- Low-level device control
- Data compression
- Cryptography
- Performance-critical applications

Each bitwise operation affects individual bits in a binary number.

◆ 1. & - Bitwise AND

python	О Сору	* Edit
5 & 3 # 0101 & 0011 = 0001 → 1		

How it works:

- Compares each bit of two numbers.
- Returns 1 only if both bits are 1.

Bit position	5 (0101)	3 (0011)	Result
3	0	0	0
2	1	0	0
1	0	1	0
0	1	1	1

Result: 0001 = 1

♦ 2. | – Bitwise OR

python

How it works:

- Compares each bit.
- Returns 1 if at least one bit is 1.

Bit position	5 (0101)	3 (0011)	Result
3	0	0	0
2	1	0	1
1	0	1	1
0	1	1	1

Result: 0111 = 7

♦ 3. ^ – Bitwise XOR (Exclusive OR)

python	Сору	* Edit
5 ^ 3 # 0101 ^ 0011 = 0110 → 6		

How it works:

• Returns 1 only if the bits are different.

Bit position	5 (0101)	3 (0011)	Result
3	0	0	0
2	1	0	1
1	0	1	1
0	1	1	0

Bit position

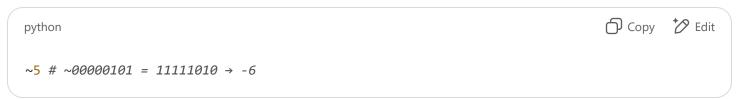
5 (0101)

3 (0011)

Result

Result: 0110 = 6

♦ 4. ~ – Bitwise NOT (One's Complement)

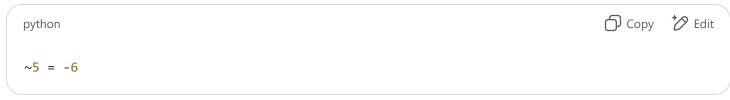


How it works:

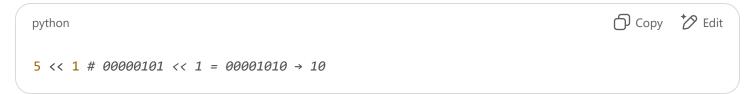
- Inverts every bit: $0 \rightarrow 1$, $1 \rightarrow 0$
- For integers, the result is in two's complement form (used for negative numbers).

Python uses infinite-length signed integers, so:





♦ 5. << - Left Shift</p>

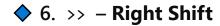


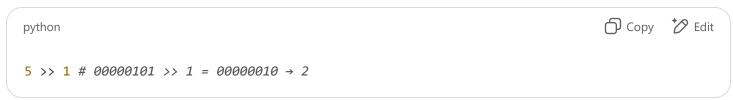
How it works:

- Shifts bits to the **left**, adding 0 s at the right.
- Effectively multiplies the number by 2ⁿ.

python $5 << 2 = 5 \times 2^2 = 20$







How it works:

- Shifts bits to the right, dropping bits off the end.
- Effectively divides the number by 2ⁿ, discarding remainder.

```
python

5 >> 2 = 5 // 4 = 1
```

Example with Binary Output in Python

Summary Table

Operator	Name	Description
&	AND	1 only if both bits are 1

Operator	Name	Description
,	•	OR
۸	XOR	1 if bits are different
~	NOT	Invert all bits
<<	Left Shift	Shift bits left (multiply by 2 ⁿ)
>>	Right Shift	Shift bits right (divide by 2 ⁿ)