logical operators with example

Here is a **complete explanation of Logical Operators in Python** along with **examples and truth table** 👇

🔐 Logical Operators in Python

Logical operators are used to combine **conditional statements** (conditions that return True or False).
Python has **three logical operators**:

Operator	Description	Example
`and`	True only if **both conditions are True**	`a > 10 and a < 20`
`or`	True if **at least one condition is True**	`a > 10 or a < 5`
`not`	**Reverses** the result (True → False, False → True)	`not(a > 10)`

📌 Example in Python

python Copy code

```python
a = 15
b = 5

# AND operator
print(a > 10 and b < 10)   # True (both conditions True)

# OR operator
print(a < 10 or b < 10)    # True (one condition True)

# NOT operator
print(not(a > 10))         # False (because a > 10 is True)
```

📍 Truth Table of Logical Operators

✔ AND Operator (and)

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

✔ OR Operator (or)

A	B	A or B
True	True	True

A	B	A or B
True	False	True
False	True	True
False	False	False

✔️ NOT Operator (`not`)

A	not A
True	False
False	True

🎯 Real Life Example

python Copy code

```python
age = 22
have_id = True

if age >= 18 and have_id:
    print("You are allowed to vote.")
else:
    print("You cannot vote.")
```

💡 Interview Tips

- `and` → Both conditions must be true
- `or` → At least one condition must be true
- `not` → Reverses the boolean value

Would you like **practice questions** with solutions? 😊👍

Bitwise example with all concept

Here is a **complete explanation of Bitwise Operators in Python** with all key concepts + examples + interview notes 👇

⚡ Bitwise Operators in Python

Bitwise operators work on **bits (0 and 1)** of integers.
They perform operations at the **binary (bit) level**.

Operator	Symbol	Meaning
AND	`&`	Returns 1 if **both bits** are 1

Operator	Symbol	Meaning
OR	|	Returns 1 if **any bit** is 1
XOR	^	Returns 1 if **bits are different**
NOT	~	Flips bits (gives negative result in Python)
Left Shift	<<	Shifts bits **left** (multiply by 2)
Right Shift	>>	Shifts bits **right** (divide by 2)

🧱 Bitwise AND `&` Example

python Copy code

```python
a = 5      # 5 = 0101
b = 3      # 3 = 0011

print(a & b)   # 0101 & 0011 = 0001 → 1
```

🧠 **Output:** `1`

🔦 Bitwise OR `|` Example

python Copy code

```python
a = 5   # 0101
b = 3   # 0011

print(a | b)   # 0111 → 7
```

🧠 **Output:** `7`

🔀 Bitwise XOR `^` Example

python Copy code

```python
a = 5   # 0101
b = 3   # 0011

print(a ^ b)   # 0110 → 6
```

🧠 **Output:** `6`

📌 XOR gives 1 when **bits are unequal**.

🌀 Bitwise NOT `~` Example

python Copy code

```python
a = 5   # 0101
print(~a)
```

🧠 **Output:** `-6`

📌 NOT formula:

Copy code

```
~x = -(x + 1)
```

So: `~5 = -(5 + 1) = -6`

⏩ Left Shift `<<` Example

python Copy code

```python
a = 5          # 5 = 0101
print(a << 1)  # 1010 = 10
print(a << 2)  # 10100 = 20
```

📌 Left shift = Multiply by `2^n`

So:

- `5 << 1 = 5 * 2 = 10`
- `5 << 2 = 5 * 4 = 20`

⏪ Right Shift `>>` Example

python Copy code

```python
a = 20         # 20 = 10100
print(a >> 1)  # 1010 = 10
print(a >> 2)  # 101 = 5
```

📌 Right shift = Divide by `2^n`

So:

- `20 >> 1 = 20 / 2 = 10`
- `20 >> 2 = 20 / 4 = 5`

📃 Bitwise Operator Truth Table

A	B	A & B	A | B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

🧠 Interview Questions Based on Bitwise Operators

📌 Q1: Swap two numbers without using third variable

python Copy code

```
a = 10
b = 20

a = a ^ b
b = a ^ b
a = a ^ b

print(a, b)  # 20 10
```

📌 Q2: Check if a number is odd or even using bitwise

python Copy code

```
num = 7

if num & 1:
    print("Odd")
else:
    print("Even")
```

📌 Q3: Multiply a number by 8 using bitwise

bash Copy code

```
num << 3
```

🎉 Want practice questions with answers?

Just reply **"Yes, send practice questions"** 😊

m = 10 1010 010100 101000
n = 4 0100
128 64 32 16 8 4 2 1 explain

Here is the full explanation of your example with **binary conversion + shifting + bit positions** 👇

📌 Given:

ini Copy code

```
m = 10 → 1010
n = 4  → 0100
```

Before understanding this, we must know **how binary bits represent numbers**.

🧮 Binary Position Table

Bit Position	128	64	32	16	8	4	2
Power of 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1

📌 **Every binary number is made using these values.**

🔢 Convert `m = 10` to Binary

Find numbers that add up to 10 using the above table:

```ini
10 = 8 + 2
```

8	4	2	1
1	0	1	0

So, **10 = 1010 in binary** ✔

```ini
m = 10 → 1010
```

🔢 Convert `n = 4` to Binary

```ini
4 = 4
```

8	4	2	1
0	1	0	0

So, **4 = 0100 in binary** ✔

```ini
n = 4 → 0100
```

📍 Understanding Your Question:

You wrote:

```ini
m = 10  1010     010100  101000
```

This is showing **left shift operations**:

Expression	Meaning	Result Binary	Value
1010	original	1010	10
1010 << 1	shift left by 1 bit	010100	20

Expression	Meaning	Result Binary	Value
1010 << 2	shift left by 2 bits	101000	40

🔁 Left Shift Rule

bash Copy code

```
x << n = x × (2^n)
```

So:

- `10 << 1 = 10 × 2 = 20`
- `10 << 2 = 10 × 4 = 40`

🔎 Example in Python

python Copy code

```python
m = 10

print(m << 1)  # 20
print(m << 2)  # 40
```

✔️ Final Explanation Summary

Concept	Example
Binary values	128 64 32 16 8 4 2 1
10 in binary	1010
4 in binary	0100
Left shift 1	`1010 << 1 = 010100 = 20`
Left shift 2	`1010 << 2 = 101000 = 40`

🎯 Want more?

Do you want **right shift examples + negative number bit representation** also?
Reply **Yes** 😊

print(m & n) 0000

You wrote:

scss Copy code

```
print(m & n) → 0000
```

Let's explain **why the output becomes 0000.**

📌 Given:

ini Copy code

```
m = 10 → 1010
n = 4  → 0100
```

🧠 Bitwise AND `&`

The **AND operator** compares each bit:

Bit	Rule
1 & 1 → 1	
1 & 0 → 0	
0 & 1 → 0	
0 & 0 → 0	

🔍 Calculate `m & n`

yaml Copy code

```
  1010   (10)
& 0100   (4)
------
  0000
```

➡️ Why?

There is **no position where both bits are 1 at the same time**.

🧠 Final Output

python Copy code

```
print(m & n)   # 0
```

✔ **Binary result = 0000**
✔ **Decimal value = 0**

🎁 Small Tip (Interview)

If two numbers **do not share any common bit of 1**,
their AND will always be **0**.

Would you like **OR | and XOR ^ explanation with the same numbers (10 & 4)?** 😊

print(m | n) (01,10,11)=1...00=0 1110

Let's clearly explain your statement:

```scss
print(m | n)
```

We know:

```ini
m = 10 → 1010
n = 4  → 0100
```

🔦 Bitwise OR | Rules

Bits Compared	Result
1 OR 1 → 1	
1 OR 0 → 1	
0 OR 1 → 1	
0 OR 0 → 0	

📌 **If any one bit is 1, result becomes 1**

🔍 Operation

```yaml
  1010   (10)
| 0100   (4)
------
  1110
```

🧠 Explanation of Each Bit:

m	n	m | n
1	0	1
0	1	1
1	0	1
0	0	0

👉 Final = **1110**

🎯 Convert 1110 to Decimal

ini Copy code

```ini
1110 = 8 + 4 + 2 + 0 = 14
```

✔️ Final Output

python Copy code

```python
print(m | n)  # 14
```

- ✳️ **Binary = 1110**
- ✳️ **Decimal = 14**

📌 Tip (Very Important)

Bitwise OR produces **1** if **at least one bit is 1**.

Would you like to see **XOR ^ example for 10 and 4 next?** 😊

print(m ^ n) (01,10)=1...(00,11)=0 1110

Let's explain your line:

bash Copy code

```bash
print(m ^ n)
(01,10)=1 … (00,11)=0     → 1110
```

🔀 Bitwise XOR ^ Rules

Bits Compared	Result
0 ^ 0 → 0	
1 ^ 1 → 0	
0 ^ 1 → 1	
1 ^ 0 → 1	

📌 XOR gives 1 only when bits are different.

📌 Given:

ini Copy code

```
m = 10 → 1010
n = 4  → 0100
```

🔍 Operation: m ^ n

yaml Copy code

```
   1010   (10)
 ^ 0100   (4)
  ------
   1110
```

🧠 Bit-by-bit Explanation:

m	n	m ^ n	Explanation
1	0	1	Bits different
0	1	1	Bits different
1	0	1	Bits different
0	0	0	Bits same

👉 Final result = **1110**

🎯 Convert 1110 to Decimal

ini Copy code

```
1110 = 8 + 4 + 2 + 0 = 14
```

✔️ Final Output

python Copy code

```
print(m ^ n)  # 14
```

📌 **Binary = 1110**
📌 **Decimal = 14**

💡 Quick Tip (Interview)

- XOR (^) is used when you want to detect differences between bits.
- Used in **encryption, swapping numbers, error detection**.

🎁 Want an example of XOR used to *swap numbers without third variable*?

Reply **Yes** 😊

print(m >> 2) 1010->>> 0101 ->>> 0010

Let's explain your line:

yaml Copy code

```
print(m >> 2)
1010 → 0101 → 0010
```

📌 Given:

ini Copy code

```
m = 10
Binary = 1010
```

⏪ Bitwise Right Shift >>

Right shift **moves bits to the right**.
Each shift **divides the number by 2** (integer division).

🧮 Step-by-Step Shifting

🔸 **Step 1:** `m >> 1`

scss Copy code

```
  1010   (10)
→ 0101   (5)
```

📌 10 → 5 (divided by 2)

🔸 **Step 2:** `m >> 2`

scss Copy code

```
  0101   (5)
→ 0010   (2)
```

📌 5 → 2 (again divided by 2)

✔️ Final Output

python Copy code

```
print(m >> 2)   # 2
```

🧠 Binary Explanation

yaml Copy code

```
m = 10 = 1010

1010 >> 2 = 0010 (2)
```

💡 Mathematical Shortcut

nginx Copy code

```
m >> n = m / (2^n)
```

So:

Copy code

```
10 >> 2 = 10 / (2^2) = 10 / 4 = 2
```

🎉 Final Summary Table

Operation	Binary Result	Decimal
10 >> 1	0101	5
10 >> 2	0010	2

Would you like to learn how **right shift works with negative numbers (`-10 >> 2`)**?
Reply **Yes** 😊

print(m << 2) 0000 1010
0001 0100
0010 1000

Let's explain your example:

yaml Copy code

```
print(m << 2)
0000 1010
0001 0100
0010 1000
```

📌 Given:

ini Copy code

```
m = 10
Binary = 0000 1010
```

⏩ Bitwise Left Shift `<<`

Left shift **moves bits to the left**.
Each shift **multiplies the number by 2**.

🧮 Step-by-Step Shifting

🔸 Step 1: `m << 1`

yaml Copy code

```
  0000 1010   (10)
→ 0001 0100   (20)
```

✔️ Shifting left once = multiply by 2

`10 × 2 = 20`

🔸 Step 2: `m << 2`

yaml Copy code

```
  0001 0100   (20)
→ 0010 1000   (40)
```

✔️ Shifting left twice = multiply by 4

`10 × 4 = 40`

✔️ Final Output

python Copy code

```python
print(m << 2)   # 40
```

🧠 Binary Final:

yaml Copy code

```
0010 1000 = 40
```

💡 Mathematical Rule

bash Copy code

```
m << n = m × (2^n)
```

So:

bash Copy code

```
10 << 2 = 10 × (2^2) = 10 × 4 = 40
```

🎯 Summary Table

Operation	Binary	Decimal
10	0000 1010	10
10 << 1	0001 0100	20
10 << 2	0010 1000	40

✔ Want more practice?

Should I send **bitwise questions for interview practice?** 😊
Reply **Yes** if you want!

Want more practice?

Great! Here are **Bitwise Operator Practice Questions** (from easy to interview level). Try to solve them, and I will provide answers after you attempt 😊

🟢 Level 1 — Basic Practice

1. Convert the following decimal numbers to binary:
   - a) 12
   - b) 25
   - c) 33
2. Evaluate:

```python
print(8 & 5)
```

3. Evaluate:

```python
print(7 | 2)
```

4. Evaluate:

```python
print(6 ^ 3)
```

5. Predict output:

```python
print(10 >> 1)
```

6. Predict output:

```python
print(12 << 2)
```

🔵 Level 2 — Moderate Practice

7. Swap two numbers without using third variable using XOR.
   (Write full code)
8. Check whether a number is EVEN or ODD using bitwise.
9. Convert to decimal:

Copy code

```
101101
```

10. What is the output?

python Copy code

```python
a = 15  # 1111
b = 10  # 1010
print(a & b)
print(a | b)
print(a ^ b)
```

🔴 Level 3 — Interview Questions

11. Using bitwise operator, multiply a number by 8.
    (Write expression only)
12. Using bitwise operator, divide a number by 4.
    (Write expression only)
13. Find output:

python Copy code

```python
x = 5
y = x << 3
z = y >> 2
print(z)
```

14. Write code to count how many bits are 1 in a number.
    (Hint: AND + SHIFT)
15. Predict output (tricky):

python Copy code

```python
print(~5)
print(~(-6))
```

🧠 Want Solution?

Reply **"Show Answers"** and I'll provide **step-by-step solutions with explanation**! 😊 💡

interview quetions

Here are **Top Bitwise Interview Questions (with answers)** 💡
These are commonly asked in **FAANG, MAANG & major product-based companies.**

🧩 1) Swap two numbers without using a third variable

Using XOR:

python Copy code

```python
a = 10
b = 20

a = a ^ b
b = a ^ b
a = a ^ b

print(a, b)  # 20 10
```

📌 XOR swaps because repeated XOR cancels bits.

🔢 2) Check if a number is even or odd without using modulus

python Copy code

```python
num = 7
if num & 1:
    print("Odd")
else:
    print("Even")
```

📌 Odd numbers always have last bit = `1`.

🚀 3) Multiply a number by 8 without using `*` operator

python Copy code

```python
x << 3     # Multiply by 2^3
```

📌 Left shift multiplies by powers of 2.

🎯 4) Divide a number by 4 without using `/`

python Copy code

```python
x >> 2     # Divide by 2^2
```

📌 Right shift divides by powers of 2 (integer division).

🔍 5) Check if a number is a power of 2

python Copy code

```python
def isPowerOfTwo(n):
    return n > 0 and (n & (n - 1)) == 0

print(isPowerOfTwo(16))   # True
print(isPowerOfTwo(18))   # False
```

📌 Power of 2 numbers have only **one bit = 1** (e.g., 2, 4, 8, 16...).

🔐 6) Count number of 1s in the binary representation

python Copy code

```python
def countSetBits(n):
    count = 0
    while n:
        n = n & (n - 1)
        count += 1
    return count

print(countSetBits(13))  # 13=1101 => 3
```

📌 Clever trick: `(n & (n-1))` removes the last `1`.

🔄 7) Reverse bits of a number

Example: Reverse 5 (0101 → 1010)

python Copy code

```python
def reverseBits(n):
    r = 0
    for _ in range(4): # assuming 4 bit numbers
        r = (r << 1) | (n & 1)
        n >>= 1
    return r

print(reverseBits(5))   # 10
```

🔁 8) Find missing number using XOR

python Copy code

```python
arr = [1, 2, 4, 5]
miss = 0

for i in range(1, 6):  # 1 to 5
    miss ^= i

for x in arr:
    miss ^= x

print(miss)  # 3 is missing
```

📌 XOR cancels duplicates.

❓ Want More?

We can continue with advanced problems like:

- ✔ Bitmasking
- ✔ Subset generation using bits
- ✔ Nth magic number
- ✔ Fast exponentiation using bits

Would you like **Advanced Bit Manipulation Questions (for DSA + CP)?**
Reply 👉 **"Yes, Advanced"** 😊🔥

We can continue with advanced problems like:

- ✔ Bitmasking
- ✔ Subset generation using bits
- ✔ Nth magic number
- ✔ Fast exponentiation using bits

Awesome! Let's continue with **Advanced Bit Manipulation Problems** 🔥
These are very important for **DSA, Competitive Programming, and Product-Based Interviews.**

🚀 1) Bitmasking (Using Bits to Represent Sets)

Bitmasking allows us to **store subsets using integers**, where each bit represents whether an item is present (1) or absent (0).

📌 Example

Suppose we have 4 elements `{A, B, C, D}`

Bit Position	3	2	1	0
Element	A	B	C	D

mathematica Copy code

```mathematica
Set {B, D} = 0101  (Binary) = 5 (Decimal)
```

⭐ Operations

python Copy code

```python
mask = 0

# Add element (set bit)
mask |= (1 << 2)

# Remove element (unset bit)
mask &= ~(1 << 1)

# Check if element exists
```

```
if mask & (1 << 3):
    print("present")
```

🧩 2) Subset Generation Using Bits

If there are **N elements** → total subsets = `2^N`.

We use numbers from `0` to `(1<<N)-1`.

📌 Example Code

python Copy code

```
arr = [1, 2, 3]
n = len(arr)

for mask in range(1<<n):  # 0 to 7
    sub = []
    for i in range(n):
        if mask & (1 << i):
            sub.append(arr[i])
    print(sub)
```

🔥 Output

css Copy code

```
[]
[1]
[2]
[1,2]
[3]
[1,3]
[2,3]
[1,2,3]
```

🔮 3) Nth Magic Number (Google Favorite)

Magic numbers are formed using powers of 5.

For example: `1, 5, 25, 125...`

To find Nth magic number:

- Consider binary representation of N.
- Replace each bit with corresponding power of 5 and sum.

📌 Formula

arduino Copy code

```
Nth magic number = sum of (bit * 5^position)
```

💡 Example Code

python Copy code

```python
def magicNumber(n):
    ans = 0
    base = 5
    while n:
        if n & 1:
            ans += base
        base *= 5
        n >>= 1
    return ans

print(magicNumber(3))  # Output: 30
```

⚡ 4) Fast Exponentiation Using Bits (Binary Exponentiation)

Time Complexity: **O(logN)** instead of O(N).

📌 Example: Find `a^b`

python Copy code

```python
def fastExpo(a, b):
    res = 1
    while b > 0:
        if b & 1:         # If last bit is 1
            res *= a
        a *= a            # Square base
        b >>= 1           # Right shift exponent
    return res

print(fastExpo(2, 10))  # 1024
```

🏅 Summary Table