Bitwise Not Operator

🔹 **STEP 1: Convert the number to binary**

Take the number:

5

Binary of 5 is:

101

👉 Computers don't work with just 3 bits, they use **fixed size bits** (8 bits, 32 bits, etc.).

So we write 5 in **8-bit binary**:

5 = 00000101

(Leading zeros are normal and important.)

🔹 **STEP 2: Apply bitwise NOT (~) → 1's complement**

~ means **flip every bit**:

- 0 → 1
- 1 → 0

So flip **each bit** of 00000101:

Original (5): 00000101

After ~ (flip): 11111010

👉 This result is called **1's complement**.

⚠️ Important:

- This is **NOT the final answer yet**
- This binary represents a **negative number**, but computers use **2's complement**, not 1's.

🔹 **STEP 3: Why 2's complement is needed**

Computers store **negative numbers** using **2's complement**, not 1's complement.

So when Python sees:

11111010

it treats it as a **2's complement negative number**.

To understand its value, we must interpret it using **2's complement rules**.

🔹 **STEP 4: Convert to 2's complement (to find the value)**

To find the decimal value of a **negative 2's complement number**, do this:

Rule:

1. Take **1's complement**
2. Add **1**
3. Convert to decimal
4. Add a **minus sign**

Apply to our number:

Binary we got:

11111010

1️⃣ **Take 1's complement (flip again):**

00000101

2️⃣ **Add 1:**

00000101

\+ 1

00000110

3️⃣ **Convert to decimal:**

00000110 = 6

4️⃣ **Add minus sign:**

-6

✅ **FINAL ANSWER**

~5 = -6

🔹 **VERY IMPORTANT SHORTCUT (Python rule)**

Instead of doing all these steps every time, Python follows this **always-true formula**:

~x = -(x + 1)

So:

~5 = -(5 + 1) = -6

~3 = -(3 + 1) = -4

~0 = -(0 + 1) = -1

🔹 **Why this works (intuitive explanation)**

- ~ flips bits (1's complement)
- Python stores negatives using 2's complement
- 2's complement mathematically equals -(x + 1)

That's why the formula works **every time**.

🔑 **FINAL MEMORY POINTS (please remember these)**

1. Convert number to binary
2. ~ flips all bits → 1's complement
3. Python uses **2's complement** for negatives
4. Therefore:

~x = -(x + 1)

🔑 **KEY IDEA (this decides everything)**

In **2's complement representation** (which Python uses):

The leftmost bit (MSB – Most Significant Bit) decides the sign

- 0 → **Positive number**
- 1 → **Negative number**

This leftmost bit is called the **sign bit**.

🔹 **Step 1: Look at the binary after ~**

From your example:

After ~ (flip): 11111010

Now look at the **leftmost bit**:

1 1111010

↑

sign bit

👉 The sign bit is **1**

✅ That means this number is **NEGATIVE**

That's how we determine it.

🔹 **Step 2: Why does 1 mean negative?**

Because computers use **2's complement** to store signed integers.

In 2's complement:

Sign bit	Meaning
0	Positive or zero
1	Negative

This rule is **fixed** in computer systems.

🔹 **Now let's map this to your question**

❓ **"If in 1's complement when it is 0 or 1 what to do?"**

✅ **Correct interpretation:**

- If **leftmost bit = 0**
  → number is **positive**
  → **STOP**
  → convert directly to decimal
- If **leftmost bit = 1**
  → number is **negative**
  → **MUST take 2's complement**
  → then convert to decimal
  → add minus sign

🔑 FINAL ANSWER (memorize this)

If sign bit is `1` → take 2's complement
If sign bit is `0` → do NOT take 2's complement

🔹 **First: What is "zero" in decimal?**

In normal math, **zero is only one value**:

0

There is **no +0 and -0** in real life math.

🔹 **Now see what happens in 1's complement**

Assume **8-bit numbers** (just for understanding).

Positive zero

00000000 → 0

Rule of 1's complement

Negative = flip all bits

So flip 00000000:

11111111

This is treated as **negative zero** in 1's complement.

❌ **Result in 1's complement**

Binary	Meaning
00000000	+0
11111111	-0

👉 **Two different binaries for zero**
👉 This is what "two zeros" means ❌

🔹 **Why is this BAD?**

Because now the computer has to handle:

- +0
- -0

Questions like:

- Is +0 equal to -0?
- Which zero should I store?

- Which zero should I print?

This **confuses hardware and logic**.

🔹 **Now see 2's complement (this fixes it)**

Rule of 2's complement

Negative = (flip bits) + 1

Let's try to make **-0**.

Start with zero:

00000000

Flip bits (1's complement):

11111111

Add 1:

11111111

+ 1

00000000

✅ **Result in 2's complement**

Binary	**Meaning**
00000000	0

👉 There is **NO separate -0**
👉 Both +0 and -0 collapse into **ONE zero**

🔑 **THIS is what "only one zero" means**

- **1's complement** → two ways to write zero ❌
- **2's complement** → only one way to write zero ✅

🧠 **One-line understanding (memorize this)**

In 2's complement, there is no -0.
Only 00000000 represents zero.

❓ **Does computer always use 2's complement?**

✅ **Yes. All modern computers and Python use 2's complement**, exactly because:

- one zero only
- simpler arithmetic
- simpler hardware

Summary:

1's complement means:

Flip every bit

- 0 → 1
- 1 → 0

2's complement means:

1's complement + 1