International Journal of Mathematics Trends and Technology
ISSN: 2231-5373/ https://doi.org/10.14445/22315373/IJMTT-V71I8P108

Volume 71 Issue 8, 49-53, August 2025
© 2025 Seventh Sense Research Group®

*Original Article*

# An Algorithm for the XOR Operation in Python Programming

Sanjay M Chinchole

*Department of Mathematics, Arts, Science and Commerce College, Manmad, Maharashtra, India.*

*Corresponding Author : mr.sanjaymaths@gmail.com*

Received: 21 June 2025 Revised: 29 July 2025 Accepted: 16 August 2025 Published: 31 August 2025

***Abstract*** *- Bitwise operators are fundamental tools used in computer science programming. The Exclusive OR operator is one of the bitwise operators. It is a binary operator, also known as the XOR operator, which plays a vital role in computer programming. When the XOR operator operates on two different input bits, it results in a value 'true' (or '1'). Otherwise, it results in a value 'false' (or '0'). It has been helpful in many applications such as error detection, cryptography, digital circuit design, etc. In Python, the logical operation XOR compares two Boolean values. It results in '1' if exactly one of the two binary inputs is '1' and results in '0' otherwise. The caret symbol (^) is used to denote this operation in Python. The XOR bitwise operation can be performed on two integers. This article explains the mechanism of the XOR operator and presents a simple algorithm to manually compute the logical XOR of two variables in Python.*

***Keywords*** *- Decimal to binary conversion, Bitwise operator, Cryptography, Python programming, Algorithm.*

## 1. Introduction

Python math module [2,3] provides an encyclopedic set of mathematical functions commonly used in scientific calculations and software engineering solutions. The mathematical modules in the Python library are specially developed to enable most mathematical computations. These modules facilitate the mathematical functions [1,5] to tackle the basic operations such as addition (+), subtraction (-), multiplication (*), division (/), and the advanced operations, including trigonometric, logarithmic, and exponential functions. The operator module provides different functions that correspond to Python's built-in operators. For example, the expression $x + y$ correspond to the operator add. $(x, y)$.

Bitwise operators are fundamental tools used in computer science and digital logic. The Exclusive OR operator [4], known as the XOR operator, plays a vital role in computer programming. The XOR operator is a binary operator that results in a value 'true' (or '1') if the two input bits are different and results in a value 'false' (or '0') if the two input bits are the same. It has been helpful in many applications such as error detection, cryptography, and digital circuit design. In Python, the logical operation XOR is used to compare two Boolean values. It is true if exactly one of the two inputs is true, and false otherwise. This operation can be performed in Python using the caret symbol (^) for integers. This article explains the mechanism of the XOR operator and presents a simple algorithm to manually compute the logical XOR of two variables in Python.

## 2. Preliminaries

### *2.1. Decimal to Binary Conversion*

*2.1.1. Algorithm*

The XOR operation on two positive integers in the decimal form mainly operates on their binary forms. The transformation of a number [3] in decimal form (base-10 numeral system) to binary form (base-2 numeral system) is the process that converts a number in decimal form (which uses digits 0 through 9) to the number in binary form (which uses digits 0 and 1 only). The conversion of decimal to binary is necessary in computer science, as digital systems operate with binary representations. The standard method for converting a decimal number to binary is repeated division by 2 to the quotients and recording the remainder at each step. By writing the remainders obtained in this process in reverse order, the binary conversion of the given number in decimal form results. This conversion can be performed manually or using any mathematical software.

The algorithm [3] to convert a number in decimal form into binary form is as follows:

I) Initial Step:

- Take the decimal number you wish to convert.
- Create an empty list to save the remainders, which are binary digits 0 or 1.

II) Divide and save the remainders:

- Divide the given (initial) decimal number by 2.
- Find the remainder (which is either 0 or 1) of the division and store it in the list.
- Replace the decimal number with the integer quotient of the division.

III) Repeat the steps:

- Repeat step II) until the quotient becomes 0.

IV) Reverse the list of remainders:

- Write the remainders in the list of stored remainders in reverse order. This is the binary representation of the given decimal number.

*2.1.2. Illustration*

Example: Convert the decimal number 29 into a binary number.

The decimal number 29 can be converted into binary form by applying the above algorithm. The process of division by 2 and a record of remainders in each step is shown below:

**Table 1. The process of division by 2**

| Number | Quotient | Remainder |
|---|---|---|
| 29 | 14 | 1 |
| 14 | 7 | 0 |
| 7 | 3 | 1 |
| 3 | 1 | 1 |
| 1 | 0 | 1 |

After writing the remainders in reverse order, the binary representation obtained is 11101.
Therefore, the binary number equivalent to the decimal number 29 is 11101.

***2.2. The Bitwise XOR Operator in Python***

The caret symbol (^) is used to denote the bitwise XOR operator in Python. This operator is also known as the Exclusive OR operator. It performs a binary XOR operation on two integers. The result of the binary XOR operation is again an integer. Being a bitwise operator, XOR compares the individual bits of two integers after converting them into their binary representations. The operation results in a binary digit '1' in each bit position when the corresponding bits of the inputs differ and a binary digit '0' when they are the same. The result in binary form is then converted back into decimal form as an integer. The truth table for the bitwise XOR (Exclusive OR) operation on two binary digits '$a$' and '$b$' is shown below:

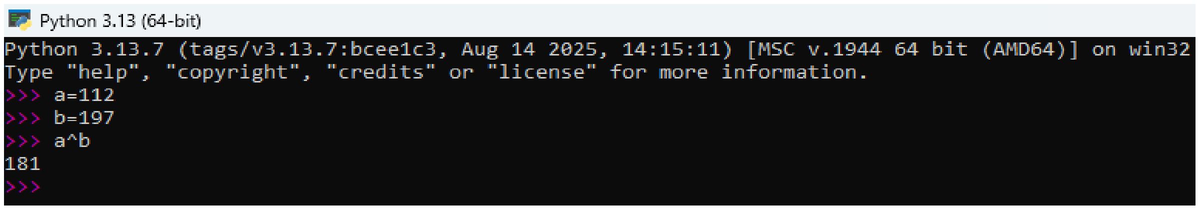
**Table 2. The truth table for XOR (binary Bitwise exclusive or) of $a$ and $b$**

| $a$ | $b$ | $a$ ^ $b$ |
|---|---|---|
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

The use of the caret symbol '^' to perform the bitwise XOR operation between two positive integer variables in Python is illustrated in the following subsection.

*2.2.1. Illustration*

Example: Using the bitwise XOR operator '^', the XOR of two positive integers, 112 and 197, in Python can be obtained as below:



```
Python 3.13.7 (tags/v3.13.7:bcee1c3, Aug 14 2025, 14:15:11) [MSC v.1944 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a=112
>>> b=197
>>> a^b
181
>>>
```

**Fig. 1 Output of 112 ^ 197 using Python programming software**

The actual procedure for finding the XOR of 112 and 197 is shown below:

- Convert both positive integers to binary:

112 (decimal) = 01110000 (binary)
197 (decimal) = 11000101 (binary)

- Perform bitwise XOR operation (compare each bit):

```
     01110000    (112)
^
     11000101    (197)
----------------------------------
     10110101   (Result in binary)
```

result $= a \wedge b$

- Convert the result in binary to decimal:

10110101 in binary
$= 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$= 128 + 0 + 32 + 16 + 0 + 4 + 0 + 1 = 181.$
Therefore, the XOR of 112 and 197 is 181.

## 3. Algorithm for XOR Operation in Python

### *3.1. Algorithm for XOR*

Objective: To compute the bitwise XOR of two positive integers using Python.

I) Initial Step:
- Let the two positive integers (in decimal form) be $a_0 = a$ and $b_0 = b$, whose XOR is to be computed.
- Create an empty list to store the assigned binary digits.

II) Observe and assign the binary digits:
- If $a_0 = a$ and $b_0 = b$ are of the same parity (both even or both odd), assign 0; otherwise, assign 1.
- Store the assigned number in the list.
- Divide both $a$ and $b$ by 2.
- Replace the decimal numbers with the greatest integer values. $a_i = \left[\frac{a_{i-1}}{2}\right]$ and $b_i = \left[\frac{b_{i-1}}{2}\right]$ respectively.

III) Repeat:
- Repeat step II) until both the updated decimal numbers reduce to 0.

IV) Reverse the list of binary digits:
- Write the numbers stored in the list in reverse order to obtain the binary representation of the decimal number $a$^$b$.

V) Convert into the decimal form:
- Convert the binary representation of $a$^$b$ into its decimal form to obtain the decimal value of $a$^$b$.

### *3.2. Key Points*

This algorithm works because the two decimal numbers have the same binary digit in the least significant bit (LSB) of their binary representations if and only if they have the same parity (both even or both odd). However, two decimal numbers have different binary digits in the least significant bit (LSB) of their binary representations if and only if they have different parities (one odd and the other even). Also, the greatest integer value of half of any decimal number is equal to the quotient obtained by dividing the integer by 2.

Further, the exclusive OR (XOR) operation compares corresponding bits of two binary numbers and returns ‘1’ if the bits are different and ‘0’ if they are the same. By processing each bit individually, starting from the least significant bit and recording results in reverse order, the algorithm accurately indicates the behavior of the bitwise XOR operator. Finally, the correct XOR value of the input positive decimal integers is obtained by converting the resulting sequence of binary digits into a decimal number.

## 4. Numerical Method

### *4.1. Numerical Method for XOR*

The numerical method involves repeatedly dividing the decimal numbers by 2, determining their integral (quotient) values, and recording the binary numbers in each step according to their parity (1 for the same parity and 0 for different parity). This process continues until the quotients become zero. The binary equivalent of their XOR is then obtained by reading the binary numbers in reverse order. The procedure is detailed in the table below.

**Table 3. The computation of the XOR (binary Bitwise exclusive or) of *a* and *b***

| Step i | $a_i$ | $b_i$ | Parity of $a_i$ and $b_i$ | LSB |
|---|---|---|---|---|
| 0 | $a_0 = a$ | $b_0 = b$ | same/different | 0/1 |
| 1 | $a_1 = \left\lfloor\frac{a_0}{2}\right\rfloor$ | $b_1 = \left\lfloor\frac{b_0}{2}\right\rfloor$ | same/different | 0/1 |
| 2 | $a_2 = \left\lfloor\frac{a_1}{2}\right\rfloor$ | $b_2 = \left\lfloor\frac{b_1}{2}\right\rfloor$ | same/different | 0/1 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| *N-1* | $a_{N-1} = \left\lfloor\frac{a_{N-2}}{2}\right\rfloor$ | $b_{N-1} = \left\lfloor\frac{b_{N-2}}{2}\right\rfloor$ | same/different | 0 |
| *N* | $a_N = \left\lfloor\frac{a_{N-1}}{2}\right\rfloor = 0$ | $b_N = \left\lfloor\frac{b_{N-1}}{2}\right\rfloor = 0$ | Process ends | |

Writing the least significant bits in reverse order results in the binary representation of the XOR of $a$ and $b$, which, when converted to decimal form, gives the final value of $a$ ^ $b$.

### *4.2. Numerical Method for XOR*

Example: To find the XOR of two positive integers, 112 and 197:

**Table 4. The computation of the XOR (binary Bitwise exclusive or) of *a* and *b***

| Step i | $a_i$ | $b_i$ | Parity of $a_i$and $b_i$ | LSB |
|---|---|---|---|---|
| 0 | $a_0 = 112$ | $b_0 = 197$ | Different | 1 |
| 1 | $a_1 = \left\lfloor\frac{112}{2}\right\rfloor = 56$ | $b_1 = \left\lfloor\frac{197}{2}\right\rfloor = 98$ | Same | 0 |
| 2 | $a_2 = \left\lfloor\frac{56}{2}\right\rfloor = 28$ | $b_2 = \left\lfloor\frac{98}{2}\right\rfloor = 49$ | Different | 1 |
| 3 | $a_3 = \left\lfloor\frac{28}{2}\right\rfloor = 14$ | $b_3 = \left\lfloor\frac{49}{2}\right\rfloor = 24$ | Same | 0 |
| 4 | $a_4 = \left\lfloor\frac{14}{2}\right\rfloor = 7$ | $b_4 = \left\lfloor\frac{24}{2}\right\rfloor = 12$ | Different | 1 |
| 5 | $a_5 = \left\lfloor\frac{7}{2}\right\rfloor = 3$ | $b_5 = \left\lfloor\frac{12}{2}\right\rfloor = 6$ | Different | 1 |
| 6 | $a_6 = \left\lfloor\frac{3}{2}\right\rfloor = 1$ | $b_6 = \left\lfloor\frac{6}{2}\right\rfloor = 3$ | Same | 0 |
| 7 | $a_7 = \left\lfloor\frac{1}{2}\right\rfloor = 0$ | $b_7 = \left\lfloor\frac{3}{2}\right\rfloor = 1$ | Different | 1 |
| 8 | $a_8 = \left\lfloor\frac{0}{2}\right\rfloor = 0$ | $b_8 = \left\lfloor\frac{1}{2}\right\rfloor = 0$ | Process ends | |

The process ends when both values are. $a_i$ and $b_i$ They have been reduced to zero, at which point the final XOR result was obtained. Writing the least significant bits in reverse order results in the binary representation. 10110101 of the XOR of a = 112 and b = 197, which, when converted to decimal form, gives the final value of $a$ ^ $b$.

Thus, $a$ ^ $b$ in binary
$= 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$= 128 + 0 + 32 + 16 + 0 + 4 + 0 + 1$
$= 181.$
In this way, the value 112 ^ 197 is obtained manually, which matches the result produced by the Python program.

## 5. Conclusion

The method explained in this study clearly illustrates the concept of the exclusive OR (XOR) operation in Python programming. It is both well-defined and one of the simplest manual techniques for obtaining the result of an XOR operation without relying on Python or any programming tools.

## References

[1] M.F. Sanner, "Python: A Programming Language for Software Integration and Development," *Journal of Molecular Graphics and Modelling*, vol. 17, no. 1, pp. 57-61, 1999. [Google Scholar] [Publisher Link]

[2] A. Bogdanchikov, M. Zhaparov, and R. Suliyev, "Python to Learn Programming," *Journal of Physics: Conference Series*, vol. 423, no. 1, 2013. [CrossRef] [Google Scholar] [Publisher Link]

[3] Anil K. Maini, *Digital Electronics: Principles, Devices and Applications*, John Wiley & Sons, 2007. [Google Scholar] [Publisher Link]

[4] Allen Downey, *Think Python: How to Think Like a Computer Scientist (Version 2.4. 0)*, Green Tea Press, 2015. [Google Scholar] [Publisher Link]

[5] Rydhm Beri, *Python Made Simple: Learn Python Programming in Easy Steps with Examples*, 2019. [Google Scholar] [Publisher Link]