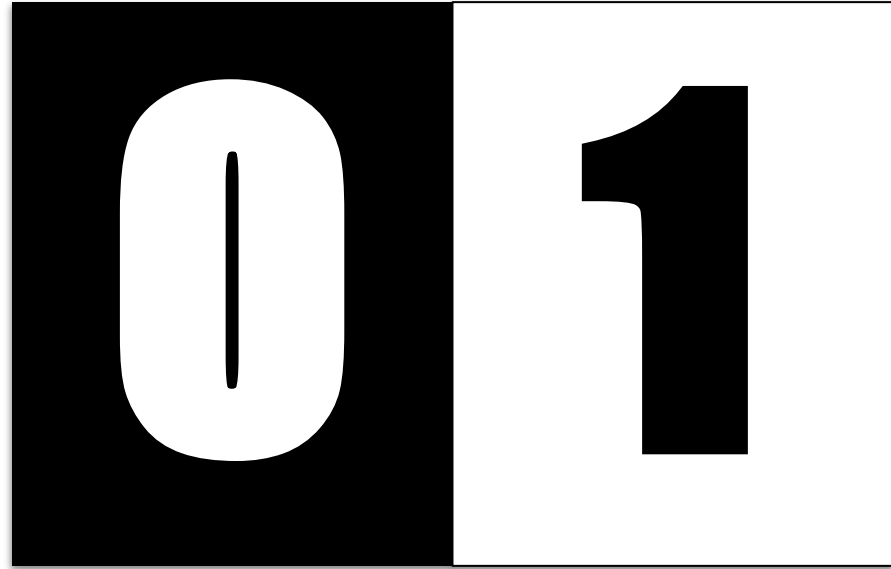


Bitwise Operations



Lecture Flow

- Pre-requisites
- Definition of Bits
- Bit representation of integers
- Bitwise operators
- Bitwise operators properties
- Bit Masking
- Bit Masking operations
- Python's built in functions
- Practice questions
- Quote of the day



Pre-requisites

No prerequisites folks





01
10

What are Bits?





Bit is the smallest unit of data in a computer system and represents a **binary digit**, which can have one of two values:
0 or 1.

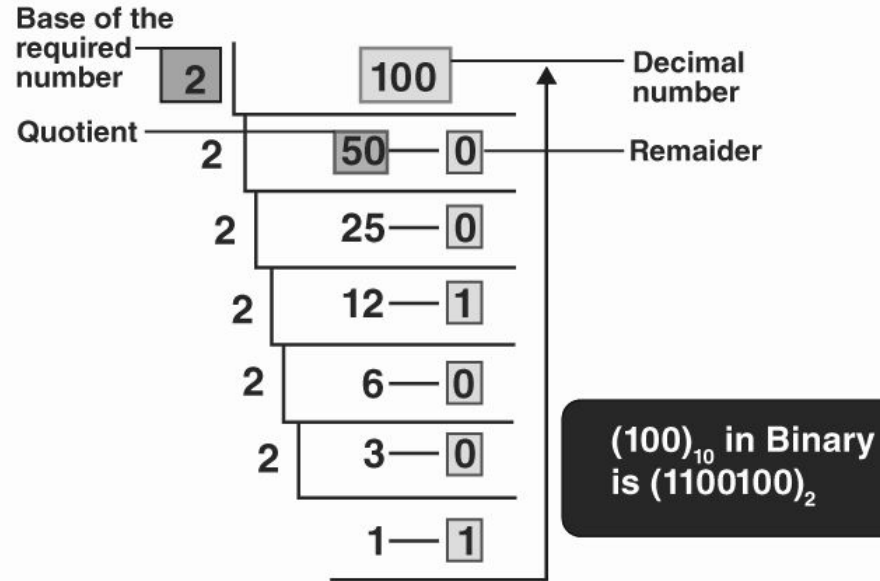


01
10

Bit Representation for unsigned?



Bit Representation



Binary

Decimal

MSB

LSB

00000000 = 000

$2^7 \ 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$
2 2 2 2 2 2 2 2

$10^2 \ 10^1 \ 10^0$
10 10 10

0 + 0 + 0 + 0 + 0 + 0 + 0 + 0

= 0 + 0 + 0



01
10

Bit Representation for signed?



Bit Representation

Signed Integer

$\begin{array}{c|c|c} \text{+/-} & & \text{number} \\ \hline 1 & 000000 & 1 \end{array}$

01
10

2's Complement

2's Complement

- A method to represent negative numbers
- Most popular method
- Allows adding negative numbers with the same logic gates as positive numbers
- Main Idea: $x + (-x) = 0$

2's Complement

How to convert number to 2's complement

1. Convert the positive number to binary
2. Flip the bits (1 to 0 and 0 to 1)
3. Add 1

Two's complement binary	Decimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8



Why Bits?

- Some questions require bit manipulations.
- It can be used to optimize solutions and simplify solutions.

01
10

Bit Operators

Bit Operators

- NOT (~)
- AND (&)
- OR (|)
- XOR (^)
- Bit Shifts (<<,>>)

NOT

0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

NOT (~)

AND

1	0	0	1	1	1	0	0
---	---	---	---	---	---	---	---

AND (&)

0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

Check yourself

$$3 \& 2 = ?$$

OR

1	0	0	1	1	1	0	0
---	---	---	---	---	---	---	---

OR (|)

0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

Check yourself

$$7 \mid 8 = ?$$

XOR

1	0	0	1	1	1	0	0
---	---	---	---	---	---	---	---

XOR (^)

0	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

Check yourself

$$4 \wedge 9 = ?$$

Left Shift



Left Shift (<<)

Check yourself

$$1 \ll 3 = ?$$

Right Shift



Right Shift (>>)

Check yourself

$$16 \gg 3 = ?$$

- Each right shift operation reduces the number to its half.

Bit Operators

A	B	$A \mid B$	$A \& B$	$A \wedge B$	$\sim A$
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

01
10



Checkpoint

Question #1

First let's solve it using normal approach.

Then let's solve it using bits (shifting operation)

338. Counting Bits

Hint 

Easy



8.6K

408



 Companies

Given an integer `n`, return an array `ans` of length `n + 1` such that for each `i` ($0 \leq i \leq n$), `ans[i]` is the **number of 1's** in the binary representation of `i`.

Example 1:

Input: `n = 2`

Output: `[0,1,1]`

Explanation:

`0 --> 0`

`1 --> 1`

`2 --> 10`

Solution

```
class Solution:
    def countBits(self, n: int) -> List[int]:
        bitCounts = []
        for i in range(n+1):
            count = 0
            while i > 0:
                if i & 1 == 1:
                    count += 1
                i >>= 1
            bitCounts.append(count)
        return bitCounts
```


Bitwise operators properties

- Commutative
 - $x \wedge y = y \wedge x$
- Associative
 - $x \& (y \& z) = (x \& y) \& z$
- Do these properties hold for all bit operators?

More Bitwise Operator Properties

- Identity
 - XOR: $x \wedge 0 = x$
 - What about the other operators?
- Inverse
 - XOR: $x \wedge x = 0$
 - What about the other operators?

01
10



Checkpoint

Question #2

First let's solve it using normal approach.

Then let's solve it using bits.

268. Missing Number

Easy



7834



3002



Add to List



Share

Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return *the only number in the range that is missing from the array*.

Example 1:

Input: `nums = [3,0,1]`

Output: 2

Explanation: `n = 3` since there are 3 numbers, so all numbers are in the range `[0,3]`. 2 is the missing number in the range since it does not appear in `nums`.

Example 2:

Input: `nums = [0,1]`

Output: 2

Explanation: `n = 2` since there are 2 numbers, so all numbers are in the range `[0,2]`. 2 is the missing number in the range since it does not appear in `nums`.

Example 3:

Input: `nums = [9,6,4,2,3,5,7,0,1]`

Output: 8

Explanation: `n = 9` since there are 9 numbers, so all numbers are in the range `[0,9]`. 8 is the missing number in the range since it does not appear in `nums`.

Solution

```
def missing_number(nums):  
    arr_xor = 0  
    range_xor = 0  
  
    for idx, num in enumerate(nums):  
        arr_xor ^= num  
        range_xor ^= idx + 1  
  
    return range_xor ^ arr_xor
```

```
def missing_number(nums):  
    N = len(nums)  
    range_sum = N * (N + 1) // 2  
    arr_sum = 0  
  
    for num in nums:  
        arr_sum += num  
  
    return range_sum - arr_sum
```

01
10

Bit masking



Bit masking

- Way of optimizing storage
- Store information in a single bit



01
10

Test 5th Bit

num = 10001000**1**00111

1 = 0000000000000001



Test 5th Bit

num = 10001000100111

1 = 000000000000001

1<<5 = 00000000100000

& 10001000100111
00000000100000

00000000100000 (!= 0)



Implement



Bit masking operations

Test k^{th} bit is set: `num & (1 << k) != 0` .

Set k^{th} bit: `num |= (1 << k)` .

Turn off k^{th} bit: `num &= ~(1 << k)` .

Toggle the k^{th} bit: `num ^= (1 << k)` .

01
10



Checkpoint

46. Permutations



Medium



15.1K



256



Companies

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

Input: `nums = [1,2,3]`

Output: `[[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]`

Example 2:

Input: `nums = [0,1]`

Output: `[[0,1], [1,0]]`

Example 3:

Input: `nums = [1]`

Output: `[[1]]`

Question #3

- Use bit mask to keep track of used numbers

Solution

```
def permute(self, nums: List[int]) -> List[List[int]]:
    ans, curr = [], []
    used = 0

    def build(i=0):
        nonlocal used
        if i == len(nums):
            ans.append(curr.copy())
            return

        for idx, num in enumerate(nums):
            if (used & (1 << idx)) == 0:
                used ^= 1 << idx
                curr.append(num)
                build(i+1)
                curr.pop()
                used ^= 1 << idx

    build()
    return ans
```

01
10



Python Built-in Functions

Bit masking

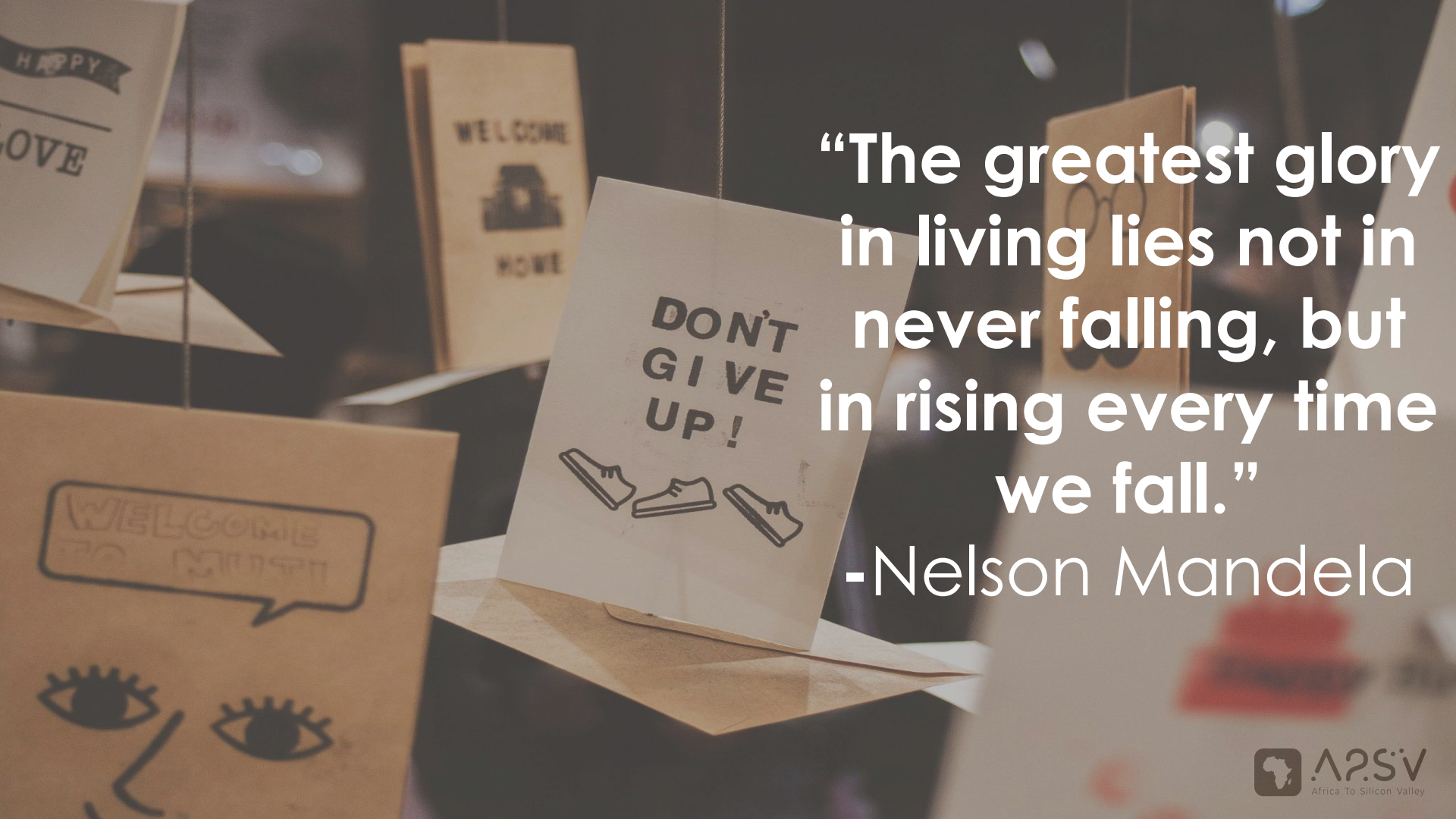
- `bin()`: This built-in function can be used to convert an integer to a binary string.
- `int()`: This built-in function can be used to convert a binary string to an integer.

Bit masking

- `bit_length()`: This method can be called on an integer and returns the number of bits required to represent the integer in binary, excluding the sign bit.
- `bit_count()` : this method can be called on an integer and returns the number of set bits.

Practice Problems

- [Number Complement](#)
- [Hamming Distance](#)
- [Cirno's Perfect Bitmasks Classroom](#)
- [Subsets](#)
- [Add Binary](#)
- [Single Number II](#)
- [Single Number III](#)
- [Sum of Two Integers](#)
- [Maximum Product of Word Lengths](#)

The background of the image is a collection of several paper cards hanging from thin strings. One card in the foreground has the text 'DON'T GIVE UP!' and three line drawings of sneakers. Another card to the left has a speech bubble saying 'WELCOME TO MUSA' and a drawing of a face with large eyes. Other cards in the background have 'HAPPY LOVE', 'WELCOME HOME', and 'WELCOME' written on them.

**“The greatest glory
in living lies not in
never falling, but
in rising every time
we fall.”**

-Nelson Mandela