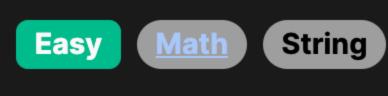
## 168. Excel Sheet Column Title



## Problem Description

seen in an Excel sheet. In Excel, columns are labeled alphabetically from A to Z, then continuing with AA, AB, etc., after Z. This sequence creates a pattern similar to base-26 numeral system, except that it doesn't contain a zero and it starts from A (1) rather than zero (0).

In this pattern, A corresponds to 1, B to 2, and so on up to Z which corresponds to 26. Then, the sequence continues with two-

The given problem requires us to convert an integer, columnNumber, into a string that represents the corresponding column title as

letter combinations such as AA for 27, AB for 28, and so on. Unlike our decimal system, where the digit '0' represents zero and serves as a place-holder, there is no 'zero' character in the Excel column titles.

The challenge is to convert the given columnNumber into this special alphabet-based naming system. This includes figuring out

how many times we can divide the number by 26 to advance letters and how to correctly wrap around after reaching the end of the alphabet.

## To solve this problem, we can use a method similar to converting a number from base-10 to base-26, with the twist that Excel's

ntuition

to 27 (AA), rather than the standard 0, 1, ..., 24, 25 (A, B, ..., Y, Z) and then wrap to 26 (AA).

Here's the approach step-by-step:

1. Initialize an empty list res to accumulate the characters (from the least important digit to the most important one).

column naming scheme is 1-indexed, not 0-indexed. This means the sequence goes 1, 2, ..., 25, 26 (A, B, ..., Y, Z) and then wraps

2. Enter a loop that will continue as long as columnNumber is greater than zero. This iterative process will divide the columnNumber until it cannot be divided anymore, indicating completion.

corresponding to the letters from A to Z.

the characters together to form a string.

detailed walk-through of the implementation steps.

the number from base-10 to the equivalent 'base-26'.

list res back to the correct order.

Enter the while loop since columnNumber > 0.

1-indexed column naming system.

string.

- 3. Inside the loop, since the numbering is 1-indexed, we decrease columnNumber by 1 to convert it to a 0-indexed number.
- 4. Obtain the remainder of the current columnNumber when divided by 26. This modulo operation gives us an index

Prepare for the next iteration by dividing columnNumber by 26 since we have already handled one "digit" of our base-26

- 5. Convert this index into a character by adding it to the ASCII value of 'A' and append the resulting character to the list res.
- number.

  7. Once the loop is done, we reverse res because we have been appending the less significant characters first, and then we join
- 8. Return the resulting string, which is the column title that corresponds to the original columnNumber.
- By repeatedly taking modulus and division, we handle the number 'column by column', just like an actual base conversion, but always taking into consideration the offset caused by the lack of 'zero' in the Excel system. This approach keeps us within Excel's

The implementation of the solution involves a straightforward approach that resembles a base-conversion algorithm. Here's a

Create a class Solution with a method convertToTitle, which takes an integer columnNumber as its parameter and returns a

Solution Approach

want. Use chr() to get the character from this ASCII value and append it to our list res.

The method convertToTitle returns this string, completing the conversion process.

Subtract 1 from columnNumber to make it 0-indexed: columnNumber = 705 - 1 = 704.

Inside the convertToTitle method, initialize an empty list res which will store the characters of our result in reverse order (the least significant 'digit' first).
 Use a while loop to process the columnNumber as long as it's greater than 0. This loop will be used to iteratively break down

- 4. As Excel columns start from 1 (A) instead of 0, we subtract 1 from columnNumber each iteration to properly align our values with the index for character mapping.
- least significant 'digit' of the number.

  3. Use ord('A') to get the ASCII value of 'A', then add the remainder to it. This will give us the ASCII value of the character we

Calculate the remainder with columnNumber % 26 to determine which character in the range A-Z corresponds to the current

subsequent 'digit'.

8. Once the while loop concludes, we have all the characters of our column title in reverse order. We use [::1] to reverse the

Prepare for the next iteration by performing integer division columnNumber //= 26 to reduce our columnNumber for the

- 9. Finally, we join the list of characters with join() into a string, which represents the column title equivalent to the initial columnNumber.
- manipulation capabilities.

  Example Walkthrough

This solution doesn't use any complex libraries or specific data structures beyond a basic list and standard Python string

spreadsheet 1-indexed base. It is an elegant and efficient solution that works within the confines of Python's simple data

operations. The pattern here mirrors a familiar concept of converting between numeral systems, accommodating the unique Excel

Let's walk through an example to illustrate the solution approach by converting the columnNumber 705 to its corresponding Excel

column title.

1. We begin with columnNumber = 705. Our result res is currently an empty list.

## 26 = 704 % 26 = 24.

• columnNumber = 27 - 1 = 26

columnNumber = 26 // 26 = 1

 $\circ$  columnNumber = 1 - 1 = 0

remainder = 0 % 26 = 0

Solution Implementation

**Python** 

Java

class Solution {

class Solution:

columnNumber = 0 // 26 = 0

We loop again:

a character with chr(): character = chr(ord('A') + remainder) = chr(65 + 24) = 'Y'. Append 'Y' to res.

6. Update columnNumber for the next iteration: columnNumber //= 26 = 704 // 26 = 27.

Calculate the remainder of columnNumber divided by 26 to determine the character for this 'digit': remainder = columnNumber %

Convert this remainder to a character by finding the ASCII value of 'A', adding the remainder to it, and then converting back to

remainder = 26 % 26 = 0
 'A' + 0 = 'A', append 'A' to res. Now, res = ['Y', 'A'].

As columnNumber is still greater than 0, we repeat steps 3-6:

'A' + 0 = 'A', append 'A' to res. Now, res = ['Y', 'A', 'A'].

def convertToTitle(self, column\_number: int) -> str:

column number -= 1

column\_number //= 26

public String convertToTitle(int columnNumber) {

std::string convertToTitle(int columnNumber) {

int remainder = columnNumber % 26;

// std::reverse(title.begin(), title.end());

// Return the column title in the correct format.

title.insert(title.begin(), 'A' + remainder);

std::string title;

while (columnNumber > 0) {

columnNumber /= 26;

return title;

columnNumber--;

// StringBuilder to build the result in reverse order

# Initialize an empty list to store the characters that will

9. The loop ends because columnNumber is now 0.

10. Reverse res and join the characters to form a string: res.reverse() = ['A', 'A', 'Y'], result = ".join(res) = "AAY".

Return the result, "AAY", which is the corresponding Excel column title for column number 705.

We have converted the input columnNumber 705 into the Excel column title "AAY" following the solution steps detailed above.

# represent the column title in reverse order.
title\_chars = []

# Continue to iterate as long as the column\_number is greater than zero.
while column\_number:

# Calculate the character that corresponds to the current modulo of

# Update column\_number by dividing it by 26, for the next iteration

# as we move to the next most significant digit in the title.

# column\_number and 'A'. Append the character to the title\_chars list.

# Adjust column\_number to zero-indexed for modulo operation.

title\_chars.append(chr(ord('A') + column\_number % 26))

```
# Since the title_chars list was constructed backwards,
# reverse it and join the characters into a single string
# to form the column title, then return the result.
return ''.join(title_chars[::-1])
```

// This method converts a given column number to its corresponding Excel column title.

```
StringBuilder result = new StringBuilder();
        // Loop until the columnNumber is 0
       while (columnNumber > 0) {
            // Decrement columnNumber to adjust for 1-indexing in Excel columns
            columnNumber--;
            // Calculate the character to append using the remainder
            // and concatenate it to our result string
            result.append((char) ('A' + columnNumber % 26));
            // Update columnNumber by dividing by 26 for the next iteration
            columnNumber /= 26;
       // Reverse the result since we've been appending characters from least significant (right most) to most significant
        return result.reverse().toString();
C++
#include <string>
#include <algorithm> // for std::reverse
// This function converts a given column number to a title as it would appear in an Excel sheet.
// For example, 1 becomes 'A', 27 becomes 'AA', and so on.
```

// Initialize a string to hold the characters of the final result.

// Get the remainder when columnNumber is divided by 26 to find

// Convert the remainder to the corresponding uppercase character

// by adding it to 'A', then prepend it to the 'title' string.

// the character that corresponds to the current place value.

// Decrement to map the number to a zero-indexed scale where A=0, B=1, ..., Z=25.

// The ASCII value of 'A' is 65, but adding remainder to 'A' gives the correct character.

// Go to the next place value by dividing columnNumber by 26 before the next iteration.

// The characters are inserted in reverse order, so we must reverse the final string.

// This can be done during insertion as in the line above or by reversing the string here.

// Continue the loop until the columnNumber is reduced to 0.

```
TypeScript
  // This function converts a given column number to a title as it would appear in an Excel sheet.
  // For example, 1 becomes 'A', 27 becomes 'AA' and so on.
  function convertToTitle(columnNumber: number): string {
      // Initialize an array to hold the characters of the final result.
      let titleCharacters: string[] = [];
      // Continue the loop until the columnNumber is reduced to 0.
      while (columnNumber > 0) {
          // Decrement to map the number to a zero-indexed scale where A=0, B=1, ..., Z=25.
          columnNumber--;
          // Get the remainder when columnNumber is divided by 26 to find
          // the character that corresponds to the current place value.
          let remainder: number = columnNumber % 26;
          // Convert the remainder to the corresponding ASCII uppercase character
          // and unshift it to the start of the titleCharacters array.
          // ASCII value of 'A' is 65, so adding remainder to it gives the correct character.
          titleCharacters.unshift(String.fromCharCode(remainder + 65));
          // Go to the next place value by dividing columnNumber by 26 before the next iteration.
          columnNumber = Math.floor(columnNumber / 26);
      // Join the titleCharacters array into a string to get the column title in the correct format.
      return titleCharacters.join('');
class Solution:
   def convertToTitle(self, column_number: int) -> str:
```

# Initialize an empty list to store the characters that will

# Continue to iterate as long as the column\_number is greater than zero.

# Calculate the character that corresponds to the current modulo of

# Update column\_number by dividing it by 26, for the next iteration

# as we move to the next most significant digit in the title.

# column\_number and 'A'. Append the character to the title\_chars list.

# Adjust column\_number to zero-indexed for modulo operation.

title\_chars.append(chr(ord('A') + column\_number % 26))

# represent the column title in reverse order.

```
column_number //= 26

# Since the title_chars list was constructed backwards,

# reverse it and join the characters into a single string

# to form the column title, then return the result.
```

Time and Space Complexity

return ''.join(title\_chars[::-1])

title\_chars = []

while column\_number:

column\_number -= 1

The time complexity of the function is  $0(\log_{26}(n))$ , where n is the columnNumber. This is because the loop divides the columnNumber by 26 in each iteration, effectively converting the number from base 10 to base 26. The number of iterations is proportional to the number of times n can be divided by 26 before it becomes 0, which is essentially the number of digits in the base-26 representation of n.

The space complexity of the function is  $0(\log_{26}(n))$  as well. The reason behind this is that space is allocated for the res list, which stores each character corresponding to a digit in the base-26 number. The length of the res list is equal to the number of digits in the base-26 representation of n, which results from the same logarithmic relationship as in the time complexity analysis.