

Problem Description

The given problem requires us to take an integer n and format it as a string that represents the number with a dot . inserted every three digits from the right. This is commonly known as adding a "thousand separator." For example, if n is 123456789, the output should be "123, 456, 789". If n is less than 1000, it should be returned without change, since there are no thousands to separate.

Intuition

significant digit). We can achieve that by continuously dividing the number by 10 and getting the remainder, which represents the current rightmost digit. Each time we extract a digit, we append it to a result list. We also need to keep a count of how many digits we've added so that we

To solve this problem, one intuitive approach is to process the integer digit by digit from right to left (least significant digit to most

know when to insert a dot. For this problem, we insert a dot every three digits. When there are no more digits left (i.e., the remaining number is 0), we stop the process. Finally, we reverse the result list since we built the number from right to left, join the elements to form a string, and return the formatted number. Here are the specific steps of the process:

Initialize a count cnt to 0. This will keep track of the number of digits processed.

- 2. Initialize an empty list ans to build the answer from individual digits and dots.
- 3. Enter a loop that will run until there are no more digits to process in the number n. Within the loop:
- Divide n by 10, separate the quotient and the remainder. The remainder is the current digit to add to the answer, while the quotient is the reduced number for the next iteration.
 - Convert the remainder to a string and append it to ans. Increment the count cnt.
 - If cnt is equal to 3 and there are still digits in n (i.e., n is not 0), append a . to ans and reset cnt to 0.
- 4. Break the loop when n is reduced to 0.
- 5. Reverse the list ans since we built it backwards.
- 6. Join the elements in ans to form the final string.
- 7. Return the resulting string.

following this numeric processing method:

Solution Approach

This method ensures that we are adding a dot every three digits while taking the basic rules of string formatting into account.

The solution to this problem involves a straightforward implementation of the intuition described. The solution uses simple arithmetic operations and a list as the primary data structure to build the answer. No particular algorithmic pattern is necessary other than

1. The solution class Solution contains the method thousandSeparator, which takes an integer n as an argument and returns a string with the formatted number.

2. A counter cnt is initialized to 0. This counter is used to track the number of digits added to the answer list ans since the last dot

- was added (or since the beginning if no dot has been added yet). 3. An empty list ans is created to accumulate the digits and dots in reverse order, as we will be processing the digits from least
- significant to most significant. 4. The while loop while 1: ensures that the loop continues until explicitly broken.

5. Inside the loop, n, v = div mod(n, 10) divides n by 10, storing the quotient back in n for the next iteration and the remainder in v.

7. The counter cnt is incremented by 1 each time a digit is added to ans.

6. ans. append(str(v)) adds the current digit to the answer list as a string, since we want the final output to be a string.

The remainder represents the current least significant digit to be added to the ans list.

- 8. A conditional if n == 0: checks whether the number n has been fully processed. If n is 0, the loop is terminated by executing a
- '.' is appended to ans, and cnt is reset to 0 to count the next three digits. 10. Once the loop is broken, return ''.join(ans[::-1]) is executed. This joins the elements of ans together into a string, and

9. Another conditional if cnt == 3: checks if three digits have been added to ans since the last dot or since the start. If true, a dot

By maintaining a counter and using the divmod function to split the integer into digits, the approach avoids string-to-integer conversions except when appending digits to the answer list. The use of a list to construct the answer in reverse order helps

ans [::-1] reverses the list since we built the number from the least significant digit to the most significant.

converted into a list of characters, the reversal and join operations form the final string to be returned.

Example Walkthrough Let's walk through the solution with a small example. Supposing the input integer n is 12345. We need to format it as a string with a

efficiently build the result since lists in Python have a time complexity of O(1) for typical append operations. Once the number is fully

break.

1. Initialize the counter cnt to 0 and the list ans to an empty list. 2. Enter the loop since n is non-zero.

4. In the second iteration, now n = 123 and v = 4. We append '4' to ans and increment cnt by 1.

dot inserted every three digits from the right. So, our expected output is "12.345".

Here are the steps demonstrating how the algorithm would process this input:

5. In the third iteration, n = 12 and v = 3. We append '3' to ans. cnt is now 3, so we append a '.' to ans and reset cnt to 0.

6. In the fourth iteration, n = 1 and v = 2. We append '2' to ans and increment cnt by 1.

7. In the fifth iteration, n = 0 and v = 1. We append '1' to ans. Since n is now 0, we break out of the loop.

3. In the first iteration, divmod(n, 10) gives us n = 1234 and v = 5. We append '5' to ans and increment cnt by 1.

8. Now ans is ['5', '4', '3', '.', '2', '1']. We need to reverse the list to get the digits in the correct order. 9. After reversing, ans is ['1', '2', '.', '3', '4', '5'].

Note: If n had been smaller than 1000, say n = 12, the process would be the same without the insertion of a dot, resulting in "12".

- 11. We return the resulting string "12.345".
- **Python Solution** class Solution: def thousandSeparator(self, value: int) -> str:

10. Joining the elements in ans with '', we get the final result "12.345".

digit_counter = 0 # Initialize a list to build the answer incrementally result_parts = []

Loop until the entire number has been processed

result_parts.append(str(remainder))

Increment the digit counter

break

if (count == 3) {

formattedNumber.append('.');

// and convert it to a String before returning

* Function to add a thousand separator in the given integer.

function thousandSeparator(n: number): string {

* It inserts a period '.' every three digits from right to left.

* @param {number} n - The number in which the thousand separator must be added.

let count = 0;
// Initialize a counter to keep track of the number of digits.

* @returns {string} - The number as a string with thousand separators added.

return formattedNumber.reverse().toString();

count = 0; // Reset the counter after appending the dot

// Reverse the StringBuilder content to maintain the correct order

Initialize a counter to track the number of digits processed

Divide the number by 10 to get the next digit and the remainder 11 value, remainder = divmod(value, 10) 12 # Convert the remainder (a digit) to a string and append to the list 14

```
digit_counter += 1
18
20
               # Check if the number has been completely divided
                11 value == 0:
```

10

15

16

17

22

20

23

24

25

26

28

29

31

30 }

while True:

```
23
24
               # If three digits have been processed, insert a period and reset counter
               if digit_counter == 3:
26
                   result_parts.append('.')
27
                   digit_counter = 0
28
           # Since digits are processed in reverse order, reverse the list and join the parts into a string
29
           formatted_number = ''.join(result_parts[::-1])
30
31
32
           return formatted_number
33
Java Solution
   class Solution {
       // This method converts an integer to a string with dot separators for every three digits
       public String thousandSeparator(int number) {
           int count = 0; // Initialize a counter to keep track of every three digits
           StringBuilder formattedNumber = new StringBuilder(); // Use StringBuilder to efficiently manipulate strings
           // Loop until the entire number has been processed
           while (true) {
               int digit = number % 10; // Extract the last digit of the number
               number /= 10; // Remove the last digit from the number
10
               formattedNumber.append(digit); // Append the digit to the StringBuilder
11
12
               count++; // Increment the counter
13
               // If the number is reduced to zero, break out of the loop
14
               if (number == 0) {
16
                   break;
17
18
19
               // After every third digit, append a dot to the StringBuilder
```

```
C++ Solution
1 class Solution {
2 public:
       // Function to add a thousand separator in the given integer.
       // It inserts a period '.' every three digits from right to left
       string thousandSeparator(int n) {
           int count = 0;
                                     // Initialize a counter to keep track of the number of digits
           string formattedNumber;
                                     // Initialize an empty string to build the result
8
           // Proceed to iterate until the entire number has been processed
9
           do {
10
               int digit = n % 10;
                                     // Extract the rightmost digit
                                     // Remove the rightmost digit from the number
12
               n /= 10;
               formattedNumber += to_string(digit); // Append the digit to the result string
13
                                     // Increment the digit counter
14
               count++;
15
16
               // Insert a period after every third digit from the right, but only if more digits are left to process
               if (count == 3 && n != 0) {
17
                   formattedNumber += '.';
                                     // Reset the digit counter after inserting a period
19
                   count = 0;
20
                                     // Continue as long as there are digits left
21
           } while (n != 0);
22
           // Since the digits were added in reverse order, reverse the string to get the correct format
23
24
           reverse(formattedNumber.begin(), formattedNumber.end());
25
26
           return formattedNumber; // Return the properly formatted number
27
28 };
29
Typescript Solution
  /**
```

```
let formattedNumber = ''; // Initialize an empty string to build the result.
10
       // Proceed to iterate until the entire number has been processed.
12
       do {
13
           const digit = n % 10; // Extract the rightmost digit.
           n = Math.floor(n / 10); // Remove the rightmost digit from the number.
14
           formattedNumber += digit.toString(); // Append the digit to the result string.
15
                                   // Increment the digit counter.
           count++;
16
           // Insert a period after every third digit from the right, but only if more digits are left to process.
           if (count === 3 && n !== 0) {
19
20
               formattedNumber += '.';
21
                           // Reset the digit counter after inserting a period.
               count = 0;
22
       } while (n !== 0);
                                   // Continue as long as there are digits left.
24
25
       // Since the digits were added in reverse order, reverse the string to get the correct format.
26
       formattedNumber = reverseString(formattedNumber);
27
28
       return formattedNumber; // Return the properly formatted number.
29
30
31 /**
    * Helper function to reverse a string.
    * @param {string} str - The string to be reversed.
    * @returns {string} - The reversed string.
35
   function reverseString(str: string): string {
       return str.split('').reverse().join('');
37
38 }
39
Time and Space Complexity
The time complexity of the provided code is O(d), where d is the number of digits in the integer n. This is because the while loop runs
```

used is proportional to the number of digits d.

once for each digit of n until n becomes 0, performing a constant amount of work inside the loop for each digit (division, modulo, and counter operations). The space complexity is also 0(d), as the main additional space used is the list ans which stores each digit as a character. In the

worst case, for every three digits, there is an additional period character, resulting in d/3 period characters. Thus, the total space