1281. Subtract the Product and Sum of Digits of an Integer

Easy Math

Problem Description

Given an integer `n`, the task is to find the difference between two quantities related to the digits of `n`. These quantities are the product of all the digits of `n` and the sum of all the digits of `n`. To clarify, if `n` is `234`, the product of its digits is `2 * 3 * 4` which equals `24`, and the sum of its digits is `2 + 3 + 4` which equals `9`. The result to return for the problem would be the product minus the sum, thus `24 - 9 = 15`.

Intuition

To solve this problem, we can simulate the process of extracting each digit of the number `n`, and then perform the respective multiplication and addition operations with these digits.

Firstly, consider the need to handle each digit of the integer `n` separately. A convenient way to extract the digits of an integer is by repeatedly dividing the number by `10` and taking the remainder, which gives us the last digit of the current number. In each iteration, we remove this last digit by performing integer division by `10`.

With each digit extracted, we can directly calculate two running totals: one for the product of the digits, and another for the sum of the digits.

1. We initialize a variable (let's call it `x`) to `1` to hold the product (since the product of zero is zero, which would nullify all subsequent multiplications).
2. We initialize another variable (let's call it `y`) to `0` to hold the sum (since adding zero does not change the sum).
3. As we extract each digit `v`, we update `x` by multiplying it with `v` (`x = x * v`) and update `y` by adding `v` to it (`y = y + v`).
4. After we have finished processing all the digits of `n`, we calculate the difference between the product and the sum (`x - y`) and return this as our result.

This approach allows us to keep track of both the product and the sum of the digits of `n` in a single pass over the number, which is both efficient and straightforward.

Solution Approach

The solution follows a straightforward simulation approach. The goal is to iterate through each digit of the given number `n` and calculate both the product and the sum of these digits. To make it possible, we employ a basic `while` loop that continues to execute as long as the number `n` is greater than 0.

Below are the steps outlining the implementation details and the logical flow of the program:

1. We start by initializing two variables: `x` with a value of `1` to serve as the accumulator for the product, and `y` with a value of `0` to serve as the accumulator for the sum of the digits.
2. We enter a `while` loop that will run until `n` becomes `0`.
3. Inside the loop, we utilize the `divmod` function, which performs both integer division and modulo operation in a single step. The statement `n, v = divmod(n, 10)` divides `n` by `10`, with the quotient assigned back to `n` (essentially stripping off the last digit) and the remainder assigned to `v` (which is the last digit of the current `n`).
4. With the digit `v` isolated, we update our product variable `x` by multiplying it by `v` (since `x` keeps the running product of the digits).
5. Similarly, we update our sum variable `y` by adding `v` to it, keeping a running sum of the digits.
6. After the loop has completed (all digits of `n` have been processed), the variable `x` holds the product of the digits, and the variable `y` holds the sum of the digits of the initial `n`.
7. Lastly, we return the difference between these two values, `x - y`.

This algorithm makes use of very basic operators and control structures in Python and does not require the use of any complex data structures or patterns. The simplicity of the algorithm—iterating over each digit, maintaining running totals for the product and sum, and then returning the difference—illustrates the power of a linear implementation that performs the required operations in a single pass over the digits of the input number.

Example Walkthrough

Let's illustrate the solution approach using the integer `n = 523`.

1. Initialize two variables: `x` as `1` (for product) and `y` as `0` (for sum).
2. Begin a `while` loop; `n` is `523`, which is greater than `0`, so we proceed.
3. Inside the loop, we apply `divmod` operation: `n, v = divmod(523, 10)` which gives `n = 52` and `v = 3`.
4. Update `x` by multiplying it with `v`: `x = 1 * 3 = 3`.
5. Update `y` by adding `v`: `y = 0 + 3 = 3`.
6. Loop iterates again. Now `n` is `52`. Apply `divmod`: `n, v = divmod(52, 10)` which gives `n = 5` and `v = 2`.
7. Update `x`: `x = 3 * 2 = 6`.
8. Update `y`: `y = 3 + 2 = 5`.
9. Loop iterates again. Now `n` is `5`. Apply `divmod`: `n, v = divmod(5, 10)` which gives `n = 0` and `v = 5` (we have extracted the last digit).
10. Update `x`: `x = 6 * 5 = 30`.
11. Update `y`: `y = 5 + 5 = 10`.
12. Loop ends since `n` is now `0`.
13. We now have `x = 30` and `y = 10`. Calculate the difference: `x - y = 30 - 10 = 20`.
14. The result is `20`, which is the difference between the product of `523`'s digits and the sum of `523`'s digits.

The walkthrough demonstrates the simple, yet effective, process of using a loop to tear down the number into its constituent digits, calculate the running product and sum, and finally determine the difference between these two quantities.

Solution Implementation

Python

```python
class Solution:

    def subtractProductAndSum(self, n: int) -> int:
        # Initialize the product and sum variables
        product_of_digits = 1  # Holds the product of the digits
        sum_of_digits = 0      # Holds the sum of the digits

        # Loop through each digit of the number
        while n:
            # Divmod returns the quotient and the remainder
            # n becomes the quotient, and digit holds the remainder (current digit)
            n, digit = divmod(n, 10)

            # Multiply the current digit by the product
            product_of_digits *= digit

            # Add the current digit to the sum
            sum_of_digits += digit

        # Return the result of subtracting the sum from the product of digits
        return product_of_digits - sum_of_digits
```

Java

```java
class Solution {

    // Function to subtract the product of digits from the sum of digits of an integer
    public int subtractProductAndSum(int n) {
        // Initializing product to 1 as we will multiply digits with it
        int product = 1;
        // Initializing sum to 0 as we will add digits to it
        int sum = 0;

        // Looping through each digit of the number
        while (n > 0) {
            // Getting the last digit of the number
            int digit = n % 10;

            // Multiplying the current digit with the product
            product *= digit;
            // Adding the current digit to the sum
            sum += digit;

            // Removing the last digit from n
            n /= 10;
        }
        // Returning the difference between the product and the sum
        return product - sum;
    }
}
```

C++

```cpp
class Solution {
public:
    int subtractProductAndSum(int n) {
        int product = 1; // Initialize product to 1 since we are multiplying
        int sum = 0;     // Initialize sum to 0 since we are adding

        // Loop through each digit of the number
        while (n > 0) {
            int digit = n % 10; // Get the last digit
            product *= digit;   // Multiply the digit to the product
            sum += digit;       // Add the digit to the sum
            n /= 10;            // Remove the last digit from the number
        }

        // Return the difference between the product and sum of digits
        return product - sum;
    }
};
```

TypeScript

```typescript
// Function to calculate and return the difference between the product
// of digits and the sum of digits of the given number.
function subtractProductAndSum(n: number): number {
    // Initialize product to 1 and sum to 0
    let product = 1;
    let sum = 0;

    // Loop to process each digit in the number
    while (n > 0) {
        // Extract the last digit using modulo operation
        const digit = n % 10;

        // Multiply the product by the digit
        product *= digit;

        // Add the digit to the sum
        sum += digit;

        // Remove the last digit by dividing by 10 and flooring the result
        // to get an integer for the next iteration
        n = Math.floor(n / 10);
    }

    // Return the difference between product and sum
    return product - sum;
}
```

```python
class Solution:

    def subtractProductAndSum(self, n: int) -> int:
        # Initialize the product and sum variables
        product_of_digits = 1  # Holds the product of the digits
        sum_of_digits = 0      # Holds the sum of the digits

        # Loop through each digit of the number
        while n:
            # Divmod returns the quotient and the remainder
            # n becomes the quotient, and digit holds the remainder (current digit)
            n, digit = divmod(n, 10)

            # Multiply the current digit by the product
            product_of_digits *= digit

            # Add the current digit to the sum
            sum_of_digits += digit

        # Return the result of subtracting the sum from the product of digits
        return product_of_digits - sum_of_digits
```

Time and Space Complexity

The time complexity of the given code is `O(log n)` where `n` is the input integer. This is because the while loop runs once for each digit of `n`, and the number of digits in `n` is proportional to the logarithm of `n`.

The space complexity of the code is `O(1)`. This is constant space complexity because the algorithm only uses a fixed amount of additional space (variables `x` and `y`, and temporary variables for intermediate calculations like `n` and `v`) that does not scale with the input size `n`.