2703. Return Length of Arguments Passed

Easy

Leetcode Link

Problem Description

The problem requires the implementation of a function named `argumentsLength` that takes a variable number of arguments and returns the total count of these arguments. The arguments can be of any type, including numbers, strings, booleans, objects, etc. The main goal is to determine how many arguments are passed into the function when it is called.

Intuition

The intuition behind the solution comes from understanding what the rest parameters syntax `...args` in JavaScript (or TypeScript in this case) is used for. Rest parameters allow us to represent an indefinite number of arguments as an array. This makes it easier to handle a function parameter that can have any number of values.

By using rest parameters, the `argumentsLength` function can accept any number of arguments, and those arguments are gathered into an array. Then, the `length` property of the array is used to find out how many items (arguments) it contains. This property directly gives us the count of the arguments supplied to the function, which is the required output.

Therefore, the solution approach is straightforward:

1. Gather all the arguments passed to the function into an array using the rest parameters syntax (`...args`).
2. Return the length of this array using the `length` property, which represents the count of the arguments.

Solution Approach

The implementation of the `argumentsLength` function follows a very straightforward algorithm due to the simplicity of the task at hand, which does not require complex data structures or patterns. The steps of the algorithm can be described as follows:

1. Define the function `argumentsLength` using the rest parameters syntax represented by the three dots `...` before the parameter name `args`. This allows the function to accept an indefinite number of arguments.
2. When `argumentsLength` is called, all arguments passed are automatically placed within the `args` array.
3. Use the `length` property of the array `args` to determine how many arguments were passed into the function.
4. Return the value of `args.length`.

In this case, the "data structure" used is simply an array, but it's provided by the language's syntax for handling functions with a variable number of arguments. There is no need for a manual iteration over arguments or any additional logic to count the arguments, as the array's built-in `length` property conveniently provides this information.

Here's a step-by-step walkthrough using the provided TypeScript function:

- When `argumentsLength` is invoked, for example, as `argumentsLength(1, 2, 3);`, the rest parameter `...args` collects the arguments into an array: `args = [1, 2, 3]`.
- The array's `length` property is accessed using `args.length`, which in this case evaluates to `3`.
- The function then returns this value, which represents the count of arguments passed.

Since the reference solution approach section is empty, the walkthrough provided here is derived from the intuitive understanding of the rest parameters and the `length` property found on all arrays in JavaScript and TypeScript.

Example Walkthrough

Let's consider a simple use case to understand the `argumentsLength` function implementation. Imagine we want to call the `argumentsLength` function with three different types of arguments: a number, a string, and a boolean. Here's how the function call might appear:

```
let argCount = argumentsLength(10, "LeetCode", true);
```

Now let's walk through the steps that the `argumentsLength` function would take:

1. Once the function is called, rest parameters syntax `...args` captures all supplied arguments into an array. For this example, inside the function, `args` becomes `[10, "LeetCode", true]`.
2. Next, the function calculates the number of the elements in the array `args` by accessing its `length` property. For our array, `args.length` would evaluate to `3` since there are three items in the array.
3. Finally, the function returns the value of `args.length`, which in this instance is `3`. This is the total count of the arguments that were passed to the function.

By following the steps above, our variable `argCount` now holds the value `3`, indicating that three arguments were passed to the `argumentsLength` function. This example demonstrates the functionality of the function using different argument types and shows the simplicity and effectiveness of using the rest parameters syntax and the `length` property to count the number of function arguments.

Python Solution

```python
# Function to calculate the number of arguments passed in
def arguments_length(*args):
    """
    Returns the count of arguments passed to the function

    Parameters:
    *args: A variable-length argument list that captures all the arguments passed

    Returns:
    int: The total number of arguments passed
    """
    # Return the length of 'args', which is a tuple storing all the arguments
    return len(args)

# Example usage:
# result = arguments_length(1, 2, 3) # result is 3 because three arguments were passed

```

Java Solution

```java
// Class containing the method to calculate the number of arguments passed
public class ArgumentCounter {

    // Method to calculate the number of arguments passed in using varargs
    public static int argumentsLength(Object... args) {
        // Return the length of the 'args' array, which corresponds to the number of arguments
        return args.length;
    }

    // Example usage:
    public static void main(String[] args) {
        // Call the method with three arguments; the result is 3 because three arguments were passed
        int result = argumentsLength(1, 2, 3);
        // Output the result to the console
        System.out.println("Number of arguments passed: " + result);
    }
}

```

C++ Solution

```cpp
#include <cstddef> // for size_t

// Template function to calculate the number of arguments passed in
template<typename... Args>
size_t ArgumentsLength(Args... args) {
    // The size of the parameter pack 'Args' corresponds to the number of arguments
    return sizeof...(args);
}

// Example usage:
// size_t result = ArgumentsLength(1, 2, 3); // result is 3 because three arguments were passed
int main() {
    size_t result = ArgumentsLength(1, 2, 3); // Calls the function with three integer arguments
    // 'result' should be 3 after this call
    return 0;
}

```

Typescript Solution

```typescript
// Function to calculate the number of arguments passed in
function argumentsLength(...args: any[]): number {
    // Return the length of the 'args' array, which corresponds to the number of arguments
    return args.length;
}

// Example usage:
// const result = argumentsLength(1, 2, 3); // result is 3 because three arguments were passed

```

Time and Space Complexity

The time complexity of the `argumentsLength` function is `O(1)` because it simply returns the length of the arguments array without any iterations or additional operations that depend on the number of arguments.

The space complexity of the `argumentsLength` function is `O(n)` where "`n`" represents the number of arguments passed to the function. This is because the space used by the function is directly proportional to the number of arguments, as they are stored in an array internally by the rest parameter "...args".