Problem 2620: Counter

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given an integer

n

, return a

counter

function. This

counter

function initially returns

n

and then returns 1 more than the previous value every subsequent time it is called (

n

,

n + 1

,

n + 2

, etc).

Example 1:

Input:

n = 10 ["call","call","call"]

Output:

[10,11,12]

Explanation:

counter() = 10 // The first time counter() is called, it returns n. counter() = 11 // Returns 1 more than the previous time. counter() = 12 // Returns 1 more than the previous time.

Example 2:

Input:

n = -2 ["call","call","call","call","call"]

Output:

[-2,-1,0,1,2]

Explanation:

counter() initially returns -2. Then increases after each sebsequent call.

Constraints:

-1000

<= n <= 1000

0 <= calls.length <= 1000

calls[i] === "call"

Code Snippets

JavaScript:

```
/**
* @param {number} n
* @return {Function} counter
*/
var createCounter = function(n) {

return function() {

};
};

/**
* const counter = createCounter(10)
* counter() // 10
* counter() // 11
* counter() // 12
*/
```

TypeScript:

```
function createCounter(n: number): () => number {

return function() {

}
}


/**
* const counter = createCounter(10)
* counter() // 10
* counter() // 11
* counter() // 12
```

```
*/
```

Solutions

JavaScript Solution:

```
/**
* Problem: Counter
* Difficulty: Easy
* Tags: general
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {Function} counter
*/
var createCounter = function(n) {

return function() {

};
};

/**
* const counter = createCounter(10)
* counter() // 10
* counter() // 11
* counter() // 12
*/
```

TypeScript Solution:

```
/**
* Problem: Counter
* Difficulty: Easy
* Tags: general
```

```
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function createCounter(n: number): () => number {

return function() {

}
}



/**
* const counter = createCounter(10)
* counter() // 10
* counter() // 11
* counter() // 12
*/
```