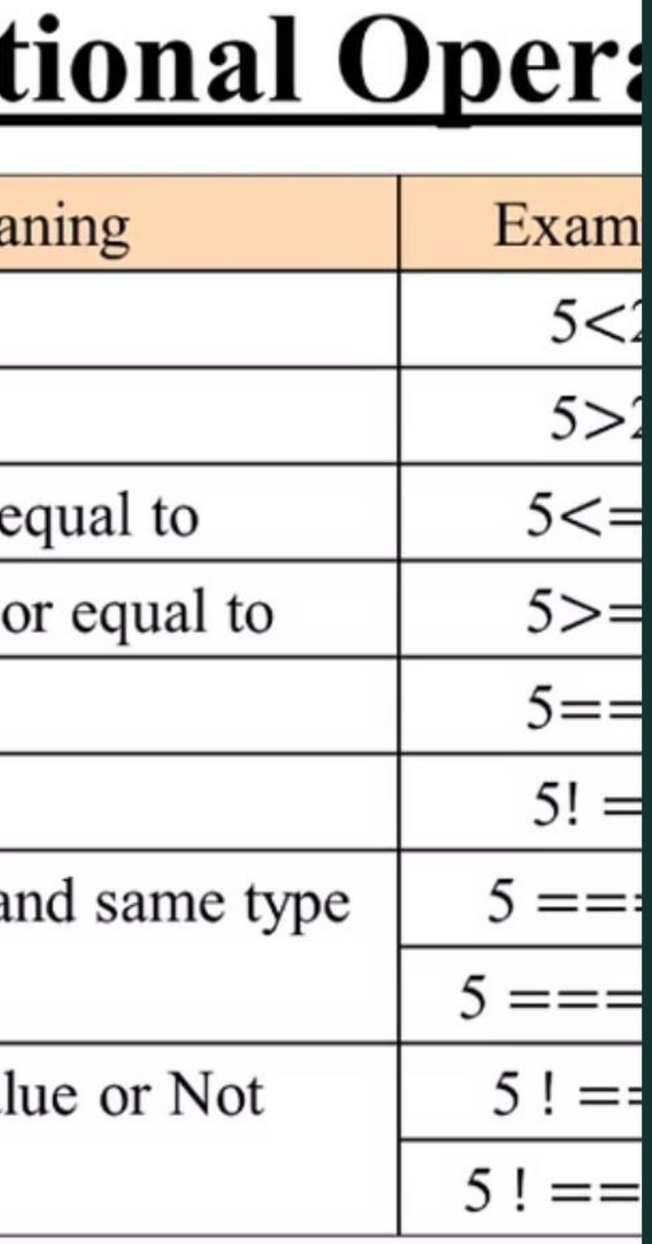
Introduction to JavaScript 30-Day Challenge: Day 2

In this second day of the JavaScript 30-Day Challenge, we dive deeper into JavaScript fundamentals by exploring two exciting coding exercises: "To Be Or Not To Be" and "Counter ||".



To Be Or Not To Be

Equality Check

Develop a function "expect" that verifies if two values are strictly equal (===) or not equal (!==).

Throw Errors

The function should throw custom errors if the values are not equal or equal, providing clear feedback to the developer.

Test Code Easily

This challenge helps developers write more robust and reliable code by providing a simple testing utility.

Two Methods for "expect"

Method 1

Use arrow functions and ternary operators to implement the "expect" function in a concise manner.

Method 2

Implement the "expect" function using traditional conditional statements for a more verbose but perhaps clearer approach.

Comparing Approaches

Both methods achieve the same result, showcasing the flexibility of JavaScript in solving problems.

Method : 1

```
 var expect = function(val) {
return{
 toBe:(value) => (value === val) ? true : (()=>{throw new Error("Not Equal")})(),
notToBe: (value) => (value !== val) ? true : (()=>{throw new Error("Equal")})()
   }
 };
/**
 * expect(5).toBe(5); // true
 * expect(5).notToBe(5); // throws "Equal"
 */
```

Method : 2

```
var expect = function(val) {
return{
toBe:(value) =>{
if(value === val){
return true;
 }
else{
throw new Error("Not Equal");
 }
 },
notToBe:(value) =>{
if(value !== val){
return true;
 }
else{
throw new Error("Equal");
            }
```

Counter ||

1. **Increment**

   The "createCounter" function should provide an "increment" method to increase the current value by 1.

2. **Decrement**

   The "createCounter" function should also include a "decrement" method to decrease the current value by 1.

3. **Reset**

   Additionally, the "createCounter" function should offer a "reset" method to set the current value back to the initial value.

Implementing createCounter

1 **Closure Concept**

The "createCounter" function uses a closure to maintain the current value of the counter.

2 **Encapsulation**

The internal "counter" variable is kept private, ensuring the integrity of the counter's state.

3 **Return Object**

The function returns an object with the three required methods: increment, decrement, and reset.

Counter || Challenge

Increment

The counter can be incremented by 1 with each call to the "increment" method.

Decrement

The counter can be decremented by 1 with each call to the "decrement" method.

Reset

The counter can be reset to its initial value by calling the "reset" method.

Counter || Implementation

init	The initial value of the counter, passed as an argument to the
increment()	Increases the current value by 1 and returns the new value.
decrement()	Decreases the current value by 1 and returns the new value.
reset()	Sets the current value back to the initial value and returns the reset value.

```
var createCounter = function(init) {
    let counter = init;
    return{
        increment: function(){
            return ++counter;
        },
        reset: function(){
            counter = init
            return counter;
        },
        decrement: function(){
            return --counter;
        }
    };
```

Counter || Usage Examples

1 2 3 4

Create Counter

Call the "createCounter" function with an initial value to get a counter object.

Increment

Use the "increment" method to increase the counter's value.

Reset

Call the "reset" method to set the counter back to its initial value.

Decrement

Decrease the counter's value using the "decrement" method.

Conclusion and Next Steps

This JavaScript 30-Day Challenge has covered two valuable exercises, "To Be Or Not To Be" and "Counter ||", which have strengthened your understanding of JavaScript fundamentals. As you continue your JavaScript learning journey, be sure to practice these concepts and explore more advanced topics to become a skilled JavaScript developer.