Make Counter

</> Easy 5mins 14.8k done

Implement a function `makeCounter` that accepts an optional integer value and returns a function. When the returned function is called initially, it returns the initial value if provided, otherwise 0. The returned function can be called repeatedly to return 1 more than the return value of the previous invocation.

Examples

```
const counter = makeCounter();
counter(); // 0
counter(); // 1
counter(); // 2
```

With a custom initial value:

```
const counter = makeCounter(5);
counter(); // 5
counter(); // 6
counter(); // 7
```

Asked at these companies

Solution

This question evaluates your knowledge on closures and higher-order functions.

Approach 1: Decrement then postfix increment

1. The `makeCounter` function accepts an optional parameter `initialValue`, which is set to 0 by default.
2. Inside the `makeCounter` function, we declare a variable `count` and initialize it with `initialValue` - 1. We have to declare using `let` since we need to increment it.
3. We return an anonymous function (a closure) that captures the `count` variable from the outer scope.
4. Whenever the returned function is called, we increment `count` then return it.

It's necessary to initialize `count` with one less than `initialValue` because in the returned function we increment before returning. Doing this will allow the first call of the returned function to return the `initialValue`.

JavaScript TypeScript

```javascript
/**
 * @param {number} initialValue
 * @return {Function}
 */
export default function makeCounter(initialValue = 0) {
  let count = initialValue - 1;

  return () => {
    count += 1;
    return count;
  };
}
```

Approach 2: Postfix increment

In the previous solution, it is a little awkward to decrement `initialValue` by 1 only to increment it later. Thankfully we can use the postfix increment operator to increment a variable **after** the value has been returned.

JavaScript TypeScript

```javascript
/**
 * @param {number} initialValue
 * @return {Function}
 */
export default function makeCounter(initialValue = 0) {
  let count = initialValue;

  return () => {
    return count++;
  };
}
```

Approach 3: One-liner

We can make the solution even shorter by not initializing a `count` variable and incrementing the `defaultValue` instead. Mutating a function's parameters is usually not recommended due to causing of side effects. However in this case `initialValue` is a primitive and incrementing it will not cause any side effects.

JavaScript TypeScript

```
/**
 * @param {number} value
 * @return {Function}
 */
export default function makeCounter(value = 0) {
  return () => value++;
}
```