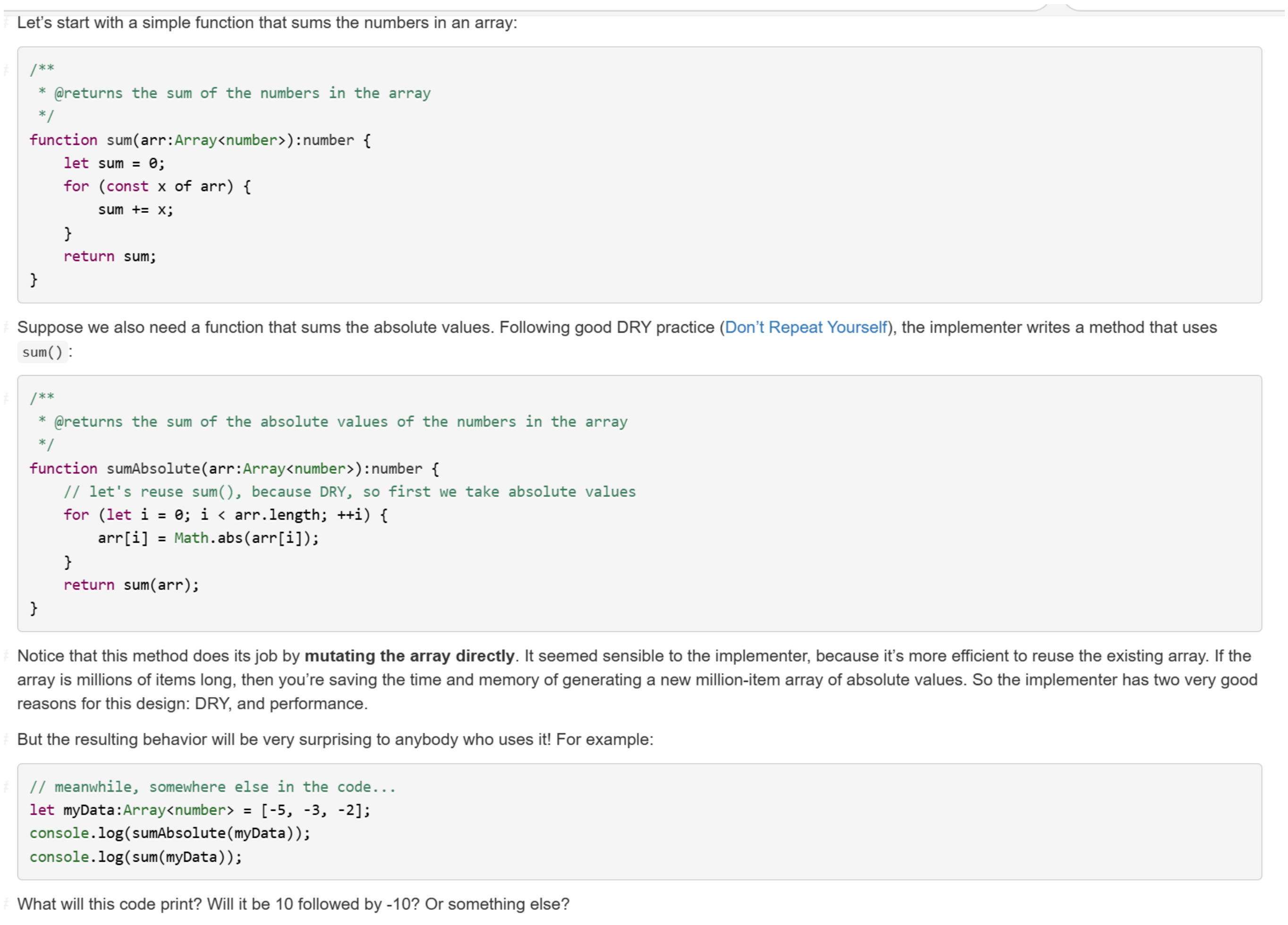
8.可变性与不变性

尽量用不可变的

例子：



Let’s start with a simple function that sums the numbers in an array:

```
/**
 * @returns the sum of the numbers in the array
 */
function sum(arr:Array<number>):number {
    let sum = 0;
    for (const x of arr) {
        sum += x;
    }
    return sum;
}
```

Suppose we also need a function that sums the absolute values. Following good DRY practice (Don’t Repeat Yourself), the implementer writes a method that uses `sum()`:

```
/**
 * @returns the sum of the absolute values of the numbers in the array
 */
function sumAbsolute(arr:Array<number>):number {
    // let's reuse sum(), because DRY, so first we take absolute values
    for (let i = 0; i < arr.length; ++i) {
        arr[i] = Math.abs(arr[i]);
    }
    return sum(arr);
}
```

Notice that this method does its job by **mutating the array directly**. It seemed sensible to the implementer, because it’s more efficient to reuse the existing array. If the array is millions of items long, then you’re saving the time and memory of generating a new million-item array of absolute values. So the implementer has two very good reasons for this design: DRY, and performance.

But the resulting behavior will be very surprising to anybody who uses it! For example:

```
// meanwhile, somewhere else in the code...
let myData:Array<number> = [-5, -3, -2];
console.log(sumAbsolute(myData));
console.log(sum(myData));
```

What will this code print? Will it be 10 followed by -10? Or something else?

结果将是两个10，因为abs真的修改了arr中的值。

```
function dropCourse6(subjects:Array<string>):void
```

Note that `dropCourse6` explicitly says in its spec that its `subjects` argument may be mutated.

Next, following test-first programming, we devise a testing strategy that partitions the input space, and choose test cases to cover that par

```
// Testing strategy:
//   partition on subjects.length: 0, 1, n > 1
//   partition on contents: no 6.xx, some (but not all) 6.xx, all 6.xx
//   partitions on position: subjects has a first element that is 6.xx? yes, no
//                                  ... a middle element 6.xx? yes, no
//                                  ... a last element 6.xx? yes, no

// Test cases:
//   [] => []
//   ["8.03"] => ["8.03"]
//   ["14.03", "9.00", "21L.005"] => ["14.03", "9.00", "21L.005"]
//   ["2.001", "6.01", "18.03"] => ["2.001", "18.03"]
//   ["6.045", "6.031", "6.036"] => []
```

Finally, we implement it:

```
function dropCourse6(subjects:Array<string>):void {
    let iter:MyIterator = new MyIterator(subjects);
    for (let subject = iter.next(); subject !== undefined; subject = iter.next()) {
        if (subject.startsWith("6.")) {
            // remove the subject from the array
            subjects.splice(subjects.indexOf(subject), 1);
        }
    }
}
```

Now we run our test cases, and they work! … almost. The last test case fails:

```
// dropCourse6(["6.045", "6.031", "6.036"])
//   expected [], actual ["6.031"]
```

这个之所以会保留6.031，是因为去掉6.045后6.031自动改为0号索引。这样就会忽略近乎一半的数字！

所以尽可能创建一个新的对象来复制原来的对象，然后对新对象进行操作。