Tip 1: Signal Unchanging Values With const

In this tip, you'll learn to use const to avoid reassignment and signal your intention to other developers.

We'll cover the following

Which variable declaration to use?

Modern JavaScript introduced several *new* variable declarations, which is great. But it also introduced a new problem: *Which variable declaration should be the default? And when should we use another type?*

In the past, you had only one option for *non-global* variable assignment: `var`. Now there are many different options—`var`, `let`, and `const`—and each one has an appropriate usage. Try and keep things simple. In most cases, `const` is the best choice, not because it allows you to do the most, but because it lets you do the least. It has restrictions that make your code more *readable*.

> **ECMAScript** is the official technical specification for JavaScript. JavaScript incorporated major syntax changes in *ECMAScript 5 and ECMAScript 6*, which are referred to as ES5 and ES6. Going forward, the spec will be updated yearly. Most developers now refer to the spec by year, such as ES2017.

Using `const`

`const` is a variable declaration that you **can't** reassign within the context of the block. In other words, once you establish it, it can't be changed. That doesn't mean

it's **immutable**—*a value that cannot be changed*. If it's assigned to an array, the items in the array can be changed. We'll look at this more shortly.

It may seem odd to developers in other languages with a constant assignment that `const` is the preferred declaration. In those languages, a constant is usually something you'd write in **ALLCAPS** and only use on rare occasions to denote things that are never going to change, like the first digits of *pi*.

In JavaScript, though, `const` is a great default choice precisely because it can't be reassigned. When you assign a value, you aren't just declaring a piece of information. You're also signaling what you plan to do with that information. When you assign values and signal that they won't be changed, you give future developers (including yourself!) the knowledge that they can forget about a value while they skim the code. And when you're reading a large amount of code that you haven't seen before, you'll be happy that you can forget some of what you read.

Example 1

Let's assume you're fixing a bug in a piece of code. You're skimming through the code to get an idea of how it works and to see if you can guess where the problem might be.

Consider two programs. The first program uses `const` to assign a variable.

```
const taxRate = 0.1;
const total = 100 + (100 * taxRate);
// Skip 100 lines of code
console.log(`Your Order is ${total}`);
```

The second uses `var` to assign a variable.

```
var taxRate = 0.1;
var total = 100 + (100 * taxRate);
// Skip 100 lines of code
console.log(`Your Order is ${total}`);
```

They look nearly identical, but the first is much easier to understand. Ignore the fact that a block of code shouldn't be 100 lines long; you have a large amount of

code where lots of changes are occurring.

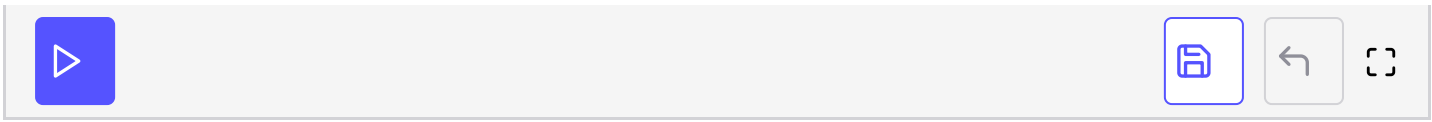
With the first block, you know exactly what will get returned: `Your Order is 110`. You know this because `total` is a constant that can't be reassigned. With the second block, you have no idea what the return value is going to be. You are going to need to go through the 100 lines of additional code looking for *loops* or *conditionals* or *reassignments* or anything that might change the value. Maybe the code is adding a shipping cost. Maybe additional items will be added to the total. Maybe a discount is going to be applied and the total will drop.

You have no idea what the `total` is going to be when it's assigned with `var`. When you assign a variable with `const`, it removes one additional piece of information that you need to retain in your head while reading code.

Example 2

Consider one last example:

```
const taxRate = 0.1;
const shipping = 5.00;
let total = 100 + (100 * taxRate) + shipping;
// Skip 100 lines of code
console.log(`Your Order is ${total}`);
```



Take a moment and think about what you can be certain of from this code. You know you can't be sure of the total. The developers have signaled that `taxRate` and shipping are unchanging (if only that were true), but the total isn't permanent. You know this value can't be trusted.

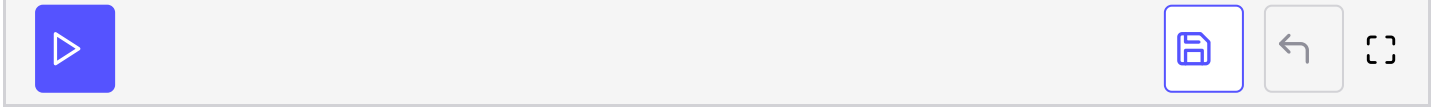
The best case is to know that an assignment won't change. The second best case is to know that it might change. If you can see that the developers used `const` regularly and let rarely, you can guess areas of change.

Make all variable assignments either a *known-known or a known-unknown*.

Changing the value of a `const` variable

There's one important consideration when using `const`: **A value assigned to const is not immutable**. In other words, you can't reassign the variable, but you can change the value. That may seem contradictory, but here it is in practice.

```
const discountable = [];
const cart = [
    {
        item: 'Book',
        discountAvailable: false,
    },
    {
        item: 'Magazine',
        discountAvailable: true,
    },
];
// Skip some lines
for (let i = 0; i < cart.length; i++) {
    if (cart[i].discountAvailable) {
        discountable.push(cart[i]);
    }
}
console.log(discountable);
```

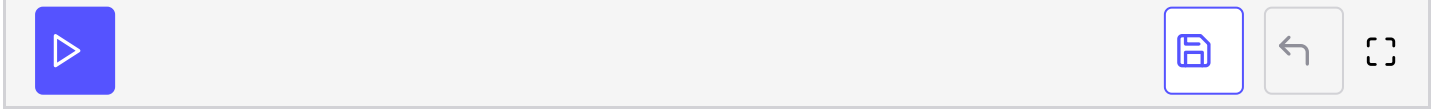


This is perfectly valid code. Even though discountable is assigned with `const`, you can still push items to it. This creates the exact problem we saw earlier: You can't be certain of what you'll see later in the code. For objects, arrays, or other collections, you'll need to be more disciplined.

There's no clear consensus on what you should use, but your best bet is to avoid mutations as much as possible.

Here's an example of the previous code written without mutations.

```
const cart = [
    {
        item: 'Book',
        discountAvailable: false,
    },
    {
        item: 'Magazine',
        discountAvailable: true,
    },
];
const discountable = cart.filter(item => item.discountAvailable);
console.log(discountable);
```



Same result. No mutations. If the code is confusing, you can jump to Chapter 5,

Simplify Loops, for more information about array methods.

For now, just use `const` as a default. Once the code changes to the point where `const` is no longer appropriate, you can try a different declaration.

1

What will be the output of the code given below?

```
const obj = {};
obj.name = "Mark";
console.log(obj);
```

2

What will be the output of the code given below?

```
const value = 0;
for(let i=0; i<10; i++){
   value += i;
}
console.log(value);
```

3

Which of the following is true about the code given below?

```
1- var planet = "Mars";
2- const days = 365;
3- var value = 10;
4- if(value > 5) {
5-   var planet = "Jupiter";
6-   days = 1050;
7- }
8- console.log(planet);
9- console.log(days);
```

Retake Quiz

In the next tip, you'll see precisely when `const` is no longer an appropriate choice and why you should use a new declaration: `let`.