When to use const and when to use let

If you can't do as much with `const` as you can with `let`, why would you prefer to use it rather than `let`? In fact `const` is very useful. If you use `const` to name a value, it tells anyone looking at your code that this name will never be assigned to a different value. Any time they see this name, they will know what it refers to.

In this course, we adopt the following principle about when to use `let` and when to use `const`:

Use `const` when you can, and use `let` when you have to.

This means that if you can initialize a variable when you declare it, and don't need to reassign it later, make it a constant.

When to use `let` and `const` in JavaScript

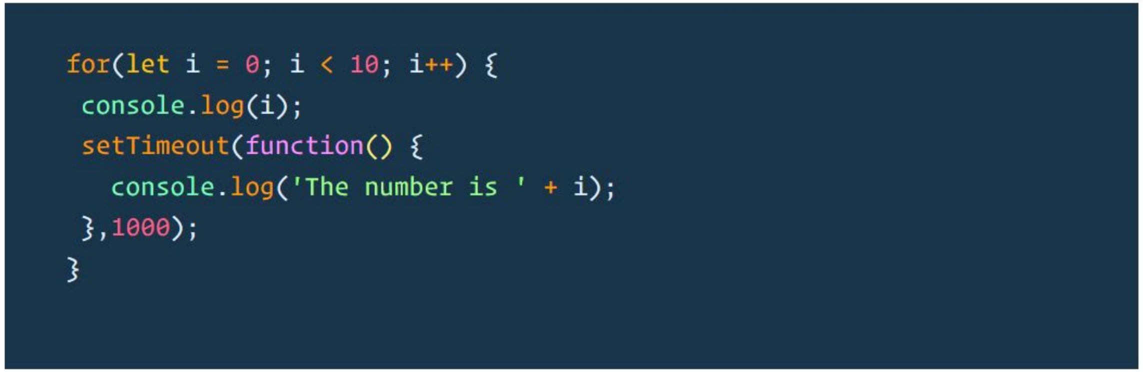
Keyword **`let`**

- Will the value stored in the variable change at some point?
- Use with loops (to keep track of the loop count)

Keyword **`const`**

- Lets other programmers know the value stored in the variable can’t be reassigned
- Meaning: (this value) has to stay (this value)

Use **let** in a JS FOR Loop



```
for(let i = 0; i < 10; i++) {
 console.log(i);
 setTimeout(function() {
   console.log('The number is ' + i);
 },1000);
}
```

What do we know about `let`? It's block-scoped. We have curly brackets in the `for` loop. If you run it now, after a second we'll log zero through nine. We're not getting 10, 10 times. We getting it as it was declared each and every time.

As a note, you couldn't use a `const` for this because it needs to overwrite itself, and you can't assign the same variable twice. When we use `let`, it's going to scope `i` to our curly brackets.