Let vs Const in Loops and Objects

Welcome to Scribbles on Scripts!

Declaring variables is one of the first things you learn on your coding journey, and you're likely already a pro with let and const. If you want to keep the value of a variable the same, you use const, and otherwise you use let. End of story, right? Weeeellllll. New contexts can come with new questions about old concepts. Let's get into it .

So let's say we have a puzzle store and we want to learn more about what's selling well. We want to know if any of our top 3 bestsellers are "Sunny", and if they are, we'll save them to a new array. If we know our customers generally tend to like them, maybe we can feature them in our store when there's bad weather.

```
const bestSellers = ["Sunny Mountain", "Art Attack",
                     "Sunny Village", "Flower Meadow",
                     "City Nightscape"];

const sunPuzzles = [];

for (let index = 0; index < 3; index++) {
    const hasSun = bestSellers[index].includes("Sun");

    if (hasSun) {
        sunPuzzles.push(bestSellers[index]);
    }
}
```

To make our code a little more readable, we need a variable in the loop that tells us if we're currently looking at a sunny puzzle.

So,what should we use? Const or let? If this was outside of a loop, the answer would be straightforward. We're not reassigning anything to this variable, so in goes the const. But with it being in a loop... this variable has a different value in each iteration... so aren't we changing the value?

To answer this we have to think about what's happening when we run the code. Each iteration of our loop exists separately from the others, it is its own little mini dimension or bubble, it doesn't know about the other iterations. When the last line in the body of the loop is executed, the iteration bubble bursts, and a new one comes into existence, representing the next iteration. The only sign it ever existed is any effect it had on other variables outside the loop, and the loop counter going up by one. And so, each iteration has their own version of the variable, it's never actually updated, and it's perfectly acceptable to use consts in loops. Unless of course you actually want to update the value it stores within the loop,then you still need let.

Functions are like this too, each function call is it's own bubble the same way each loop iteration is it's own bubble. You can of course call them again, but it's a different bubble.

Another context where let and const don't quite behave the way you might expect are objects. This again is easier with an example.

```
const kitty = { type: "orange", age: 0.5 };
kitty.age = 1;

let cat = { type: "tabby", age: 2 };
cat.age = 3;
```

So looking at this bit of code, you would think that this top portion using const should error out. That object shouldn't be allowed to change like that, right? But it actually can. This came up in a previous video as well, but javascript objects actually just contain an address to where the rest of the object lives. So it is this connection that can't be broken, the variable name must always point to this specific object. The properties of the object are fair game, but you can't reassign the variable. So in code, you'd be allowed to reassign like this with let, but not with const.

```
cat = kitty;

kitty = cat;
```

The moral of the story is that you should still use const if you don't intend to reassign or change the value of your variable. It really does help make your code more readable to not just other developers but to future you as well.

If you found this video helpful, please give it a like, and subscribe for more just like it. If there's a topic you'd like me to cover in a future video, let me know in the comments down below! Thanks for watching!