The IIFE Interview Question

Find other ways to solve the IIFE interview question shown in the previous lesson. We'll tie together concepts we covered earlier in the course.

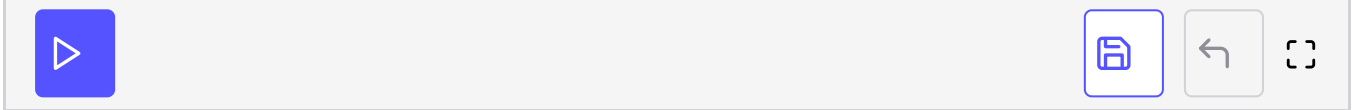
Recap

In the previous lesson, we introduced an interview question. We want the following code to print out:

```
0
1
2
3
4
```

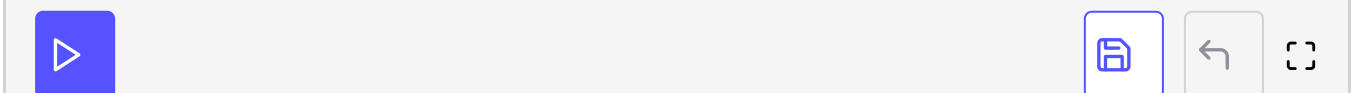
We want a 1-second pause in between these printouts. Will this code result in this printout? If not, what will it print? What changes can we make in order to make it work as expected?

```
for (var i = 0; i < 5; i++) { // We are explicitly using `var` for a reason
    const time = i * 1000;
    setTimeout(function() {console.log(i);}, time);
}
```



We gave one solution.

```
for (var i = 0; i < 5; i++) {
    (function(num) {
        const time = num * 1000;
        setTimeout(function() {console.log(num);}, time);
    })(i);
}
```

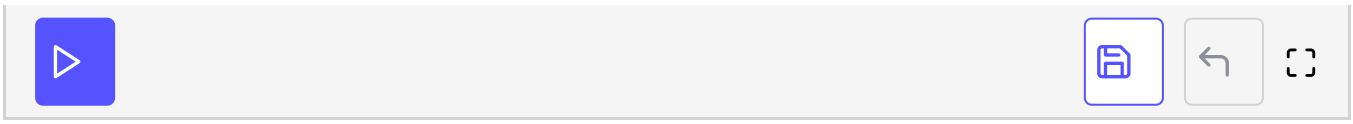


With our knowledge of JavaScript, however, there are multiple ways to solve this problem. We don't need to use an IIFE. Let's discuss a couple more solutions to revisit some concepts and solidify our knowledge.

`let`

This is one of the simplest ways to solve the problem without using an IIFE.

```
for (let i = 0; i < 5; i++) { // Using `let`
    const time = i * 1000;
    setTimeout(function() {console.log(i);}, time);
}
```



All we did in this solution is change `var` to `let`. This works because of the way `let` is scoped with loops. It's scoped to the loop's body *during its current iteration*.

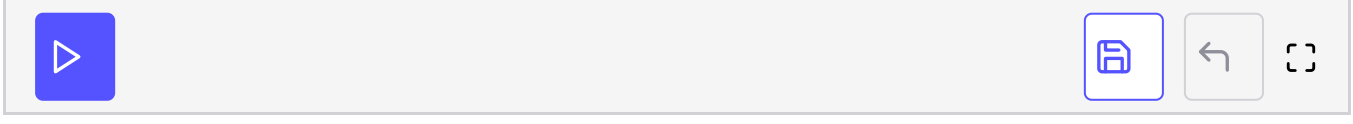
This means that what we've been trying to do with the IIFE has already been done for us by the `let` keyword. We were trying to solve the problem of unexpected scoping in a loop when iterating using `var`, and JavaScript has a solution.

There's one more way to solve this problem that I'd like to discuss.

`{}`

A pair of brackets.

```
for (var i = 0; i < 5; i++) { // We are explicitly using `var` for a reason
    {
        const j = i;
        const time = j * 1000;
        setTimeout(function() {console.log(j);}, time);
    }
}
```



We mentioned before that one of the changes introduced with ES2015 was bracket-scoping. Any pair of curly brackets creates a new scope.

Here, because we're using `var` to declare the iterator, we expect to see that `i` is not locally scoped. To get around this, we create an arbitrary pair of brackets and create another variable, `j`.

We make `j` equal to `i` and because `j` is locally scoped, when it gets its value, it will hold on to it. `i` will change through iterations, increasing, but once `j` receives its value, it's locked in.

Every run through of the loop runs a new instance of the code inside the brackets, so there are actually 5 versions of that code initialized and running in memory. They run until their `setTimeout` function completes its job.

TL;DR

In summary, each version of `setTimeout` uses the `j` value it is locked in with. `i` changes rapidly at the beginning of code execution, virtually instantaneously going from 0 - 5, but between each iteration, it creates a new instance of the code inside the brackets. It creates a new `j` and locks `setTimeout` into the same scope as `j`. This makes our code work the way we want.

Conclusion

JavaScript has given us many ways to solve our funky scoping behavior and our woes with variables. We've just covered three different ways to fix scoping inside a loop.

One of the key ideas of functional programming is keeping variables neat and tidy. We aim to keep them locked away inside their own little scope, only accessible strictly where necessary. This prevents functions and other code from messing with them and potentially changing them, preventing strange bugs that are unwieldy to track down.

We've just covered three different ways to do that inside loops. An IIFE does it perfectly, brackets can do it as well, and the built-in `let` solves exactly this

problem for loops.

IIFEs and arbitrary brackets can frequently be used interchangeably. When should we use one vs. the other? In general, an IIFE is easier to read and developers will more immediately know what the code is trying to do. Pre-2015, an IIFE was one of the simplest ways to solve our problem here, so it's what experienced developers are used to. IIFEs are also used elsewhere in JS, such as by the widely used jQuery library.

There will be times when the brackets seem more elegant. It's ultimately up to you. I tend to prefer IIFEs unless the brackets do indeed simplify how the code looks and reads. Again, the choice is always yours.