**Using Lexical *this* in a JavaScript Class Method with ES6 Arrow Functions**

The way JavaScript handles scope is unique. It makes the language so fascinating to learn. It is different from any other major programming languages and leads to unexpected output if you don’t understand it correctly. JavaScript has many quirks that make it so fascinating, such as its asynchronous nature, event loops, prototypes, closure and so on. I think *this* is one of them. It is often misunderstood and misused. But, once you understand how *this* works, it will take you to the next level of JavaScript programming.

This post will not go too deep into discussing JavaScript’s scope and its magical *this*. To do so, I probably need to write an entire book. In fact, [You Don’t Know Js](https://github.com/getify/You-Dont-Know-JS) (a highly recommended read if you want to understand JavaScript) dedicates [entire chapter](https://github.com/getify/You-Dont-Know-JS/tree/master/this%20%26%20object%20prototypes) to discuss *this*.

Instead, I want to show you how the scoping of this changes with ES6 arrow functions compared to the old way in the context of a class method.

You can see the Past Posts section on the right side of this website and it has a Show more option (at least, for now. It may be different by the time you are reading this post). That toggler is in fact injected by JavaScript (the real code is actually written in TypeScript, which I will show you at the end).

The class creates a div with the onclick event handler. Now, check out how it is written without an arrow function.

In JavaScript class, *this* points to the class object. When new is used to create the toggler object, it attaches this to the class object itself. *this* inside the class method points to the objects within the class. Therefore, you have access to node by this.node. So far, so good.

Take a look at the addEventListener in the render method. If you are using the regular function expression in the callback, *this* will fall out of scope. This is because *this* belongs to the call site, which is not the instantiated object itself, but the actual HTML dom (this.node in this case) as this is where event gets triggered. Therefore, if you try to access the clickHandler() with this, JavaScript engine will throw clickHandler not a function TypeError. Hence, we have to attach this to another variable so that the call back function has access to \_this.

<script src="https://gist.github.com/mydatahack/021e60b92988756e15899cf2bb2359f0.js"></script>

As mentioned above, this results in not a function TypeError because *this* falls out of the scope within the callback.

<script src="https://gist.github.com/mydatahack/b4b18eee8eecdb2a20a4972f073b9033.js"></script>

Here comes the ES6 arrow function expression. The syntactical difference is just the surface. The real difference is it creates lexical scope within the function, which was not possible with pre ES6 language specification.

Let’s refactor the render method. You see *this* within the arrow function can still reference the object within the class. *This* is cool.

<script src="https://gist.github.com/mydatahack/4232a799628270a3426da7252e23bf8d.js"></script>

Now all you need to do is instantiate the class and call render method.

<script src="https://gist.github.com/mydatahack/e9e7bfef78a05c6baf6e18647f616252.js"></script>

As mentioned above, the actual implementation for the archive toggler on this website is done in TypeScript (at the time this post was written). This will probably change in the near future. Here is the code just for fun.

<script src="https://gist.github.com/mydatahack/4c8b6ccea437b8d60baa1d06731cb023.js"></script>

My advice is to embrace arrow functions and use it whenever. Let the transpiler handle the older browser support.

Lastly, arrow functions’s lexical scoping opened up more possibilities to write JavaScript in a new way. Take property initialiser for example. It is an experimental ES6 JavaScript feature that let you write class in a slightly different way by taking advantage of arrow function’s lexical scope. You can check out [this post](https://www.mydatahack.com/refactoring-react-es6-class-components-with-property-initialisers/) where I use it to refactor React class component.