You Don't Know JS: Scope & Closures

Appendix C: Lexical-this

Though this title does not address the `this` mechanism in any detail, there's one ES6 topic which relates `this` to lexical scope in an important way, which we will quickly examine.

ES6 adds a special syntactic form of function declaration called the "arrow function". It looks like this:

```
var foo = a => {
        console.log( a );
};

foo( 2 ); // 2
```

The so-called "fat arrow" is often mentioned as a short-hand for the *tediously verbose* (sarcasm) `function` keyword.

But there's something much more important going on with arrow-functions that has nothing to do with saving keystrokes in your declaration.

Briefly, this code suffers a problem:

```
var obj = {
        id: "awesome",
        cool: function coolFn() {
                console.log( this.id );
        }
};

var id = "not awesome";

obj.cool(); // awesome

setTimeout( obj.cool, 100 ); // not awesome
```

The problem is the loss of `this` binding on the `cool()` function. There are various ways to address that problem, but one often-repeated solution is `var self = this;`.

That might look like:

```
var obj = {
        count: 0,
        cool: function coolFn() {
                var self = this;

                if (self.count < 1) {
                        setTimeout( function timer(){
                                self.count++;
                                console.log( "awesome?" );
                        }, 100 );
                }
        }
};

obj.cool(); // awesome?
```

Without getting too much into the weeds here, the `var self = this` "solution" just dispenses with the whole problem of understanding and properly using `this` binding, and instead falls back to something we're perhaps more comfortable with: lexical scope. `self` becomes just an identifier that can be resolved via lexical scope and closure, and cares not what happened to the `this` binding along the way.

People don't like writing verbose stuff, especially when they do it over and over again. So, a motivation of ES6 is to help alleviate these scenarios, and indeed, *fix* common idiom problems, such as this one.

The ES6 solution, the arrow-function, introduces a behavior called "lexical this".

```
var obj = {
        count: 0,
        cool: function coolFn() {
                if (this.count < 1) {
                        setTimeout( () => { // arrow-function ftw?
                                this.count++;
                                console.log( "awesome?" );
                        }, 100 );
                }
        }
};

obj.cool(); // awesome?
```

The short explanation is that arrow-functions do not behave at all like normal functions when it comes to their `this` binding. They discard all the normal rules for `this` binding, and instead take on the `this` value of their immediate lexical enclosing scope, whatever it is.

So, in that snippet, the arrow-function doesn't get its `this` unbound in some unpredictable way, it just "inherits" the `this` binding of the `cool()` function (which is correct if we invoke it as shown!).

While this makes for shorter code, my perspective is that arrow-functions are really just codifying into the language syntax a common *mistake* of developers, which is to confuse and conflate "this binding" rules with "lexical scope" rules.

Put another way: why go to the trouble and verbosity of using the `this` style coding paradigm, only to cut it off at the knees by mixing it with lexical references. It seems natural to embrace one approach or the other for any given piece of code, and not mix them in the same piece of code.

Note: one other detraction from arrow-functions is that they are anonymous, not named. See Chapter 3 for the reasons why anonymous functions are less desirable than named functions.

A more appropriate approach, in my perspective, to this "problem", is to use and embrace the `this` mechanism correctly.

```
var obj = {
        count: 0,
        cool: function coolFn() {
                if (this.count < 1) {
                        setTimeout( function timer(){
                                this.count++; // `this` is safe because of `bind(..)`
                                console.log( "more awesome" );
                        }.bind( this ), 100 ); // look, `bind()`!
                }
        }
};

obj.cool(); // more awesome
```

Whether you prefer the new lexical-this behavior of arrow-functions, or you prefer the tried-and-true `bind()`, it's important to note that arrow-functions are **not** just about less typing of "function".

They have an *intentional behavioral difference* that we should learn and understand, and if we so choose, leverage.

Now that we fully understand lexical scoping (and closure!), understanding lexical-this should be a breeze!