Arrow Functions

three ways they behave differently to regular functions

Ash and Manasseh

1. You can write less to achieve the same outcome - syntax sugar 🍬

ES6:

```
let array = ["Bob","Joe"];
let capitals = arr.map(x => x.toUpperCase());
```

ES5:

```
var array = ["Bob","Joe"];
var capitals = arr.map(function(x) { return x.toUpperCase() });
```

2. Lexical scoping (simple explanation)

“Arrow functions do not have their own this value. The value of this inside an arrow function is always inherited from the enclosing scope.” - mozilla.org

Therefore if you call “this.name” inside an arrow function, it will look for the “this.name” in it’s parent function, not in the function it is running from.

2. Lexical Scoping (complex explanation)

In ES5, this function wouldn't work:

```
function parentFunction(){
this.variable = 0;
    this.changeVariable = function(){
        this.addNumber = function(){
            return this.variable++;
        }

    }
}
```

can't see each other

This is ES5 code.

addNumber() will not work because:

this.variable is trying to add by 1. this.variable isn't found because "this" just looks inside its own function for the variable and the doesn't exist in addNumber().

Running this code would return "NaN" because this.variable isn't found by addNumber()

2.1 To fix the previous error in ES5 you would...

```
function parentFunction(){
this.variable = 0;
var that = this;
    this.changeVariable = function(){
        this.addNumber = function(){
            return that.variable++;
        }

    }
}
```

Add a variable called “that” on the level where the variable is created.

Then call that.variable instead of this.variable and the code will run.c

2.2 Using ES6, we can solve this problem by doing...

```
function parentFunction(){
this.variable = 0;
    this.changeVariable = () => {
        this.addNumber = () => {
            return this.variable++;
        }

    }
}
```

Much cleaner code to achieve the result we want.