ES6

Arrow Functions

With traditional functions there are 4 ways to control `this`.

Call as a method.

```
var foo = {

    methodName = function() {
        return this;
    }
}

foo.methodName(); // returns foo
```

Function declaration without a base

```
function functionName() {
    return this;
}

functionName(); // returns global or undefined (strict mode);
```

Using the new operator

```
function Foo() {
}

var foo = new Foo(); \\ this === foo
```

Force the context using apply, call or bind.

```
var Teddi = {
    name: "Teddi",
    speak: function() {
        return this;
    }
}

var Mat = {
    name: "Mat"
}

Teddi.speak().bind(Mat); // this === Mat
```

Arrow functions provide two huge advantages.

They are less verbose than traditional function expressions.

With arrow functions `this` is bound to the enclosing scope (lexical) at creation time.

This cannot be changed.

The `new` operator, `bind`, `call`, and `apply` have no effect on `this`.

Arrow Functions are anonymous.

Single Line Expressions

```
\\ES5

var numbers = [1,2,3,4,5];
var timesTwo = numbers.map(function (number) {
  return number * 2;
});
console.log(timesTwo); // [2, 4, 6, 8, 10]

\\ES6

var numbers = [1,2,3,4,5];
var timesTwo = numbers.map(number => number * 2);
console.log(timesTwo); // [2, 4, 6, 8, 10]
```

Arrow Function Arguments

```
() => 2 + 5; // No Argument

x => x * 2; // Single Argument

(x,y) => x * y; // Multi Argument
```

Arrow Function Body

```
// Single-Line Expression - implicit return;

x => x * 2;

// Statement Block - behaves like a normal body.

x => {
    let y = x * 2;
    return x * y;
}
```

So Pretty!

```
arr = [4,9,16]

arr.map(e => Math.sqrt(e));
```

Questions?