JavaScript This

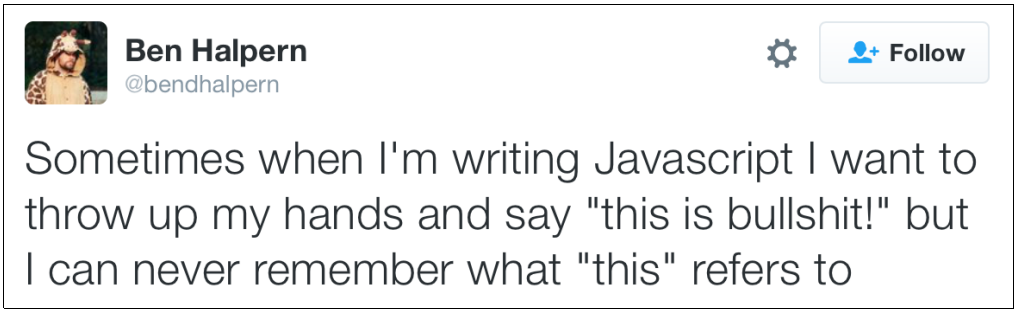
Download Demo Code <../js-this-demo.zip>

Goals

- Learn how to stop worrying and love the keyword ***this***
- Explain what ***.call*** does
- Explain what ***.bind*** does
- Use ***.call*** and ***.bind*** to reassign the value of the keyword ***this***

This & Bind

This



<_images/this-is-bullshit.png>

Mystery of the Undefined Fluffy

demo/fluffy.js

```
class Cat {
  constructor(name) {
    this.name = name;
  }

  dance(style) {
    return `Meow, I am ${this.name}` +
        ` and I like to ${style}`;
  }
}
```

makes sense…

```
let fluffy = new Cat("Fluffy");

fluffy.name;              // "Fluffy"

fluffy.dance("tango")     // works!
```

wtf?

```
let fDance = fluffy.dance;

fDance("salsa");          // error?!
```

JavaScript “Functions”

In a sense, JavaScript doesn’t have functions.

Everything is called on something, like a method.

```
function whatIsThis() {
  console.log("this =", this);
}
```

```
let o = { myFunc: whatIsThis };

o.myFunc();    // get "this = o"
```

```
whatIsThis();      // wtf?!
```

Global Object

When you call a function on nothing …

… you call it on the “global object”

- In browser JS, that’s ***window*** (the browser window)
- in NodeJS, that’s ***global*** (where some Node utilities are)

You’ve relied on that, even if you didn’t realize it!

```
alert("Hi!");

window.alert("Hi!");   // -- same thing!
```

Therefore, a “function” called at the top level is same as:

```
window.whatIsThis()
```

Undefined Fluffy

demo/fluffy.js

```
class Cat {
  constructor(name) {
    this.name = name;
  }

  dance(style) {
    return `Meow, I am ${this.name}` +
        ` and I like to ${style}`;
  }
}
```

so… what’s happening here?

```
let fluffy = new Cat("Fluffy");

fluffy.name;              // "Fluffy"

fluffy.dance("tango")     // works!

let fDance = fluffy.dance;

fDance("salsa");          // error?!
```

```
fluffy.dance("tango");
```

- **Find** the ***dance*** method on fluffy
- **Call** the ***dance*** method on fluffy — yay!

```
let fDance = fluffy.dance;
fDance("tango");
```

- **Find** the ***dance*** method on fluffy
- **Call** the ***dance*** method on the global window — ut oh

Call

Sometimes, you'll need to say "call this function *on this object*"

That's what ***call()*** is for!

```
let fDance = fluffy.dance;

// call on fluffy, passing "tango" as arg
fDance.call(fluffy, "tango");
```

```
whatIsThis.call(fluffy);       // this = fluffy
```

Note: *apply()*

There is a related function, ***apply()***: for this, you can pass the list of arguments to the function as an array, rather than passing one-by-one.

This used to be a very important technique, since it was the only reasonable way to call a function that expected several individual arguments where you already had those arguments in a list:

```
Math.max(1, 2, 3);       // Math.max expects indiv arguments

let myNums = [1, 2, 3];  // If you already have an array ...

Math.max.apply(null, myNums);  // pass that array as indiv arguments
                               // (don't care what "this" is, so pass `null`)
```

Nowadays, however, this is much more easily done with the spread operator:

```
Math.max(...myNums);
```

Bind

You can "perma-bind" a function to a context:

```
fDance("tango");       // error -- this isn't the cat
```

```
fDance.call(fluffy, "tango");   // ok but tedious to always do

let betterDance = fDance.bind(fluffy);

betterDance("tango");  // ok -- bound so that `this` is Fluffy
```

`bind` is a method on functions that returns a bound copy of the function.

Binding Arguments

You can also bind arguments to functions. That will bake them into the function.

```
function applySalesTax(taxRate, price) {
  return price + price * taxRate;
}

// "null" for "this" means it doesn't matter what "this" is
const applyCASalesTax = applySalesTax.bind(null, 0.0725);
applyCASalesTax(50);  // 53.63
```

Where This Comes Up

Callback on Methods

Want to have object method as callback:

```
myBtn.addEventListener("click", fluffy.dance);
```

That won't work – browser will call ***dance*** on global object :(

```
myBtn.addEventListener("click", fluffy.dance.bind(fluffy));
```

That will work — when it calls that callback, it will always be on Fluffy!

Pre-Binding Calls

Imagine we want three buttons which call ***popUp***, but with different args:

demo/buttons-meh.html

```
<button id="a">A</button>
<button id="b">B</button>
<button id="c">C</button>
```

demo/buttons-meh.html

```
function popUp(msg) {
  alert("Secret message is " + msg);
}

function handleClick(evt) {
  let id = evt.target.id;

  if (id === "a") popUp("Apple");
  else if (id === "b") popUp("Berry");
  else if (id === "c") popUp("Cherry");
}

const get = document.getElementById.bind(document);

get('a').addEventListener("click", handleClick);
get('b').addEventListener("click", handleClick);
get('c').addEventListener("click", handleClick);
```

demo/buttons.html

```
function popUp(msg) {
  alert("Secret message is " + msg);
}

const get = document.getElementById.bind(document);

get('a').addEventListener("click", popUp.bind(null, "Apple"));
get('b').addEventListener("click", popUp.bind(null, "Berry"));
get('c').addEventListener("click", popUp.bind(null, "Cherry"));
```

Arrow Functions

Arrow functions don't make their own ***this***

demo/timeout.html

```
class Cat {
  constructor(name) {
    this.name = name;
  }

  superGreet() {
    alert(`#1: I am ${this.name}`);    // works, obvs

    setTimeout(function () {
      alert(`#2 I am ${this.name}`);  // ut oh
    }, 500);
```

```
    setTimeout(() => {
      alert(`#3 I am ${this.name}`);  // yay!
    }, 1000);
  }
}

let kitty = new Cat("Kitty");
kitty.superGreet();
```

Looking Ahead

- Additional OO Concepts
  - Class properties
  - Static methods
- Python OO
- Function-based JS “classes”