# JavaScript's `this`

A brownbag workshop at

MX®

by Seth House

@whiteinge
seth@eseth.com

## How `this` works in a single sentence

The `this` parameter is very important in object oriented programing, and its value is determined by the *invocation pattern*.

– Douglas Crockford, JavaScript the Good Parts (2008)

## Invocation patterns

> There are four patterns of invocation in JavaScript: the method invocation pattern, the function invocation pattern, the constructor invocation pattern, and the apply invocation pattern.

- `myObj.foo()`
- `foo()`
- `new Foo()`
- `foo.apply()` (and `call` and `bind`)

## Method invocation

```
var myObj = {
    myMeth: function() { console.log(this) },
};
myObj.myMeth();
```

```
var myObj = {
    mySubObj: {
        myMeth: function() { console.log(this) },
    },
};
myObj.mySubObj.myMeth();
```

## Function invocation

What will this log and why?

```
var myMeth = myObj.mySubObj.myMeth;
myMeth();
```

## `call` & `apply` invocation

```
// Map over list items:
Array.prototype.map.call(
    document.querySelectorAll('ul > li'),
    x => x.innerHTML)

// Reduce over form fields:
// <form onsubmit="processForm(ev, {})">...</form>
function getFormData(ev, seed) {
    return Array.prototype.reduce.call(ev.target, (acc, el) => {
        if (el.name && el.value !== '') {
            if (Array.isArray(acc[el.name])) {
                acc[el.name].push(el.value);
            } else {
                acc[el.name] = el.value;
            }
        }
        return acc;
    }, seed);
}
```

## `bind`

(Modern browsers pre-bind `console.log()`.)

```
var log = console.log.bind(console);
```

## `bind`

(Modern browsers pre-bind `console.log()`.)

```
var log = console.log.bind(console);
```

Capture `this` via closure (is there a difference?):

```
function foo() {
    var that = this;
    return function() {
        console.log(that);
    };
}
```

## An aside: Partial application

```
var ajax = (method, path, body) => { }
var get = ajax.bind(undefined, 'GET')
var getFoos = ajax.bind(undefined, 'GET', '/foo')
```

## Functional mixins

(by Angus Croll)

Generic utility mixin functions where `this` is *expected* to change!

```
function asAjax() {
    this.baseURL = 'https://api.github.com'
    this.fetch = function() {
        return fetch(this.baseURL + this.path).then(r ⇒ r.json())
    };
}
```

# Functional mixins (usage)

```
function Users() {
    this.path = '/users';

    this.methodFoo = function() { }
    this.methodBar = function() { }
}

function Organizations() {
    this.path = '/organizations';

    this.methodBaz = function() { }
}

asAjax.call(Users.prototype)
asAjax.call(Organizations.prototype)

var users = new Users()
users.fetch().then(x ⇒ console.log('users', x));

var organizations = new Organizations()
organizations.fetch().then(x ⇒ console.log('organizations', x));
```

## Arrow functions

Arrow functions don't use `this` at all!
They only capture lexical scope like with any other variable.

```
var myObj = {
    myMeth: () => { console.log(this) },
};

myObj.myMeth();
myObj.myMeth().call({foo: 'Foo'})
```