Function call forms and the value of `this`

Relevant Links

- Flanagan's book, section 8.2

Notes

There are 4 different ways that functions can be called, called "invocations". We describe them briefly here, and we will go deeper into them later. A big difference is how this is treated in each case. Recall that this is a reserved word that typically represents the object under question. But in some function calls it is not as clear what that should mean.

A further complication is whether the function is defined as an arrow function or not.

Here is a little snippet of code that attempts to showcase these possibilities:

```
let o = {
  v: 2,
  f: function() { console.log(this); },
  g: () => console.log(this)
};

o.f();      // prints the object o
o.g();      // prints the global object
let f=o.f;
let g=o.g;
f();        // prints the global object
g();        // prints the global object
new f();    // a newly created object using f as its constructor.
new g();    // an error
f.call(o); // prints whatever argument is passed
g.call(o); // prints the global object

o.h = function() { this.l = () => console.log(this) };
o.h();      // Makes sure "l" is created while o is the current object
o.l();      // prints the object o
```

So the rules are basically as follows:

- For **arrow functions**, the this object is determined **lexically**, i.e. it is the object that was current when the function was created. This is very helpful when you are trying to set up a callback for a user event.
- For **normal functions**, the this object is determined **dynamically**, and it depends on how the function is invoked, as follows:

  function invoc. ~ f(...) where f is a function.

  'this' set to the global object. CAREFUL!

method invoc. ~ m.f(...) where m is an object and f is a property of it with function value.

'this' set to 'm'.

constructor invoc. ~ new F(...) where F is a function. Constructors are by convention capitalized.

'this' set to a newly created object.

indirect invoc. ~ f.call(...), f.bind(...), f.apply(...).

'this' set to the first argument.

You need to be very careful when passing functions to some other part of your code, as you don't necessarily know how they are going to be called.

Indirect Invocations

The indirect invocations deserve further notice. There are mainly 3 functions:

`f.call`[1] Calls f with the first argument serving as this and any subsequent arguments passed as arguments to f.

e.g. f.call(null, 1, 2, 3);

`f.apply`[2] Calls f with the first argument serving as this and the second argument being an array of the arguments to be used in the call.

e.g. f.apply(null, [1,2,3]);

`f.bind`[3] Does not actually call f, but it returns a function which behaves like f except that it has "bound" the this object, and optionally has bound some number of arguments.

e.g. f.bind(o, 1, 2)(3, 4) is the same as f.call(o, 1, 2, 3, 4).

bind was really important before we had arrow functions, because it allowed us to bind the this object and still pass the function to someone who needs to do something with it, while making sure we retain ownership of the object that the function will be acting on.