I’m going to draw for you a diagram of what occurs when this code is interpreted by the JavaScript engine. Each line by line.

1. Draw dotted line - indicates stuff that has happened before we even get to line 1
2. Represent Objects as Squares and functions as Circles

There is a function which is called Object

And there is an object who the function Object is linked to

The link is called .prototype so its Object.prototype points at this object

On the object there is stuff like toString, valueOf, and several other things that you are already probably familiar with and built into the language. They call come from the Object.prototype guy

This exists at the beginning of every JavaScript program its part of the environment that gets created.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\*\***Now let's talk about line 1**

What is line 1 going to create for us?

**Line 1** is going to create a Function (circle) we give the label Foo

Also will create an object that we are linked to with same arbitrary name .prototype

ALL FUNCTIONS CREATE A CORRESPONDING OBJECT who can be accessed through .prototype

In addition to there being that .prototype connection the object has a property on it called .constructor which is a link going in the other direction

But there is this dual linkage here.

This is everything we get as of line 1.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\*\*\*\*We will skip over line 2 for now and come back to that. What happens with line 4?**

You’ll notice that we start adding properties to the Foo.prototype

We add an identify property to the object (put directly onto the object)

There is also a linkage from the object to the Object.prototype object all the way to the right

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\*\*\*\*Lets skip over line 5 we will come back to that, lets look at line 8.**

**Line 8** says new Foo()

When we use the keyword “new” there are 4 things that happen when it gets used with a function call:

1. Brand New object gets created
2. Object gets linked, linkage occurs we will explain in a moment
3. Context gets set to ‘this’ meaning when we call the Foo function the context of this will be pointing to our object which in this case is a1
   1. So when we execute line 2 and say this.me we put a property directly on the object a1
4. Returns this so gives us a label to our object a1

Who can walk me through what is going to happen on line 9?

(same exact thing)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\*\*\*Now lets look at line 11**

a2.speak

so where does that speak property get added?

Onto a2

Now at this point if I were to say something like a1.speak what would happen?

Its not there so we would get a reference error

Everyone follow me so far?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\*\*\*Lets go down to line 15**

a1.constructor - there is no constructor property on the object, OBJECT instances do not get a constructor property, PROTOTYPES do

so when we say a1.constructor and it doesn’t exist what happens?

it goes up the prototype chain until it finds it.

What we do is say?

* a1.constructor does not exist
* We then traverse the prototype chain and check to see if the next object up the prototype chain has a “constructor” property

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\*\*\*\*Now what about line 17? We reference this random property called \_\_proto\_\_**

we call this dunder proto.

Guess what that function call does? returns the internal protoype linkage of whatever the this binding is (a1 in this case). So on line 17 we can pretend like we are calling a function with parens and the this binding will be on a1

\_\_proto\_\_ is a public property that references an internal link

So now a1.\_\_proto\_\_ points at the object that is the same thing as Foo.prototype

and both a1 and a2 point to the same object.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\*\*\*What if I call a1.identify what happens then?**

Its going to check a1 for an indentify property and there is not one so it will go up the prototype link to the next object and check to see if there is an identify function there is so it uses that one

Notice how there is only one copy of the function, on line 5 references the this keyword. will be a1, a2 or whatever it is. a2.identify etc.

This keyword becomes a very nice advantage we have when dealing with the prototype chain