Welcome Back

Class 06

Objectives

- Learn what objects are
- Learn about object constructors and prototypes
- Implement and interface with JSON (Javascript Object Notation)

What is an object?

- Objects are representations of data
- Everything in Javascript is considered an object (strings, arrays, hashes, etc)
- Objects have 2 defining characteristics:
  - They posses “state” (variables, information, etc)
  - The exhibit “behavior” (they *do* things, display alerts, email people, etc)

We’ve already been using objects!

- A string is an object!

- For example:

```
var myString = "Robert";
// Strings have state...
myString.length
//=> 6

// And they have behavior
myString.toUpperCase()
//=> "ROBERT"
```

You can think of most things as objects

- A “**Person**” object could be a user-defined object you make in your programs

- **State**: (Name, Birthday)

- **Behavior**: (Speak, Type, Laugh)

Defining a simple object

- To define a simple object in Javascript, you can use curly braces.
- Like so: `var myObject = {};`
- This defines a variable with the value as an Object.
- The object doesn’t have any state, nor behavior. It’s essentially a blank object.

Creating an object with state

- Objects in javascript have “keys”. These keys either contain a certain value (state) or point to a function (behavior)

- To define an object wit a key called “age”, you can do:

  ```
  var myPerson = { age: 26 };
  ```

- This defines a new object with 1 key called “age” set to an integer, 26.

Accessing a key on an object

- Given the object:

```
var myPerson = { age: 26 };
```

- You can access the age by using a period and then the name of the key. For example:

```
var myPerson = { age: 26 };
myPerson.age
// => 26
```

Setting a key on an existing object

- If you have an object that has already been initialized, you may want to set a key on it after you've declared it.

- To do this, you can set keys just like variables, with a touch more syntax.

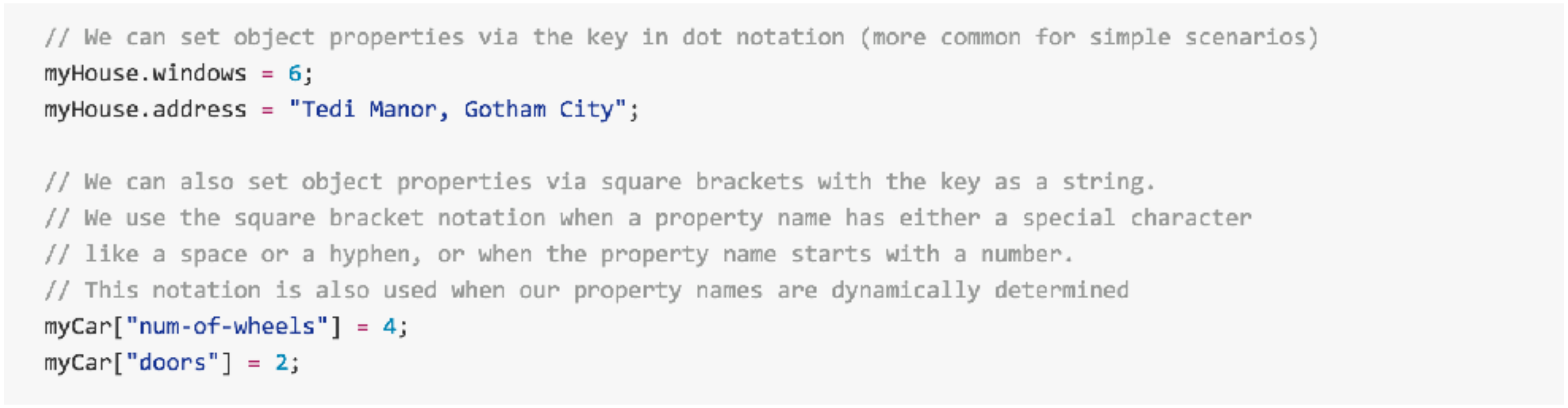
```
var myPerson = { age: 26 };
myPerson.name = "Robert Ross";
// => { age: 26, name: 'Robert Ross' }
```

Setting object properties

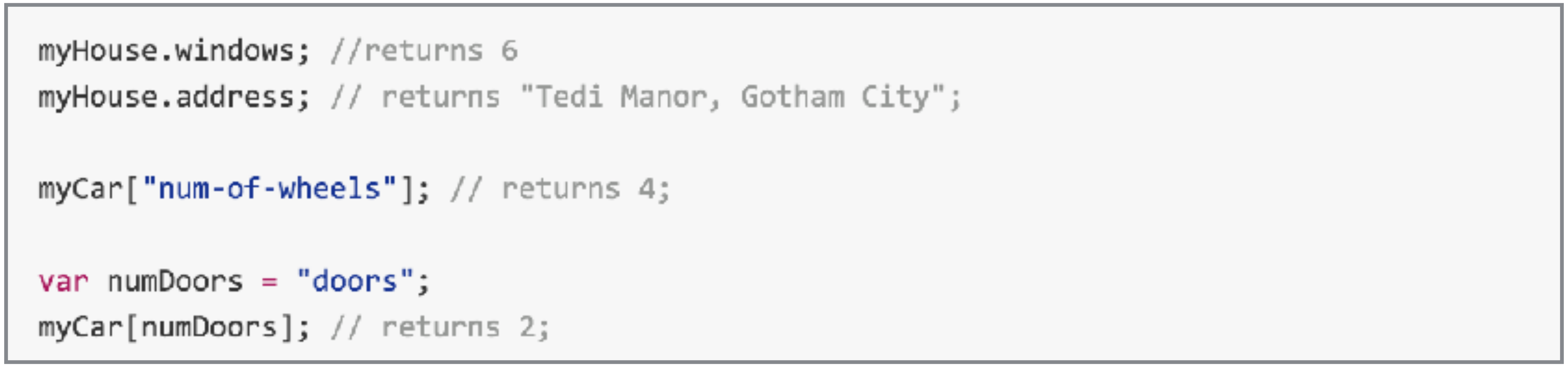
```
var myHouse = {};
```

```
// We can set object properties via the key in dot notation (more common for simple scenarios)
myHouse.windows = 6;
myHouse.address = "Tedi Manor, Gotham City";

// We can also set object properties via square brackets with the key as a string.
// We use the square bracket notation when a property name has either a special character
// like a space or a hyphen, or when the property name starts with a number.
// This notation is also used when our property names are dynamically determined
myCar["num-of-wheels"] = 4;
myCar["doors"] = 2;
```

Accessing object properties



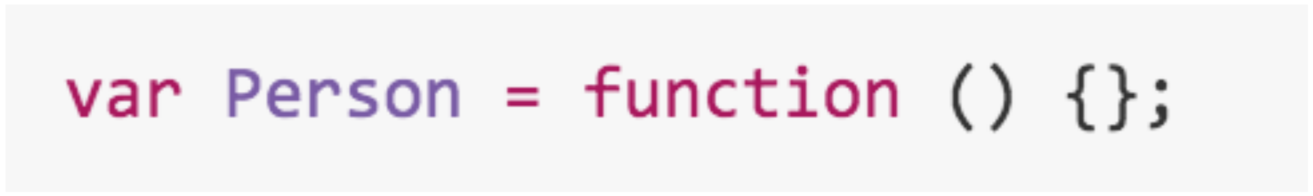
```
// We can set object properties via the key in dot notation (more common for simple scenarios)
myHouse.windows = 6;
myHouse.address = "Tedi Manor, Gotham City";

// We can also set object properties via square brackets with the key as a string.
// We use the square bracket notation when a property name has either a special character
// like a space or a hyphen, or when the property name starts with a number.
// This notation is also used when our property names are dynamically determined
myCar["num-of-wheels"] = 4;
myCar["doors"] = 2;
```



```
myHouse.windows; //returns 6
myHouse.address; // returns "Tedi Manor, Gotham City";

myCar["num-of-wheels"]; // returns 4;

var numDoors = "doors";
myCar[numDoors]; // returns 2;
```

Node Along

Another way to make objects

- By convention, the way to create an object is with a function called a constructor. This is really a JavaScript function like any other, but when you call it in a particular way JavaScript does some magic under the hood for you.
- 

```
var Person = function () {};
```

- The object We're familiar with the `new Object()` syntax from our first example today. We create an instance of our "Person" class in a similar way.
- `var clark = new Person();`

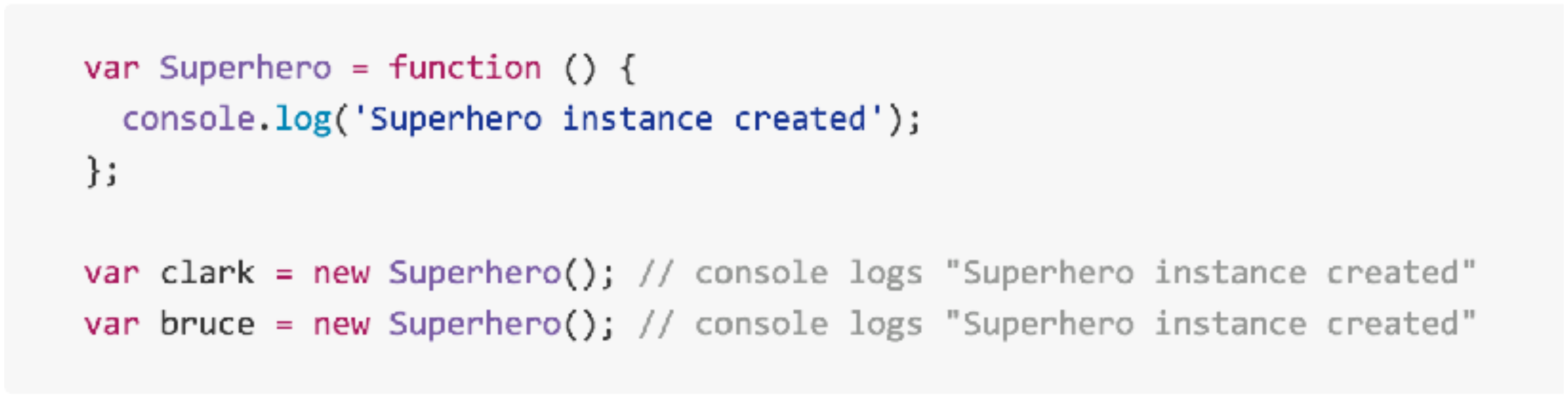
Constructors

- The constructor is called the moment you create a new object. More commonly called “instantiating”.

- These are useful for setting initial values.

- Any code within the function that you are instantiating will be run automatically for you (as if you were calling the function).

A Superhero constructor



```
var Superhero = function () {
  console.log('Superhero instance created');
};

var clark = new Superhero(); // console logs "Superhero instance created"
var bruce = new Superhero(); // console logs "Superhero instance created"
```

Whenever we call “`new Superhero()`”, we see that it logs to the console “Superhero instance created”.

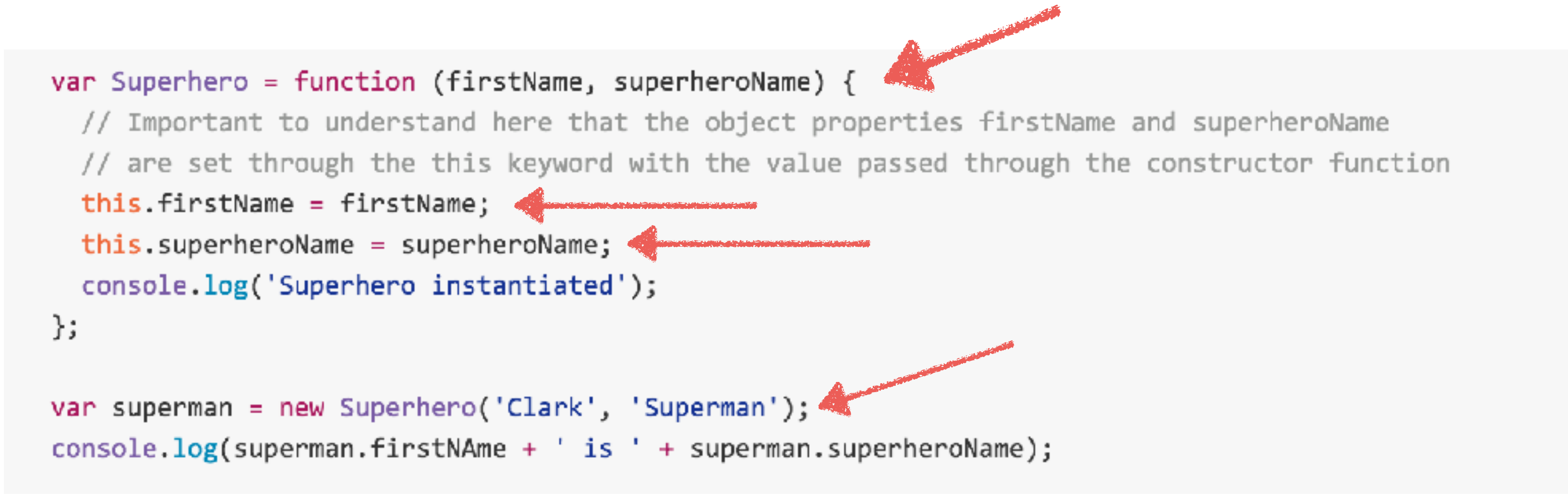
Making constructors more useful

- The point of objects is to create blueprints for a common entity. Think back to the exercise we did last class identifying different types of objects.

- It’s common to see constructors used to set initial properties on the newly instantiated object.

- Properties can be set in the constructor, so they are set specifically for each instance. This simply means that we pass them as parameters in our constructor function.

Setting properties on instantiation



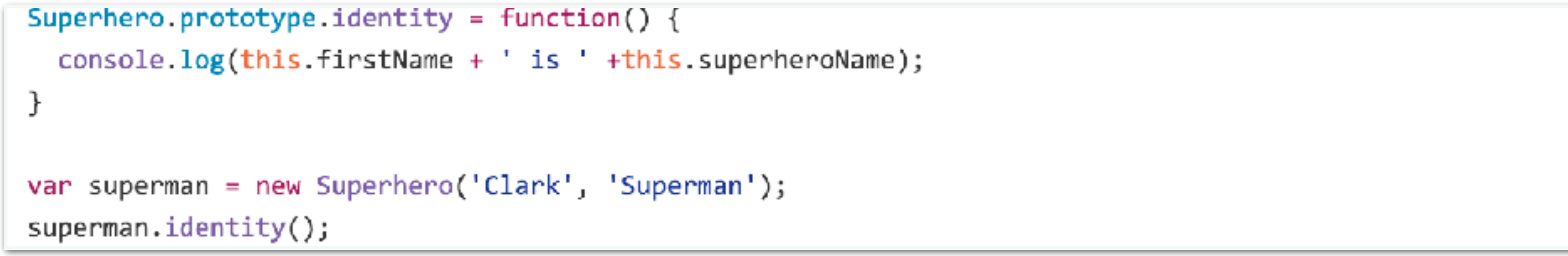
```
var Superhero = function (firstName, superheroName) {
  // Important to understand here that the object properties firstName and superheroName
  // are set through the this keyword with the value passed through the constructor function
  this.firstName = firstName;
  this.superheroName = superheroName;
  console.log('Superhero instantiated');
};

var superman = new Superhero('Clark', 'Superman');
console.log(superman.firstNAme + ' is ' + superman.superheroName);
```

That “**this**” keyboard you see is referencing the <u>current</u> object. So we’re setting the **firstName** and **superheroName** on the new superhero object.

Adding methods to your objects

- Methods are functions that are attached to an object.
- They have access to the objects properties and can set them too.
- Used to add behavior to your custom objects.

For example



```
Superhero.prototype.identity = function() {
  console.log(this.firstName + ' is ' +this.superheroName);
}

var superman = new Superhero('Clark', 'Superman');
superman.identity();
```

This code is adding an “identity” method to the superhero object. When it is invoked, it prints the properties of **firstName** and **superheroName.**

But what the smurf is that “`prototype`” mumbojombo?

To add a method to an object

- Javascript is a **prototype**-oriented language.
- When adding a method to an object, you must add it to the “prototype” key on the object’s name.
- When an object is constructed (using the **new** keyword), the methods on the prototype are inherited onto the new object.

Prototype

- Every object in JavaScript has a prototype, connected to the constructor that created it.
- What this means is that if you make a new object called Superhero, it sets its prototype automatically to “Object”.
- This allows the Superhero object to have the properties and methods on the “Object” object (top-level provided by the browser), and it’s own properties and methods.
- We cover this in-depth later on in class. For now, we can just think “use the prototype property to add methods”.

Using methods to set properties

- It’s common to to create methods that change properties on your objects.
- In these methods you can perform validations and such to ensure the data being set is allowed.
- For instance, an attribute containing a person's age in years should never contain a negative number, and will rarely contain a number over 100

Mini Goal

- I want an object to represent a *person*.
- I want the person to have a *name and age*
- I want to be able to ask the object if the person can *drive*

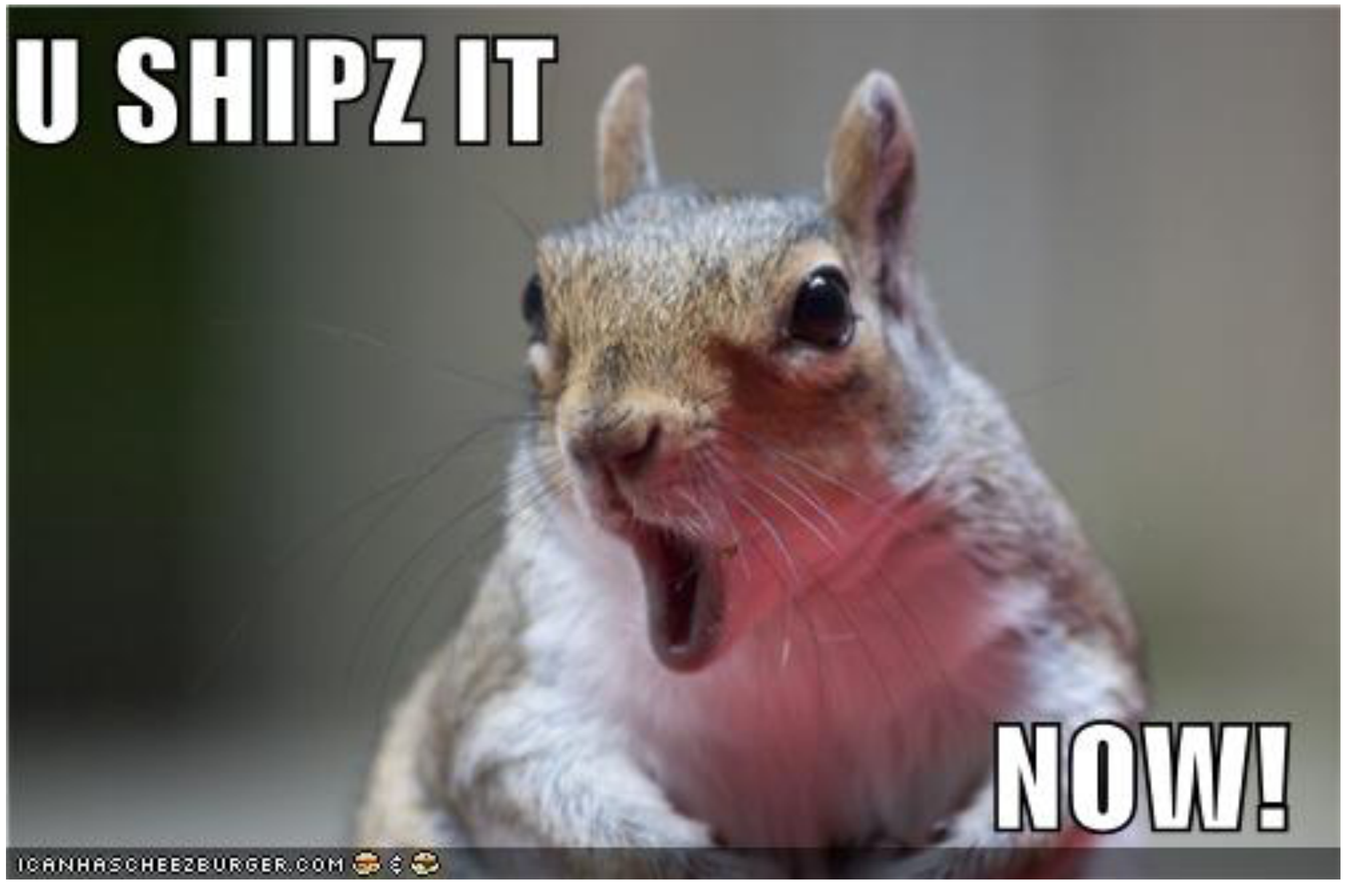
Mini object

```
// Make our object constructor for the "Person" object
function Person(name, age) {
  this.name = name;
  this.age = age;
}

// Add our "canDrive" method to our Person object
Person.prototype.canDrive = function() {
  return this.age >= 16
}

var myPerson = new Person("Robert", 26);
myPerson.canDrive();
//=> true
```

Method Example

```javascript
Person.prototype.setAge = function (newAge) {
  if (newAge < 0) {
    console.error("A person cannot be negative years old!");
  }
  else if (newAge > 150) {
    console.error("People do not generally live to the age of 150");
  }
  else {
    this.age = newAge;
  }
};

Person.prototype.age = function () {
  return this.age;
};
```

Code Along



Monkey Exercise

Partner up, and create a file called “`monkey.js`” in `~/GA-JS/06/`

- name
- species
- foodsEaten

And the following methods:

- eatSomething(thingAsString) that prints out “thingAsString” and the name of the monkey.
- introduce: producers a string introducing itself, including its name, species, and what it's eaten.

Create 3 monkeys total. Make sure all 3 monkeys have all properties set and methods defined.

Exercise your monkeys by retrieving their properties and using their methods. Practice using both syntaxes

Break

Intro to JSON

- JSON (JavaScript Object Notation) is a lightweight text-based data format that's based on JavaScript (specifically, a subset of Standard ECMA-262 3rd Edition - December 1999).

- Because it's text, and it looks like JavaScript, JSON is simultaneously both easy for humans to read and write AND easy for programs to parse and generate.

- Think of it was a well written essay that follows every english rule. Javascript can parse it very easily by following the rules.

Intro to JSON

- We use JSON objects to transfer data between applications and Javascript.

- To keep everything consistent, all JSON code must follow a number of strict conventions (stricter even than normal JavaScript!)

JSON Syntax Rules

- Property names must be double-quoted strings.
- Trailing commas are forbidden.
- Leading zeroes are prohibited.
- In numbers, a decimal point must be followed by at least one digit.
- Most characters are allowed in strings; however, certain characters (such as ', ", \, and newline/tab) must be 'escaped' with a preceding backslash (\) in order to be read as characters (as opposed to JSON control code).
- All strings must be double-quoted.
- No comments!