## **OBJECTS AND OBJECT CONSTRUCTORS**

### **Introduction**

In our JavaScript fundamentals course, you should have learned the basics of using objects to store and retrieve data. Let’s start with a little refresher.

There are multiple ways to define objects but in most cases, it is best to use the object literal syntax as follows:

const myObject = { property: 'Value!', otherProperty: 77, "obnoxious property": function() { // do stuff! } }

There are also 2 ways to get information out of an object: dot notation and bracket notation.

// dot notation myObject.property // 'Value!' // bracket notation myObject["obnoxious property"] // [Function]

Which method you use will depend on context. Dot notation is cleaner and is usually preferred, but there are plenty of circumstances when it is not possible to use it. For example, myObject."obnoxious property" won’t work because that property is a string with a space in it. Likewise, you can not use variables in dot notation:

const variable = 'property' myObject.variable // this gives us 'undefined' because it's looking for a property named 'variable' in our object myObject[variable] // this is equivalent to myObject['property'] and returns 'Value!'

If you are feeling rusty on using objects, now might be a good time to go back and review the content in Fundamentals 5 from our JavaScript Basics course.

### **Learning Outcomes**

By the end of this lesson, you should be able to do the following:

* Write an object constructor and instantiate the object.
* Describe what a prototype is and how it can be used.
* Explain prototypal inheritance.
* Understand the basic do’s and don’t’s of prototypical inheritance.
* Explain what Object.create does.

### **Objects as a Design Pattern**

One of the simplest ways you can begin to organize your code is by simply grouping things into objects. Take these examples from a ‘tic tac toe’ game:

// example one const playerOneName = "tim" const playerTwoName = "jenn" const playerOneMarker = "X" const playerTwoMarker = "O" // example two const playerOne = { name: "tim", marker: "X" } const playerTwo = { name: "jenn", marker: "O" }

At first glance, the first doesn’t seem so bad.. and it actually takes fewer lines to write than the example using objects, but the benefits are huge! Let me demonstrate:

function printName(player) { console.log(player.name) }

This is something that you just could NOT do with the example one setup. Instead, every time you wanted to print a specific player’s name you would have to remember the correct variable name and then manually console.log it:

console.log(playerOneName) console.log(playerTwoName)

Again, this isn’t *that* bad… But what if you *don’t know* which player’s name you want to print?

function gameOver(winningPlayer){ console.log("Congratulations!") console.log(winningPlayer.name + " is the winner!") }

Or, what if we aren’t making a 2 player game, but something more complicated such as an online shopping site with a large inventory? In that case, using objects to keep track of an item’s name, price, description and other things is the only way to go. Unfortunately, in that type of situation manually typing out the contents of our objects is not feasible either. We need a cleaner way to create our objects, which brings us to…

### **Object Constructors**

When you have a specific type of object that you need to duplicate like our player or inventory items a better way to create them is using an object constructor, which is a function that looks like this:

function Player(name, marker) { this.name = name this.marker = marker }

and which you use by calling the function with the keyword new.

const player = new Player('steve', 'X') console.log(player.name) // 'steve'

Just like with objects created using the Object Literal method you can add functions to the object:

function Player(name, marker) { this.name = name this.marker = marker this.sayName = function() { console.log(name) } } const player1 = new Player('steve', 'X') const player2 = new Player('also steve', 'O') player1.sayName() // logs 'steve' player2.sayName() // logs 'also steve'

### **Exercise**

Write a constructor for making “Book” objects. We will revisit this in the project at the end of this lesson. Your book objects should have the book’s title, author, the number of pages, and whether or not you have read the book

Put a function into the constructor that can report the book info like so:

theHobbit.info() // "The Hobbit by J.R.R. Tolkien, 295 pages, not read yet"

note: it is almost *always* best to return things rather than putting console.log() directly into the function. In this case, return the info string and log it after the function has been called:

console.log(theHobbit.info());

### **The Prototype**

Before we go much further, there’s something important you need to understand about JavaScript objects. All objects in JavaScript have a prototype. Stated simply, the prototype is another object that the original object *inherits* from, which is to say, the original object has access to all of its prototype’s methods and properties.

This concept is an important one, so you’ve got some reading to do. Make sure you really get this before moving on!

1. [This article](https://web.archive.org/web/20200513181548/https://javascriptissexy.com/javascript-prototype-in-plain-detailed-language/) is a straightforward introduction and demonstration of the concept. It also covers constructors again.. good time for a review! The important bits here, once you’ve covered the basics are ‘Prototype-based inheritance’ and the ‘Prototype chain’
2. To go a bit deeper into both the chain and inheritance spend some time with [this great article](http://javascript.info/prototype-inheritance). As usual, doing the exercises at the end will help cement this knowledge in your mind. Don’t skip them! Important note: this article makes heavy use of \_\_proto\_\_ which is not generally recommended. The concepts here are what we’re looking for at the moment. We will soon learn another method or two for setting the prototype.

If you’ve understood the concept of the prototype then this next bit about constructors will not be confusing at all!

function Student(name, grade) { this.name = name this.grade = grade } Student.prototype.sayName = function() { console.log(this.name) } Student.prototype.goToProm = function() { // eh.. go to prom? }

If you’re using constructors to make your objects it is best to define functions on the prototype of that object. Doing so means that a single instance of each function will be shared between all of the Student objects. If we declare the function directly in the constructor like we did when they were first introduced, that function would be duplicated every time a new Student is created. In this example, that wouldn’t really matter much, but in a project that is creating thousands of objects, it really can make a difference.

#### **Recommended Method for Prototypal Inheritance**

So far you have seen several ways of making an object inherit the prototype from another object. At this point in history, the recommended way of setting the prototype of an object is Object.create ( [here](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create) is the documentation for that method.) Object.create very simply returns a new object with the specified prototype and any additional properties you want to add. For our purposes you use it like so:

function Student() { } Student.prototype.sayName = function() { console.log(this.name) } function EighthGrader(name) { this.name = name this.grade = 8 } EighthGrader.prototype = Object.create(Student.prototype) const carl = new EighthGrader("carl") carl.sayName() // console.logs "carl" carl.grade // 8

You can probably figure out what’s going on here. After creating the constructor for EighthGrader we set it’s prototype to a new object that has a copy of Student.prototype.

A warning… this doesn’t work:

EighthGrader.prototype = Student.prototype

because it will literally set EighthGrader’s prototype to Student.prototype (i.e. not a copy), which could cause problems if you want to edit something in the future. Consider one more example:

function Student() { } Student.prototype.sayName = function() { console.log(this.name) } function EighthGrader(name) { this.name = name this.grade = 8 } // don't do this!!! EighthGrader.prototype = Student.prototype function NinthGrader(name) { this.name = name this.grade = 9 } // noooo! not again! NinthGrader.prototype = Student.prototype NinthGrader.prototype.sayName = function() {console.log("HAHAHAHAHAHA")} const carl = new EighthGrader("carl") carl.sayName() //uh oh! this logs "HAHAHAHAHAHA" because we edited the sayName function!

If we had used Object.create in this example then we could safely edit the NinthGrader.prototype.sayName function without changing the function for EighthGrader as well.

### **[Additional Resources](https://www.theodinproject.com/paths/full-stack-javascript/courses/javascript/lessons/objects-and-object-constructors#additional-resources)**

This section contains helpful links to other content. It isn’t required, so consider it supplemental if you need to dive deeper into something.

* [This article](https://dev.to/lydiahallie/javascript-visualized-prototypal-inheritance-47co) from Lydia Hallie and [This video](https://www.youtube.com/watch?v=sOrtAjyk4lQ) from Avelx explains the Prototype concept with graphics and simple language. Try using these resources if you want another perspective to understand the concept.
* [This Video](https://www.youtube.com/watch?v=CDFN1VatiJA) from mpj explains the Object.create method with great details about it, he walks through what it is, why Object.create exists in JavaScript, and how to use Object.create. Also you can check [This Video](https://www.youtube.com/watch?v=MACDGu96wrA) from techsith to understand another point of view of extending objects from others by Object.create.
* [The Principles of Object-Oriented JavaScript](https://www.amazon.com/Principles-Object-Oriented-JavaScript-Nicholas-Zakas/dp/1593275404) book by Nicholas C. Zakas is really great to understand OOP in javascript, which explains concepts simply and in-depth, which explores JavaScript’s object-oriented nature, revealing the language’s unique implementation of inheritance and other key characteristics. It's not free but it’s very valuable.