JavaScript Objects and Functions: An Overview

Because JavaScript is an object-oriented programming language, I need to tell you a little bit about JavaScript "objects" and the structure of the language generally, before we can actually start programming in JavaScript specifically.

In any programming language, you write lines of code which are executed, left-to-right, top-to-bottom, just like in HTML. In an object-oriented programming language such as JavaScript, however, much of this code is organized in OBJECTS. In fact, most of the built-in features of JavaScript use objects of one sort or another, which means that you need to understand basic object syntax before you can do much of anything, including reading the JavaScript reference books.

In an HTML page in a JavaScript-enabled web browser, most or even all things on the page are mirrored in JavaScript objects. There is a window object which refers to the browser window itself, there is a document object which refers to the HTML page itself (and, potentially, everything contained in that HTML page), there is an images object which has to do with pictures, etc. By manipulating these objects in JavaScript code, you can affect the behavior and appearance of an HTML page, call up dialog boxes or other resources, and invoke a fairly sophisticated set of commands to do your bidding.

Every object in JavaScript has two main components: a set of PROPERTIES, and a set of METHODS. Properties are generally passive in nature, describing the state of some feature of the object. Methods, on the other hand, are generally active, performing some sort of task.

Over the years, I have used something called the "object-oriented car", a common metaphor in programming instruction, to explain object-oriented programming principles. As I have grown more sophisticated in my explanations, however, the "car" object example has begun to break down. Now, instead of describing the entire car in my lectures, I describe only the STEERING WHEEL of the car using object-oriented programming terms.

Let us create a SteeringWheel object (and notice how the name of the object has NO SPACES!). This SteeringWheel object has one property, "direction", and one method, "honk()".

The direction property of the SteeringWheel object says what direction the SteeringWheel is pointed. Since this is a VERY simple SteeringWheel, only left, right, or center would be legal values for this property.

The honk() method of the SteeringWheel object honks the car's horn; we'll talk more about this method later on.

The code that defines the SteeringWheel object, with all of its properties and methods, is called a CONSTRUCTOR. The exact syntax for creating your own object constructor is really beyond the scope of this introductory lecture. However, you WILL be invoking pre-built constructors when you use JavaScript, so we'll talk about how to do this.

The SteeringWheel constructor merely defines the basic structure of the SteeringWheel object; it does NOT actually create a SteeringWheel which you can manipulate directly. To get a SteeringWheel object that you can use, you need to create an INSTANCE of the SteeringWheel object (also known as INSTANTIATING a SteeringWheel object). When you create an instance of an object, you are creating ONE copy of the object which you have control over.

In JavaScript, you will need to instantiate an object by using a variable to contain a copy of the object. First, you'll declare a variable to hold the instance of the SteeringWheel object; then you'll initialize the variable as a "new" SteeringWheel object using the "new" keyword.

Example:

var theWheel = new SteeringWheel();

Whenever you instantiate an object, you MUST use the keyword, new, before the call to the object constructor, as you see above. Also, notice the opening/closing parentheses characters following the call to the SteeringWheel constructor (and prior to the semi-colon end-of-line marker); these parentheses are called the "function call" operator (which calls/invokes a function), and are an essential part of the object instantiation process.

A FUNCTION, in JavaScript, is a collection of lines of code, a program, which does something (like switch an image, or pop up a new window, or BUILD AN OBJECT, etc). In order to execute a function, to make a function "run" in JavaScript, you must state the name of the function, followed by the "function call" operator (the opening/closing parentheses characters); this will cause the function to execute when your code is executed. Note: we'll talk about making our OWN functions a little later on in this module.

It turns out that an object constructor, such as the SteeringWheel object constructor in the above example, is made in JavaScript out of a FUNCTION. The same function syntax that you will use to create miniature programs later on is also used to make object constructors!

Just as a side note, it turns out that FUNCTIONS are also OBJECTS themselves; this is because almost everything in JavaScript is built out of objects. I'm sure that this seems confusing now, but don't worry about it; as you get deeper into JavaScript, over time, it will all start to make much more sense to you.

Here's the code again:

var theWheel = new SteeringWheel();

Whenever you instantiate an object, you will DECLARE a variable, then INITIALIZE that variable to be a NEW instance of whatever OBJECT CONSTRUCTOR you want. An object constructor is really a FUNCTION, which you must execute in order to create your object; this is done by stating the name of the object constructor function (in this case, SteeringWheel), followed by the function call operator (the parentheses characters). Again, the function call operator is necessary in order to execute the object constructor function. As in all lines of code in JavaScript, you must end the line with a semi-colon end-of-line marker.

Once you have created an instance of your desired object, you may access the PROPERTIES and METHODS of the object by using DOT SYNTAX.

Dot syntax is a way of accessing child elements of a parent object (remember parents and children?) by using period/dot (.) characters between parent and child elements to indicate a path through a complex tree of relationships, much as the slash (/) character is used in URL syntax to indicate a path through a complex directory structure on a web server.

Here's a much simpler example. If I wanted to put $10.00 cash into an empty cash register in my local coffee house in Guerneville, CA, USA, using JavaScript dot syntax to indicate the path to the cash register, the line of JavaScript code might look something like this:

california.sonomaCounty.guerneville.cinnabarCoffee.cashRegister = 10;

Dot syntax always creates a path from the most general to the most specific element, reading from left-to-right, and, as you can see from the above example, there are NO SPACES IN THE NAMES of the elements or between the dots!

Back to the instance of our SteeringWheel object, theWheel:

The SteeringWheel object has one property, direction. Object properties can be accessed via the instance of the object using dot syntax. In the following examples, I am assuming that theWheel has already been declared and instantiated.

Example:

theWheel.direction = "left";

In the above example, I have set the direction property of theWheel object equal to the string, "left". Surprise! An object property is really a VARIABLE which is attached to a particular object; this property may be filled with whatever data we like (in this case, a string).

Having executed the above line of code, theWheel is now pointed towards the left.

Example:

theWheel.direction = "right";

Now, theWheel is pointed towards the right.

Note: At this point, students often point out to me that a property is supposed to be static, yet the direction property of theWheel seems to be turning the steering wheel of the car rather than merely reflecting which direction theWheel is pointing! Well, oftentimes object properties are not as static as I have led you to believe (remember, I said "usually" static!). In a real JavaScript object, however, I would probably have written a method called "turnWheel()" which actually turned the wheel, and the direction property would merely have been a string which indicated the current direction of the wheel.

Please cut me some slack, I'm trying to make a simple example here!

Now, if I wanted to honk the horn of theWheel, I would need to call the honk() method of theWheel.

A METHOD is really a FUNCTION which is attached to an object, just as a PROPERTY is really a VARIABLE attached to an object (again, a function is a program which performs some action)! Therefore, I use the function call operator (opening/closing parentheses) after the name of the method (and dot syntax, of course).

Example:

theWheel.honk();

The above example executes the honk() method of theWheel, causing theWheel's horn to honk.

Example (complete code):

var theWheel = new SteeringWheel();

theWheel.direction = "left";

theWheel.direction = "right";

theWheel.honk();

Boy, if we drove like this down the road, we'd probably get a ticket for reckless driving!

You can create more than one instance of an object, simply by declaring and instantiating more copies of the object.

Example:

var theWheel = new SteeringWheel();

var theSecondWheel = new SteeringWheel();

theWheel.direction = "left";

theSecondWheel.direction = "center";

theWheel.honk();

In the above example, there are two copies of the SteeringWheel object (hopefully attached to two separate cars!). The direction property of theWheel is set to "left", while the direction property of theSecondWheel is set to "center". Then, I honked the horn of theWheel, but I left theSecondWheel silent.

Each instance of an object is separate, and contains separate information. Even though both objects in the above example are SteeringWheel objects, they are completely independant from one another!

I can make as many instances of an object as I want, but I think TWO SteeringWheel objects is enough for now...