Week 9

Objects in JavaScript

In JavaScript, we can create custom objects. In many languages, this is accomplished using a *class* definition. JavaScript, however, does not recognize the concept of classes; in JavaScript, we define new objects just like we would define a standard function.

Here is an example of creating a Person Object Type:

```
var PersonType = function() {
    // implementation code goes here
}
```

The above example doesn't do anything yet. Objects contain *data*(attributes, property, or *state*) and Objects contain *functionality* (methods). So far, this object contains neither.

In order to implement state, we simply add property variables inside this function identically to how we would define local variables within any standard function.

Example

```
var PersonType = function() {
    // Properties
    this.FirstName = "";
    this.LastName = "";
    this.Age = 0;
}
```

One minor difference here is that we use the **this** keyword to indicate to JavaScript that these are *instance* variables. That is, they specifically represent data that is intended to be used for an instance of an object created from this *prototype* or *template*.

So now this Object has some variables to hold state data, but it has no functionality yet.

We can add functionality by adding one or more nested functions within the **PersonType** definition.

Example

```
var PersonType = function() {
    // Properties
    this.FirstName = "";
    this.LastName = "";
    this.Age = 0;

    this.sayHello = function() {
        alert("Hello World, my name is " + this.FirstName + " " + this.LastName);
    }
}
```

Again, we're using the **this** keyword to specify that we are referring to instance members.

Now that we have a simple object type definition, the next step is to actually create one or more object instances from this prototype/template. In JavaScript, we do this using the **new** keyword.

Example

```
var PersonType = function() {
    // Properties
    this.FirstName = "";
    this.LastName = "";
    this.Age = 0;

    this.sayHello = function() {
        alert("Hello World, my name is " + this.FirstName + " " + this.LastName);
    }
}

// Create first instance of a PersonType
var myFirstPerson = new PersonType();
myFirstPerson.FirstName = "Bob";
myFirstPerson.LastName = "Smith";
myFirstPerson.Age = 23;
myFirstPerson.sayHello();

// Create second instance of a PersonType
var mySecondPerson = new PersonType();
mySecondPerson.FirstName = "Ronda";
mySecondPerson.LastName = "Rodriguez";
mySecondPerson.Age = 44;
mySecondPerson.sayHello();
```

This first example is very simple but it works well enough to get you started. Try it out!

Object Constructors

In object-oriented programming, a *constructor* method is a method that gets called automatically when an instance of an object is created from your class template. Constructor methods are not unique to JavaScript, but are, rather, standard fare in the object-oriented programming world.

Typically, the constructor methods are used to initialize data elements within an object instance.

In most object-oriented programming languages, you create a *constructor method* as a separate function within the class definition. A constructor method will often have a special name that indicates its purpose. For example, in some languages any method named **main** will act as the constructor. In Python, a constructor method must have the name **init** (using two underscores before and after - hence this is sometimes referred to with the coinage "*dunder*", as in "dunder-init.").

In JavaScript, we don't have to create any special internal functions or methods to act as the constructor. In JavaScript, the original *class function definition* is itself the constructor. That is, you don't need to create a separate function definition to get code to fire when instances are instantiated from your class; you just add

the necessary code inline, right within the actual class/prototype definition itself.

For example, if I want code to automatically fire when someone instantiates an instance of our **PersonType** class that we defined above, I can just include code in the main class definition function as follows:

Example

```
var PersonType = function() {
    // Properties
    this.FirstName = "";
    this.LastName = "";
    this.Age = 0;

    // This code will automatically fire when someone creates a new PersonType obejct.

    // Reminder: In JavaScript, I don't need to create a special internal
    // function to act as the constructor.
    console.log("New PersonType object instance just created");

    this.FirstName = "foo";
    this.LastName = "bar";
    this.Age = 28;

    this.sayHello = function() {
        alert("Hello world, my name is " + this.FirstName + " " + this.LastName);
    }
}
```

Passing Parameters to the Constructor

We usually will want to pass parameters to the constructor method so that those parameters can be used to provide the instance with its initial state data.

We do this by passing in the parameters when we call **new** on an Object type.

Example

```
var PersonType = function(FName, LName, TheAge) {
    // Properties
    this.FirstName = FName;
    this.LastName = LName;
    this.Age = TheAge;

    // This code will automatically fire when someone creates a new PersonType object.
    console.log("New PersonType object instance just created");

    this.sayHello = function() {
        alert("Hello world, my name is " + this.FirstName + " " + this.LastName);
    }
}

// Pass initial state data into the constructor as parameters.
var myFirstPerson = new PersonType("Bob", "Smith", 23);
```

```
myFirstPerson.sayHello();

// Pass initial state data into the constructor as parameters.
var mySecondPerson = new PersonType("Ronda", "Rodriguez", 44);
mySecondPerson.sayHello();
```

We obviously would want to build more useful funcitonality in our **PersonType**. Here is an example:

Example

```
var PersonType = function(FName, LName, Street, City, State, Zip, Phone, TheAge) {
    // Properties
    this.FirstName = FName;
    this.LastName = LName;
    this.Street = Street;
    this.City = City;
    this.State = State;
    this.Zip = Zip;
    this.Phone = Phone;
    this.Age = TheAge;

    // This code will fire automatically when someone creates a new PersonType object.
    console.log("New PersonType object instance just created");

    this.sayHello = function() {
        alert("Hello world, my name is " + this.FirstName + " " + this.LastName);
    }

    this.Validate = function() {
        // Code would go here to validate all the input data from the customer...
        if (customerDataValid) {
            return true;
        } else {
            return false;
        }
    }

    this.Update = function(newFname, newLname, newStreet, newCity,
                           newState, newZip, newPhone, newAge) {
        this.FirstName = newFname;
        this.LastName = newLname;
        this.Street = newStreet;
        this.City = newCity;
        this.State = newState;
        this.Zip = newZip;
        this.Phone = newPhone;
        this.Age = newAge;
    }

    this.Delete = function() {
        // Code to delete customer data goes here.
    }
}
```

Here is another real-world example:

Example

```
function browserType() {
    this.browserName;
    this.majorVersion;
    this.minorVersion;
    this.isIE;
    this.isNetscape;
    this.isChrome;
    this.isOpera;
    this.isFirefox;
    this.isSafari;
    this.isEdge;
    this.isWindows;
    this.isMac;
    this.isBlink;
    this.opSystem;

    this.browserName = navigator.appName;
    this.majorVersion = navigator.appVersion;
    this.minorVersion = "0";
    this.opSystem = navigator.platform;

    // For Netscape
    if (this.browserName.indexOf("Netscape") != -1) {
        this.isNetscape = true;
    } else {
        this.isNetscape = false;
    }

    // For Opera 8.0+
    this.isOpera = (!!window.opr && !!opr.addons) ||
                   !!window.opera ||
                   navigator.userAgent.indexOf('OPR/') >= 0;

    // Firefox 1.0+
    this.isFirefox = typeof InstallTrigger !== 'undefined';

    // At least Safari 3+: "[object HTMLElementConstructor]"
    this.isSafari = Object.prototype.toString.call(window.HTMLElement).indexOf('Constructo

    // Internet Explorer 6-11
    this.isIE = /*@cc_on!@*/false || !!document.documentMode;

    // Edge 20+
    this.isEdge = !this.isIE && !!window.StyleMedia;

    // Chrome 1+
    this.isChrome = !!window.chrome && !!window.chrome.webstore;

    // Blink engine detection
    this.isBlink = (this.isChrome || this.isOpera) && !!window.CSS;
}
```

```
myBrowser = new BrowserType();

var output = 'Browser Info:<hr>';

output += 'isFirefox:' + myBrowser.isFirefox + '<br>';
output += 'isChrome:' + myBrowser.isFirefox + '<br>';
output += 'isSafari:' + myBrowser.isFirefox + '<br>';
output += 'isOpera:' + myBrowser.isFirefox + '<br>';
output += 'isIE:' + myBrowser.isFirefox + '<br>';
output += 'isEdge:' + myBrowser.isFirefox + '<br>';
output += 'isBlink:' + myBrowser.isFirefox + '<br>';
output += 'Browser Version' + myBrowser.majorVersion + '.' +
            myBrowser.minorVersion + '<br>';
document.getElementById("divBrowserInfo").innerHTML = output;
```

The code above creates and uses a browser object that contains details of the detected Browser. Not that this browser sniffing code is likely written differently than you may choose to do it, but that just illustrates the variety of ways to accomplish the task. This should provide a good example of how to create custom objects and use them in your web applications.

Prototypes in JavaScript

We saw in the code above that we can create properties and methods for a class type by including the necessary code in the main Object Type function declaration (within the constructor).

Any property or method included in this *constructor area* will be used by any object instance that is created from this type as we proved above.

If, in code, you want to *add* a new property or method to an object on the fly, you can easily do this:

Example

```
myFirstPerson.hairColor = 'Brown';
myFirstPerson.nationality = 'Russian';
myFirstPerson.save = function() {
    // Code goes here to save data to a database.
}
```

Now, our **myFirstPerson** object has two new properties and one new method. This would work great...be encouraged to try it yourself!

The problem, though, is that these new members only apply to *this specific instance* of **PersonType**. *If* we had a second instance, a **mySecondPerson**, for example, *that* instance would not contain these members and, as a result, these two **PersonType** instances would behave differently.

So if that's what you want, then that's fine. But what if you really intended *all* instances to get these new properties and new functionality?

In order to ensure that all instances of a given type are given the new functionality, we must modify the underlying *template* - this is referred to iin JavaScript as the *prototype*.

In order to modify the prototype, we reference it as a property of the object instance as follows:

Example

```
myFirstPerson.prototype.hairColor = 'Brown';
myFirstPerson.prototype.nationality = 'Russian';
myFirstPerson.prototype.save = function() {
    // Code goes here to save data to a database.
}
```

Now, after having done this, any instances created from the **PersonType** should have the **hairColor** and **nationality** properties and the **save()** method.