# Week 4: More JavaScript Basics

## Dot Notation

JavaScript uses "dot notation." The goodies inside of an object are accessed by calling the name of the object, and then using a dot to access the property or method of that object. The syntax looks like this:

```
objectname.property
```

Or:

```
objectname.method(parameter)
```

For instance, the **Document** object has a property named "lastModified." It tracks when the page was last changed. You could grab the value of this property by writing:

```
document.lastModified
```

The **getElementById()** method of the **Document** class retrieves any element **id** passed in as a parameter to this method. If you had a page with **div** on it with the **id** of "content," you could grab it by saying:

```
document.getElementById("container");
```

Once you have hold of the **div**, you can play with and modify it however you want.

## Variables

Variables are containers that hold something of value. You give the container a name so that you can retrieve it for later use. Variables can be declared anytime, anywhere in your scripts. They can be created as global variables, which means they are available from anywhere in the script, or they can be created as local variables, which have meaning only in a very narrow set of circumstances. Variables are created using the "var" keyword.

Variables can contain lots of different kinds of data. Here are just a few:

**Numbers:** Numbers can be whole numbers or decimals, or an expression that resolves to a number such as (a+b)/c. Here are some examples:

```
var integerNumber = 5;
var decimalPointNumber = 3.14;
var formula = (5/9) * 100;
```

**Strings:** Any letter or cluster of characters, forming words or not, or an expression that resolves to words. Strings are always enclosed in quotations. Here are some examples:

```
var friendName = "Dante";
var dateToday = "August 26, 1967";
var streetAddress = "2015" + " Main Street";
```

**Booleans:** Binaries, true or false. Here are some examples:

```
var videoIsPaused = true;
var dogIsBarking = false;
```

# Functions

Functions are reusable blocks of instructions that can be called again and again by name. Like variables, they return something of value. There are lots of pre-written functions. By calling the name of the function, you trigger a block of code to execute. Object methods are functions. The **alert()** function is a good example of a pre-written function.

Functions allow the developer to break their code into small "chunks" that are targeted to accomplish a particular task. These chunks can then be applied as needed and organized in ways that make sense. Not only does this save time by making code reusable, it also allows us to structure our code in meaningful ways.

This is what a function declaration looks like:

```
function namedFunction(parameter) {
     statementBlock;
}
```

```
namedFunction(parameter);
```

The function declaration starts with a keyword, "**function**" and is followed by a name of your choosing.

The function name should be something you can remember and recognize. The best names are descriptive of what the function does.

There can be as many parameters passed as needed. Parameters are essentially variables that are local to the function; they have no meaning outside of the function. These are generalized names that act as space holders until the real values have been passed into the function call. Using parameters allows you to tailor the way your functions behave.

### A curly brace opens the statement block.

The statement block is where all the action is. This block of code determines how the function behaves and what it actually does. It can be as long and complex as needed. Programmers prefer to keep their functions short and focused on the task at hand. They'd rather write a bunch of simple, easy-to-maintain

blocks of code, rather than one long, complex chunk. The individual statements in the statement block all finish with a semi-colon.

A curly brace closes the statement block.

The function call causes the code to execute.

Here is an example that stores the date for a class meting in a variable:

```
function setClassDate(aDate){
     var classDate = aDate;
}
```

```
setClassDate("August 23, 2016");
```

# Events

An "event" is when something happens. There are lots of different kinds of events. In fact, JavaScript has an **Event** object. The **Event** object tracks all kinds of events.

The simplest kind of event to understand is user events. The user clicks on a link and this simple action triggers four distinct mouse events:

- The mouse hovers over the link.
- The user presses the mouse button down.
- The user releases the mouse button.
- The user moves their mouse off the link.

There are a lot of events that may not involve the user. For instance, the load event of the document object fires when the full page has loaded. You can listen for this event and use it to trigger a cascade of media, animations, or other wonderful user experiences.

We use "event listeners" to "listen" for an event. An event listener is attached to an object like a link or a graphic. The object is now registered to track if the event occurs.

An "event handler" is a function that is designed to respond when the event occurs. It tells your page what to do when any of those events that are being listened for occurs.

# Comments

Just like with HTML and CSS, commenting your code saves pain down the road. Do it often, and do it liberally! There are two convenient ways to comment code in JavaScript:

```
// This is a single line comment in JavaScript.
```

Or:

```
/*
 This is a longer multi-line comment.
 Many lines of content can be placed in
 this comment.
*/
```

# Adding Javascript to a First Page

It is time we add some JavaScript to an actual page. Like CSS, we can add JavaScript directly to our page or through an external file. In our demo, we will place the JavaScript in our HTML page.

We will be adding some very simple JavaScript to our page. We will create a **div** with the **id** set as **current_date** underneath our paragraph that will contain the current date. The date will be determined by the JavaScript **Date** object. This means that every time our page is loaded in the browser it will contain the current day's date.

The finished product will look like the figure below.

```
<!DOCTYPE html>
<html>
<head>
    <title>Module 1 First Page</title>
    <link rel="stylesheet" type="text/css" href="module1.css">
</head>
<body>
    <section id="container">
        <h1>About This Class</h1>
        <p class="p-font">This is where I will place info
        about this class. I could put info
        about myself, my assignments or my comments.</p>
        <div id="current_date"></div>
    </section>
    <script>
        var todays_date = new Date();
        // Get div element and add todays_date to its contents
        document.getElementById('current_date').innerHTML = todays_date;
    </script>
</body>
</html>
```

As you can see, we added the **div** underneath our paragraph and gave the **div** the **id** value of **current_date**. Then we added the **script** tags above the closing **body** tag. Inside the **script** tags we placed our logic. We first got today's date with the **Date** object, **var todays_date = new Date();**. We the script finds the **div** tag by its **id** "current_date" and then places **todays_date** inside it via **innerHTML**. **innerHTML** is a property that replaces the HTML inside a tag: **document.getElementById('current_date').innerHTML = todays_date;**

Go ahead and open the page in your favorite Web browser. If you find that it is not displaying the date, double or triple check that the code in your file looks like the figure shown here.