

Introduction

Welcome Back!

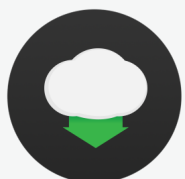
If you've ever looked at JavaScript code on your own, it probably looked as strange and meaningless as some secret code written by aliens from another planet. And you may have found that any attempt to modify that code was an exercise in absolute frustration.

If so, you'll find this lesson a relief. JavaScript code, after all, is written by people, not aliens. You *can* comprehend it. And it *is* possible to find code that almost does what you want and tweak it to fit your own needs. First, however, you need to understand the rules of the game and how JavaScript code is written. That's what we'll be doing in this lesson.



One of the first steps to making sense of it all is understanding how JavaScript treats the Web browser and document as a collection of *objects*. Those objects have properties and methods that you can use to change the page and perform other feats in response to different events that occur while the user is interacting with the page.

Terms like *object*, *properties*, and *methods* can sound a little intimidating when you're first learning about this stuff. But they're really just concepts designed to make programming easier and more like working with things in real life. After all, the room in which you're sitting right now probably contains lots of objects. Maybe a chair, a desk, a computer, or some pictures on the wall. And those objects all have properties, like size and color and such. It's the same for things in a Web page. In this lesson, you'll get some hands-on practice with objects, properties, methods, literals, and variables.



Download

Below is the link to the ZIP file that contains two pages you can take for a spin in any browser:

- **Objects Samples:** This sample demonstrates the use of object methods and properties.
- **Variables:** This page will be used to show you the difference between literals and variables.



[Download Lesson-03_ClassProject.zip](#)

750 Bytes.(Gigabytes) ZIP

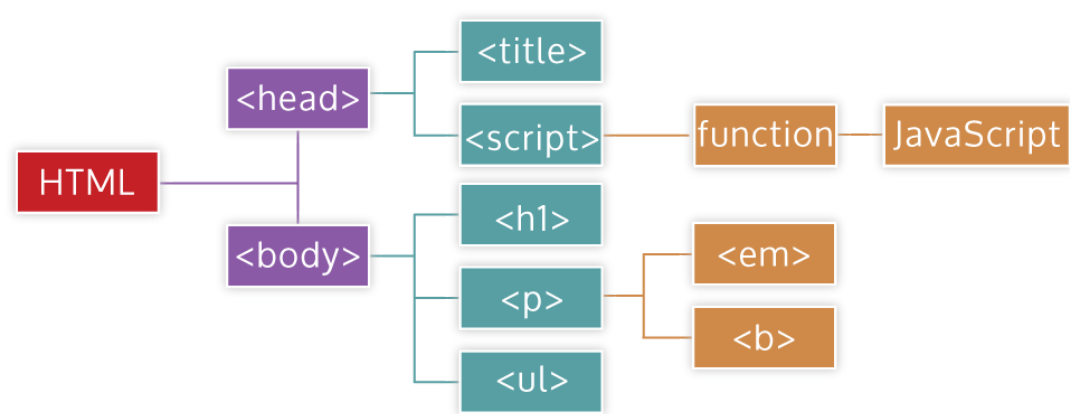
You're welcome to type these pages from scratch, if you're so inclined. Or, if you're not in the mood to practice typing right now, feel free to simply save the zip file to your *Intro JavaScript* folder, and extract the files.

We'll take a look at these pages later, for now let's mosey over to Chapter 1.

Chapter 1: The Document Object Model

The DOM

In this chapter, you're going to start learning about the *Document Object Model (DOM)*. JavaScript and the DOM go hand-in-hand because the DOM is essentially a set of rules and words you use to access and manipulate things on a Web page. Like all modern programming languages, JavaScript is *object-oriented*, which means it's designed to allow you to treat a Web page as though it were a collection of actual objects in the real world.



Let's start by picking apart the name *Document Object Model*.

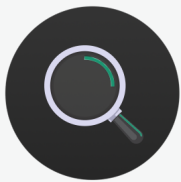
Text equivalent start.

Topic	Information
Document	The term <i>document</i> can be defined as a piece of written, printed, or electronic matter that provides information. Though, for our purposes, it's basically a webpage.
Object	An <i>object</i> can be defined as a material thing that can be seen and touched. Cars, chairs, shoes, golf balls, jets, and pretty much everything that's in the room you're sitting in right now is an object. On a webpage, an object is anything that you see or interact with.

Topic	Information
Model	The term <i>model</i> , of course, has multiple definitions. But in this case, the only definition that matters is <i>model</i> as being a three-dimensional representation of a person or thing. The key word there is <i>representation</i> . The model isn't the actual object that you can see or touch. The model is an intangible representation of a real object. In JavaScript, that representation consists of names that you type into your JavaScript code.
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

Virtually all modern programming languages use some form of *Object-Oriented Programming (OOP)*. So even though you'll be learning specifically about JavaScript in this course, the concepts you'll learn about the objected-oriented nature of the language will likely apply to any programming languages you learn in the future.

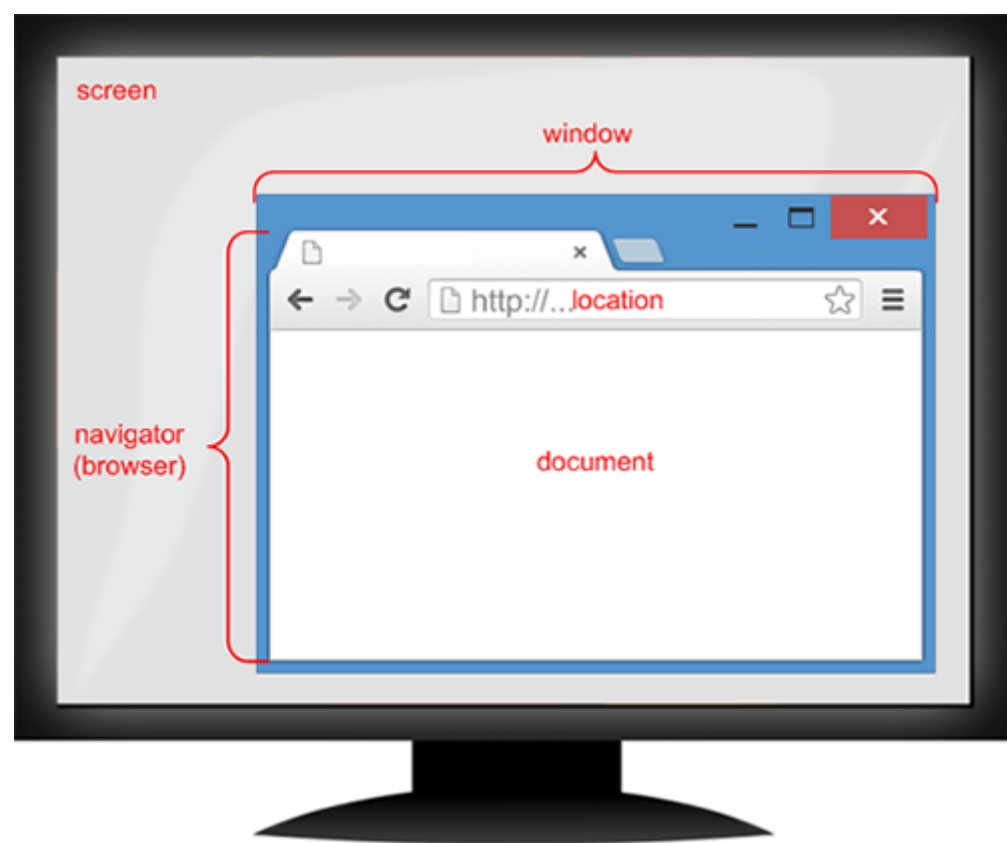


Search for Answers and Resources

Any time you need a quick definition, consider searching typing *define:* followed by the word you need defined into Google. For help with any JavaScript keyword, type *javascript* followed by a space and keyword you're seeking into Google an see what resources come up.

Objects, Properties, and Methods

Common JavaScript Objects



In JavaScript, we use different words to define objects. Below are some of the more commonly used object names:

Instructions: Click each object to learn what it refers to.

screen

window

The user's screen

Program window that contains the user agent

navigator

location

The user agent (browser) that's showing the page

Address bar of the browser

document

The webpage

Those names from the object model relate directly to things you can see right on your screen when viewing any webpage.



All of the object names are in lowercase.

That's because in JavaScript, those names must be spelled in lowercase. Since JavaScript is case-sensitive, the code won't work if you don't use the correct uppercase and lowercase letters.

Object Properties

Like real objects in the real world, JavaScript objects in the DOM have *properties*. In the real world, some common properties for objects include size, weight, color, and the materials from which it's made. In JavaScript, properties might include things like width, height, font, color, background color (all of which can be manipulated through JavaScript code).

To refer to a property of an object in JavaScript, you type the object name followed by a dot and a property name. Different objects have different properties, as you'll learn in this course. But the basic syntax is always the same:

```
object.property
```

Like object names, property names never contain blank spaces.



Camel Caps

In long property names, *camel caps* are often used to start a new word within the property name, as seen here: `window.innerWidth`. They're called camel caps because, like real-world camels, the hump is near the middle. But unlike the hump on a real camel, in text the hump is caused by uppercase letters, as with the *W* in this example. You'll see some more examples below.

- **R/O:** Some properties are read-only (R/O), which means they give you some value, but you cannot change that value. When a property is read-only, we say the property allows you to *get* the value (or the property *returns* the value).
- **R/W:** Some properties are read-write (R/W), which means you can get the value of the property and also assign a new value to the property. When a property is read-write, you can *get* or *set* the property value. Here are some examples.

Object and property	Purpose	Type
screen.height	Gets the height of the user's screen	R/O
screen.width	Gets the width of the user's screen	R/O
window.innerWidth	Gets the width of the viewport inside the browser window	R/O
window.innerHeight	Gets the height of the viewport inside the browser window	R/O
navigator.appVersion	Gets the user agent version information	R/O
navigator.platform	Gets information about the user's operating system	R/O
document.lastModified	Gets the date and time that the page was last modified	R/O
document.referrer	Gets the URL of the page that opened the current page	R/O
document.title	Gets or sets the title of the page	R/W
location.href	Gets or sets the URL in the address bar	R/W
Properties		

You'll get some hands-on practice with documents and properties shortly. First, let's look at another feature of objects in the DOM.

Methods

If properties are like descriptive nouns that describe details about an object, *methods* are more like verbs or action words. For example, in the real world, all cars have certain methods in common, such as the ability to *start*, *accelerate*, *steer*, *brake*, and *park* the car. Methods in JavaScript code are actions, too, and refer to things you can make the browser and document do.

Like properties, method names contain no spaces, and longer ones are spelled using camel caps. Unlike properties, a method name is always followed by a pair of parentheses. In some cases, you can pass a value to a method inside the parentheses, as you'll learn later in the course. Not all objects have methods. Here are some examples of commonly used methods, just so you can get sense of what the syntax looks like.

Object and method	Purpose
-------------------	---------

Object and method	Purpose
<code>window.close()</code>	Closes the current window
<code>document.write(<i>value</i>)</code>	Types text specified by <i>value</i> into the page
<code>document.getElementById(<i>value</i>)</code>	Locates the element whose ID name is <i>value</i> so that you can manipulate that element with JavaScript
<code>location.reload()</code>	Reloads the current page
Methods	



Tip

There are lots more objects, properties, and methods in the Document Object Model beyond these few examples. But don't worry about having to memorize them all. It's easy to look up information as needed, thanks to the Internet. We'll provide useful resources in the Resources for Further Learning section. Just remember to add any resources you find particularly useful to your browser's Bookmarks or Favorites so that you can easily find them in the future.

It always helps to get a little hands-on practice with these things because seeing is believing (and understanding). So, we'll try out some JavaScript code that uses DOM objects, properties, and methods.

But first, let's cement your understanding of what we've covered so far with a little matching game. Match each item on the left with the object name that refers to that item in the Document Object Model.

Text equivalent start.

Item to Match	Matching Choices	Correct Match
The webpage	<ul style="list-style-type: none">screenwindowlocationdocument	<ul style="list-style-type: none">document

Item to Match	Matching Choices	Correct Match
The program window	<ul style="list-style-type: none">documentnavigatorscreenwindow	<ul style="list-style-type: none">window
The browser	<ul style="list-style-type: none">screenlocationwindownavigator	<ul style="list-style-type: none">navigator
The address bar	<ul style="list-style-type: none">documentscreenwindowlocation	<ul style="list-style-type: none">location
The entire screen	<ul style="list-style-type: none">navigatorwindowscreendocument	<ul style="list-style-type: none">screen

Instructions: Read the item in the first column and consider which choice(s) it matches to in the second column. Read the third column to find out if you are correct.

Text equivalent stop.

Chapter 2: Hands-On DOM

In Your Editor

Open *ObjectsSamples.html* in your editor. At first glance the page might look intimidating. But in fact, it's just a regular old Web page with all the usual tags, some content, and a bit of JavaScript code here and there between the `<script>` and `</script>` tags.

```
<!DOCTYPE html>

<html>

<head>

    <title>Object Samples</title>

</head>

<body>

    Your screen width is <script> document.write(screen.width)</script> pixels.<br>
    Your screen height is <script> document.write(screen.height)</script> pixels.<br>
    Your user agent is <script> document.write(navigator.appVersion)</script>.<br>
    Your platform is <script> document.write(navigator.platform)</script>.<br>
    Your page width is <script> document.write(window.innerWidth)</script> pixels.<br>
    Your page height is <script> document.write(window.innerHeight)</script> pixels.<br>
    This page was last modified on <script> document.write(document.lastModified)</script>.<br>
    This page title is <script> document.write(document.title)</script>.<br>
    This page URL is <script> document.write(location.href)</script>.<br>

    <label for="username">Type your name:</label>

    <input type="text" id="username"> then click

    <input type="button" value="Go" onclick="alert('Hello ' +
document.getElementById('username').value)" >

    <p>

    <button onclick = "window.close()">Close Me</button>

    </p>

</body>

</html>
```

For example, let's dissect the first line under `<body>`:

```
Your screen width is <script>document.write(screen.width)</script> pixels.<br>
```

If you ignore the script tags and JavaScript code, you can see that most of the line is just regular content (text) followed by a `
` tag (a line break to end the line in the browser):

```
Your screen width is pixels.<br>
```

The script tags and JavaScript code that we ignored in the above line looks like this:

```
<script>document.write(screen.width)</script>
```

Let's break this phrase down.

Text equivalent start.

Topic	Information
<code><script></code>	The <code><script></code> tag just says to the web browser, "Hey, there's some JavaScript code coming."
<code>document.write()</code>	The <code>document.write()</code> part uses the <code>write()</code> method of the document object to write some text into the page at that exact location.
<code>screen.width</code>	What gets written to the page depends on what you put inside the parentheses of the <code>write()</code> method. In this example, we put <i>screen.width</i> between the parentheses, which uses the width property of the screen object to get (return) the screen width.
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

Most of the lines to follow in the page work the same way: a line of text with some embedded JavaScript code to get information about the system or browser or page and display it on the page. The `
` tag at the end of each line is just a line break to end each line.

In Your Browser

Open *ObjectsSamples.html* in your browser.

For the sake of example, let's assume you happen to be using a monitor with a width of 1366 pixels. That number gets substituted for `<script> document.write(screen.width)</script>` before the browser actually displays the page on the screen. So, what you see on the screen is something like this:

Your screen width is 1366 pixels.



Note

The exact number that gets returned depends on the resolution of the monitor you're using to view the page; it won't be the same for everyone. The pixel width of your screen will be there in place of 1366.

Many of the other lines of code work the same way. If you save the page and then open it in a Web browser so the code can execute, you'll see the result. It won't be the same for everyone, because the JavaScript code is displaying information specific to you:

Your Screen

- Your screen width is . . . pixels.
- Your screen height is . . . pixels.

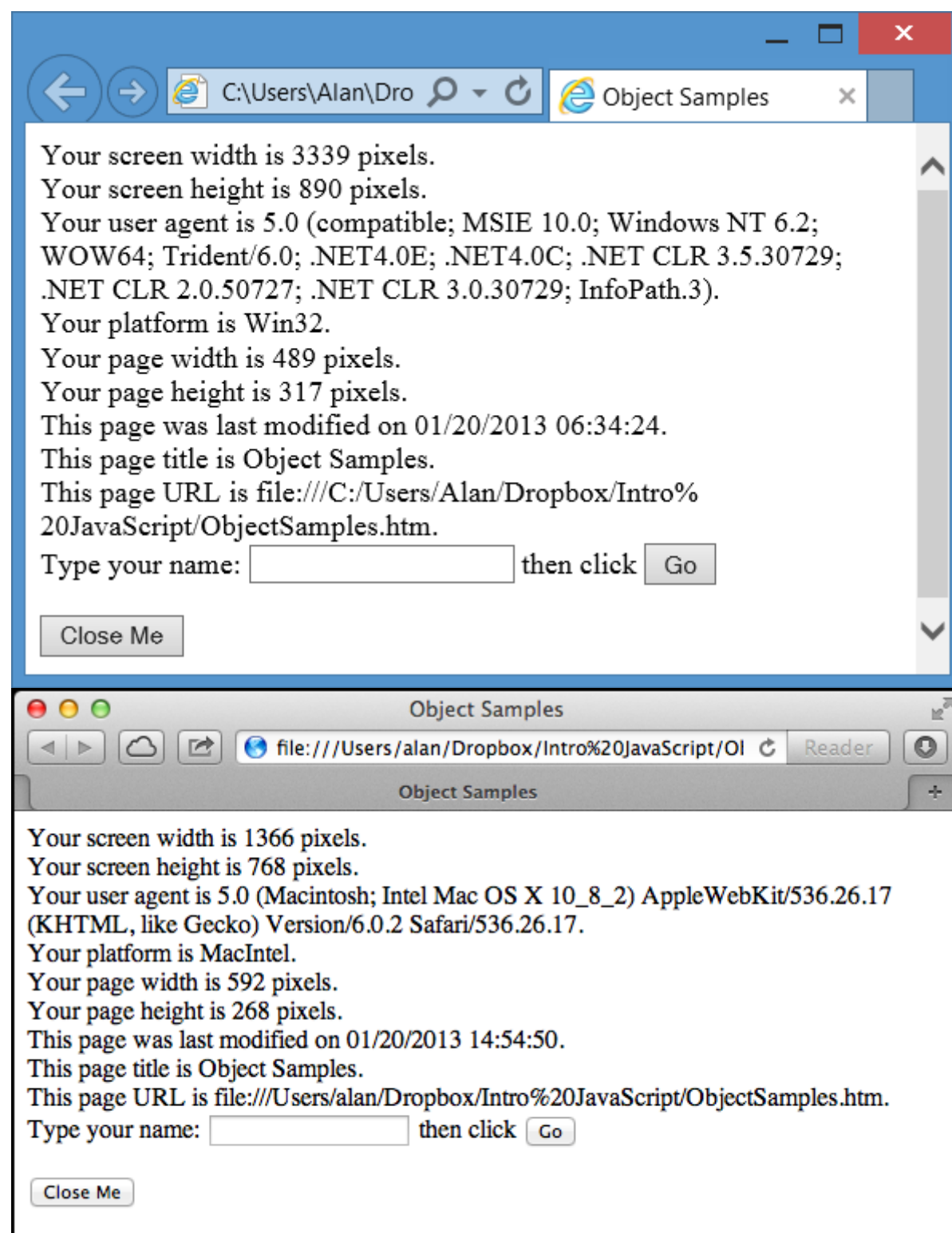
Your System

- Your user agent is . . .
- Your platform is . . .

Your Page

- Your page width is . . . pixels.
- Your page height is . . . pixels.
- This page was last modified on MM/DD/YYYY HH:MM:SS.
- This page title is Object Samples.
- This page URL is . . .

As you can imagine, these won't be the same for everyone. But the page should look (roughly) like one of these if the code is right (just don't worry about the specific information returned by JavaScript as that won't be the same for everyone).

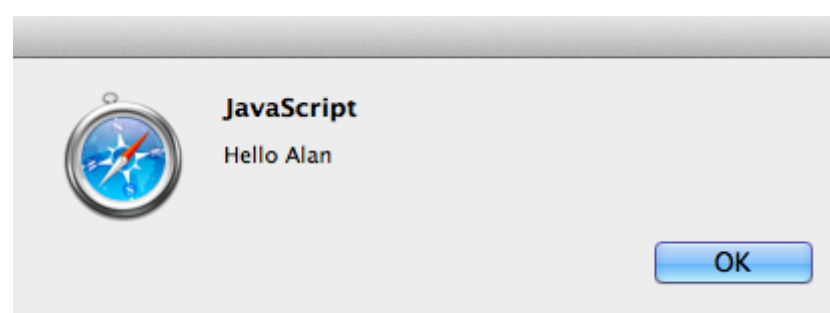


At the bottom of the page, you see the phrase *Type your name* with a textbox to the right of it. This is followed by the phrase *then click* and a **Go** button.

- When you click the label *Type your name* or the textbox, the cursor goes into the textbox. There, you can type your name (or anything else you like).

Type your name: then click

- When you click the Go button, an alert box opens showing Hello followed by whatever name you typed into the textbox.



Inputs and Alert Box

Let's talk about the code for the inputs and alert box, since it illustrates using HTML and JavaScript together. The HTML code is shown below (we'll leave out the JavaScript for now so you can focus on the HTML). If you're not familiar with HTML tags like `<label>` and `<input>`, don't worry, you'll learn about those throughout this course. As with any HTML tag, you can find information about those tags at any time online.

```
<label for="username">Type your name:</label>

<input type="text" id="username"> then click

<input type="button" Value="Go" onclick="alert('Hello ' + document.getElementById('username').value)" >
```

All of this code creates the textbox and **Go** button at the bottom of the webpage:

Type your name: then click

Control Label

```
<label for="username">Type your name:</label>
```

The `<label>` and `</label>` tags surround the text Type your name:. This tells you that the text will be visible on the webpage. While this text looks the same as any other text on the page, if you look inside the `<label>` tag, you'll see that it says `for="username"`. This part associates the label text with a control named username. It also tells the browser that, if the user clicks the label text, it should move the cursor into the control named username. You're not required to do that in your own pages. However, it is a convenience for the user because sometimes it's easier to click the label next to a control to move the cursor to the control than it is to click the actual control.



Terms

Control

The term control refers to any element on the page with which a user can interact. For example, a textbox is a control in which the user can type text. A button is a control that the user can click.

User

The user is whatever person happens to be sitting there viewing and interacting with the page.

Control Input

So, where is this control named username? It's right next to the label and defined by an input tag that looks like this:

```
<input type="text" id="username"> then click
```

The type="text" attribute makes the control a textbox into which the user can type text. The id="username" part gives the control the name username, which is used both by the <label> tag and by some JavaScript code that follows. That ID name, username, is just a name we made up. It could have been any name, so long as that name starts with a letter and contains no blank spaces.

Control Methods

The next tag displays the **Go** button and also contains an event handler and some JavaScript code:

```
<input type="button" value="Go" onclick="alert('Hello ' + document.getElementById('username').value)" >
```

There, the type="button" makes the control a button, and the value="Go" puts the word Go on the button. The onclick= attribute contains JavaScript code that's executed when the user clicks the button. So, when the user clicks the button, the JavaScript shows an alert box that contains the word Hello followed by a blank space, plus whatever text the user typed into the control named username. To find that control, the JavaScript code uses the getElementById method of the document object:

```
document.getElementById('username')
```

That part of the JavaScript code finds the specific control (textbox) that's needed. The .value on the end is a property of the textbox control that returns whatever text the user typed into the box. To see for yourself, you can click OK to close the Alert box, type something different (or nothing at all) in the textbox, and click Go again. When you do, another alert box will open to show Hello followed by whatever you typed into the textbox.

The very last tag in the body of the page looks like this:

```
<button onclick = "window.close()">Close Me</button>
```

The `<button>` tag is a tag that displays a button on the page. It's similar to `<input type="button">` but has a different syntax and offers some unique capabilities that we'll explore later. The main reason for using it here, though, is just to help you start getting used to seeing different HTML tags for controls in Web pages.

This button has an onclick event handler like the other one. But when you click this button, the JavaScript code that's executed is `window.close()`, which uses the `close()` method of the window object to close the window that's displaying the page.



Important!

This capability is controversial.

Some browsers support using this method to close the browser window, some don't. Some may even show a warning asking if you want to close the window. The reasoning is that the user should be the only one who is allowed to close the program window, not the code.

So if you test it in a browser and `window.close` doesn't work, it's only because you're using a browser that doesn't support that feature. It's not a mistake in your code, and there isn't any alternative code you can write to force the window to close. But try not to worry about it. The main thing now is to start getting used to JavaScript syntax, and notice how much of our sample code uses the DOM with its `object.property` and `object.method()` syntax.

Now, let's talk about another concept that's important in JavaScript and, in fact, all programming languages: variables.

Chapter 3: Literals and Variables

What are Literals and Variables?

In this chapter, you're going to learn about literals and variables. Like most nerd jargon, both terms can be a little scary at first. Especially that word *variable*, which might conjure up bad memories of hard math classes back from school. But like most things, they're not so scary once you remove the shroud of mystery.

Literal: A value that never changes

A literal is usually some text enclosed in quotation marks that never varies. Consider this bit of code as an example:

```
alert("Hello World")
```

When executed in a Web page, that bit of JavaScript code always shows an alert box that contains the words *Hello World*. It doesn't matter when you execute the code; it doesn't matter who is viewing the page; the alert box will always show the words *Hello World*. That's because inside the parentheses of the alert method, the words *Hello World* are enclosed in quotation marks and hence are treated as a *literal*. Another way to word it or think about it is that this code means *literally* show the words *Hello World* in an alert box.

The diagram shows the code `alert("Hello World")` with two red brackets underneath. The first bracket is under the word `alert` and has the label "In an alert box" below it. The second bracket is under the text `"Hello World"` and has the label "Show these words" below it.

Variable: A value that can vary (change)

The opposite of a literal is a *variable*. Simply stated, a variable is a placeholder for any unknown information that can vary. Variables play a huge role in many websites. Consider, for example, Google. When you go to its home page, you see a textbox waiting for you to tell it what you want to search for. It doesn't already know what you want to search for, because it can't read minds. But once you type in the word or phrase for which you want to search, it can store whatever you typed in some variable and then use that information to search its extensive database for websites that contain that word or phrase.

The syntax for defining a variable in JavaScript is simply:

```
var name
```

Naming Your Variable

The *name* is the part you get to make up yourself. But that name has to abide by a few basic rules:

- The first character must be a letter or an underscore (_).
- After the first character, the name can contain any combination of letters, numbers, underscores, or hyphens.
- The variable name cannot contain blank spaces.
- The only punctuation characters allowed are the hyphen (-) and underscore (_).

The variable name *cannot* be the same as any of the following JavaScript reserved words:



JavaScript Reserved Words

break, case, catch, class, continue, debugger, default, delete, do, else, enum, export, extends, finally, for, function, if, implements, import, in, instanceof, interface, let, new, package, private, protected, public, return, static, super, switch, this, throw, try, typeof, var, void, while, with, yield,



What is a Reserved Word?

Reserved words in a programming language are words that have special meaning in current or proposed future versions of the language.

Like everything else in JavaScript, variable names are case-sensitive. So when making up your own variable names, try to use uppercase and lowercase letters in some way that makes sense to you and is easy to remember.

Coding Literals and Variables

To practice with JavaScript, we're going to work with the *variables.html* page you saved to your *Intro JavaScript* folder earlier. Even though the code looks a lot like our first *Hello World* example, there is a subtle but important difference: the alert contains a variable, not a literal.

```
alert ("goober")  
      Literal (has double quotation marks)  
  
alert ('goober')  
      Literal (has single quotation marks)  
  
alert (goober)  
      Variable name (or something else  
                     that contains a value)
```

When text is enclosed in quotation marks is a literal (taken literally). Without the quotation marks, JavaScript assumes it's the name of a variable (or something else in JavaScript whose value can vary).

This means that unlike with the phrase *alert("Hello World")*, with the phrase *alert(goober)* you won't see an alert box with the word *goober* in it when the page first opens. If you changed the `alert()` line from *alert(goober)* to *alert("goober")* or *alert('goober')*, then when the page opened, it would show an alert box with the word *goober* in it. You can try it out and see for yourself, if you like.

However, in this chapter, we're going to focus on variables, which are the opposite of literals, and you don't put their names in quotation marks.

Defining Variables

To create a variable in JavaScript, you use the keyword *var* followed by the name of the variable.

Here are the Steps

1. Open *variables.html* in your editor.
2. Type the following simple line of code below the `<script>` tag, above the alert line, to define your variable named *goober*:

```
var goober
```

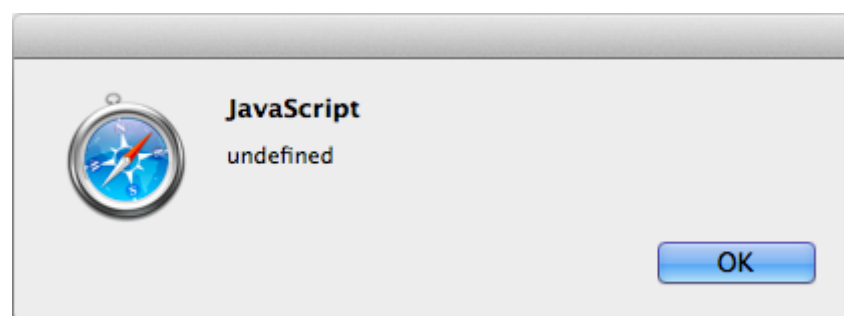


A variable has to be defined before it can be used.

It's JavaScript code, so it still has to go between the `<script> . . . </script>` tags in the page. However, you must define the variable before calling on it. You should wind up with something like this:

```
<script>
  var goober
  alert (goober)
</script>
```

3. Save the page with that change. (Make sure there are no quotation marks around *goober* anywhere in the page.)
4. Then open the page in a browser. This time you'll see an alert box that contains the word *undefined*. The exact appearance of that alert box, as always, depends on your browser.



Assigning a Value to a Variable

Now that we've established that a variable is a placeholder for information that might vary, we need to talk about how you get the information into the variable. We call that information the variable's *value*, and assigning a value to a variable is just about as simple as simple can be. You follow the variable name with an equal sign and the value you want to put in there. The syntax is:

```
name = value
```

In JavaScript, the `=` sign is sometimes referred to as the *assignment operator* because it's used to assign a value to a variable.



Initializing the Variable

You can create a variable and give it a value at the same time by using this syntax:

```
var name = value
```

This is sometimes called *initializing the variable* because we're creating the variable (with the `var` keyword) and giving it an initial value (with the `=` sign) in one fell swoop.

1. Let's give it a try! Open your *variables.html* page in your editor.
2. To assign a value to the goober variable right in the same line where we create it, simply replace the phrase *var goober* with:

```
var goober = "Howdy"
```

Or

```
var goober = 'Howdy'
```

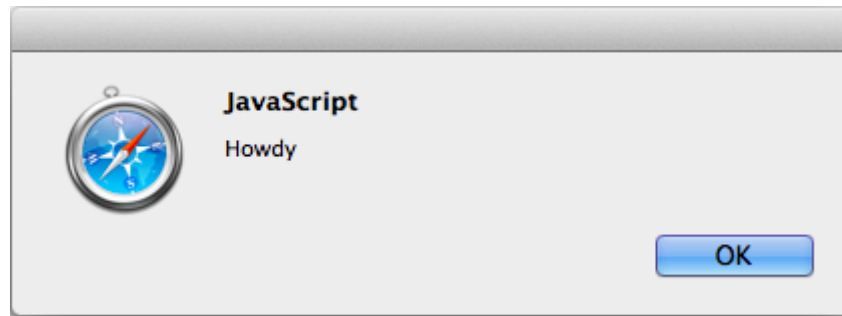


Note

The value we're assigning to the variable is some text, enclosed in quotation marks. That's important because if you tried to do it without quotation marks JavaScript would have to assume *Howdy* is the name of some other variable.

But that's not what we mean here, and we haven't created a variable named Howdy. So if you want to store the word *Howdy* in the goober variable, you'll have to let JavaScript know that you're putting literal text in there by including the quotation marks (either single or double--just make sure you use two of the same).

3. Now that the goober variable has been assigned a value of *Howdy*, its value is no longer *undefined*. Go ahead and save the page with that change.
4. Open and reload or refresh it in the browser, the alert box will show *Howdy* (the new value of the goober variable) instead of *undefined*.



The value that you assign to a variable need not be a chunk of text enclosed in quotation marks. It can be a value returned by a DOM property, a number, a date, something the user typed into a textbox, even a mathematical expression. You'll be seeing examples of those throughout the course. For now, the key thing to keep in mind is that variables are used in JavaScript (and all programming languages) as placeholders for data (information) that can vary. When you see *var* at the start of a line of code, that line of code is defining a variable that can store information that can later be used in other JavaScript code or placed on the page.

Let's review what you've learned in today's lesson.

Review

In this lesson, you learned about some of the features JavaScript shares with other programming languages, including an object model and the use of variables to store information that can vary (change over time).

- **The Document Object Model:** As you'll recall, we reviewed the DOM (Document Object Model), which defines the names used to refer to many aspects of the environment in which a Web page is showing. For example, you learned that the screen object contains information about the user's screen, while the navigator object describes the user's browser. The document object contains information about elements and controls on the page.
- **Hands-on DOM:** You also learned that different objects in the object model have different properties and methods. Properties are characteristics of the object, like size, position, color, and such. Methods are things that the object can do and that you can access through the JavaScript language.
- **Literals and Variables:** Like all programming languages, JavaScript also allows you to store information in variables. A variable is a placeholder for any information that can vary (information that isn't always the same).

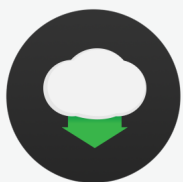
In today's assignment, you'll be getting more practice with code syntax and concepts. We'll be looking at a more complex example of the kind of code you'll encounter. By going through the exercise, you'll be a lot more familiar with how JavaScript looks and works, so be sure to set aside time to tackle it!

In these first three lessons, we've focused on fundamental aspects of JavaScript that are crucial to both writing your own code and understanding code written by others. The good news is, once you understand the basic terminology and concepts described in these first three lessons, the rest is mostly variations on those fundamentals. Much of understanding the whole big JavaScript picture really centers on understanding the things we've covered in just three short lessons.

Lesson 3 Assignment

Defining Variables with Objects, Methods, and Properties

Real-life JavaScript usually involves a combination of all the things you've learned so far in this course. So for today's assignment, let's get a little more practice with the code syntax and concepts.



Download

The *variables.htm* page that you created in the lesson is very simple. For this assignment, we're going to kick it up a notch and include some other things you've learned in this lesson.

For this assignment, you'll need the following files:

- Assignment3.html
- Assignment 3.txt



[Download L03-Assignment.zip](#)

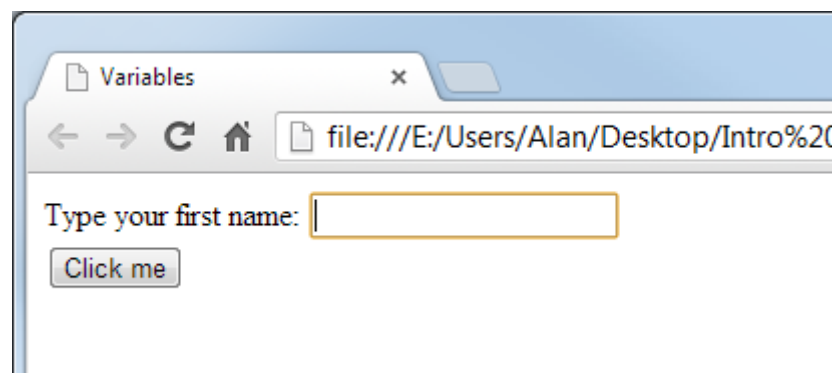
1.3 KB (Gigabytes) ZIP

You can retype or copy and paste if you'd like to run through the code, or you can simply download and save the zip file to your *Intro to JavaScript* folder and extract the files.

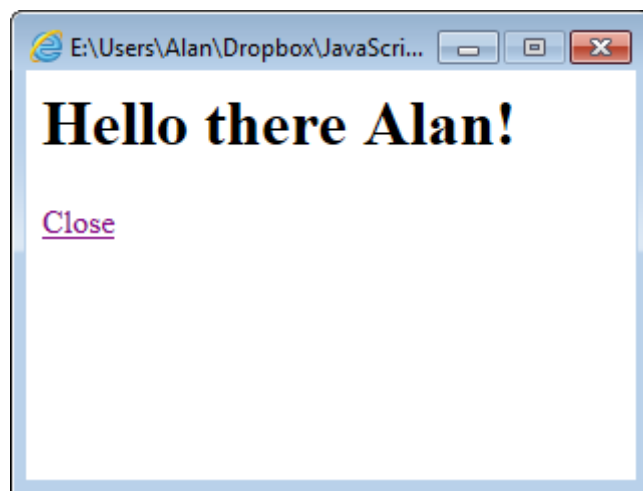
We're going to talk a bit about reading JavaScript code (so it doesn't look like meaningless gibberish to you). Knowing how to read code makes it easier to understand and modify other peoples' code and also can make it easier to debug your own code. Before we analyze the code, let's take it for a quick spin to see how it behaves.

Here are the Steps

1. Open *Assignment 3.html* in your browser. You should see the phrase *Type your first name* followed by a textbox and a button. The blinking cursor should already be in the textbox for you.



2. Type your *name* (or really any text at all) in the textbox.
3. Then, click the **Click Me** button. A new page will open with the phrase: *Hello there [your name]!* at the top of the page.



Note

Most browsers will open the new window as a separate browser window. However, some may open it as a new tab in the current window.

4. Click the **Close** link in the new page to close it.

Pretty much everything that happened there happened because of JavaScript code. This example uses several of the techniques you learned in the first three lessons. So, let's take a closer look at the code.

Function

Near the top of the page, you see a `<script>` tag followed by a function.

```
<script>

//Sample JavaScript function

function showname() {

//Store text from txtfname textbox in fname variable

var fname = document.getElementById("txtfname").value;

//Create a message from literal text and fname variable

var msg = "Hello there " + fname + "!";

//Open a new browser window (or tab)

var newWindow = window.open('', 'newWindow', 'width=300,height=200');

//In the new window, put the msg variable text in h1 tags

newWindow.document.write("<h1>" + msg + "</h1>");

//Add tags to close the new window

newWindow.document.write("<a href='' onclick='window.close()'>Close</a>")

//Make sure new window has the focus (isn't covered by something else).

newWindow.focus();

}
```

Event Handler

As you learned in Lesson 2, code in a JavaScript function doesn't actually do anything until an event handler on the page calls the function. So let's skip down to the body of the page and start from there.

```
<body>

<label for="txtfname">Type your first name: </label>

<input type="text" id="txtfname"><br>

<input type="button" value="Click me" onclick="showname()">

<!-- Use JavaScript to put cursor in the textbox -->

<script>

document.getElementById("txtfname").focus()

</script>

</body>
```

Let's break this down line by line:

Text equivalent start.	
Topic	Information
<label for="txtfname">Type your first name: </label>	Right under the <body> tag, you see a label for a control named <i>txtfname</i> that shows the words <i>Type your first name:</i> on the page.

Topic	Information
<code><input type="text" id="txtfname">
</code>	Next comes a textbox named <i>txtfname</i> .
<code><input type="button" value="Click me" onclick="showname()"></code>	After the textbox comes a button that shows the words <i>Click Me</i> . When someone clicks that button, its event handler will call the function named <i>showname()</i> near the top of the code.
<code><script> document.getElementById("txtfname").focus() </script></code>	<p>This little bit of JavaScript code near the end of the page, just above the <code></body></code> tag, is not inside a function, so it executes when the browser is just about finished loading the page. The exact code that executes is:</p> <pre>document.getElementById("txtfname").focus()</pre> <p>At first glance it might look like meaningless hieroglyphics. However, it does make sense to the web browser or user agent that's rendering the page. The <i>document.getElementById("txtfname")</i> uses the method <i>getElementById()</i> of the <i>document</i> object to locate the control named <i>txtfname</i>. In other words, this code locates the control that has <code>id="txtfname"</code> in its tag, which is a textbox in this case.</p>
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

Calling the Function

We know that when someone clicks the button, the function named *showname()* is called. Let's walk through the code as the browser would when the function is called, which is left to right, top to bottom, exactly like we read.



Note


The browser will skip the JavaScript comments, as those are notes for humans, not code to be executed. So we'll skip the comments, too.

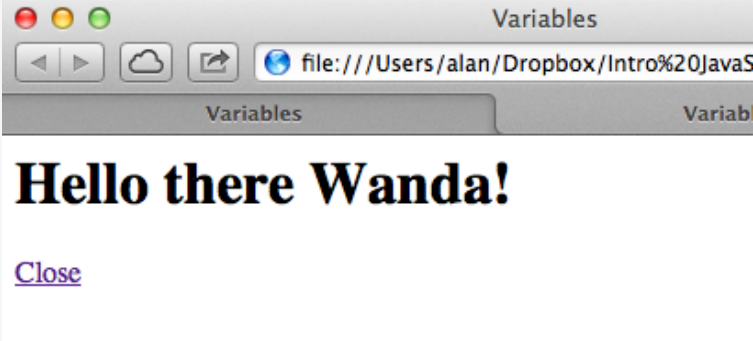
Let's start with the first line of executable JavaScript code inside the function.

Text equivalent start.

Topic	Information
-------	-------------

Topic	Information
<pre>var fname = document.getElementById("txtfname").value;</pre>	<p>This line creates a variable (placeholder) named <i>fname</i> and stores in it the value (contents) of the control named <i>txtfname</i>. In other words, whatever the user types into the textbox named <i>txtfname</i> will be stored in the variable named <i>fname</i>.</p>
<pre>var msg = "Hello there " + fname + "!";</pre>	<p>This line of code creates a second variable, which we named <i>msg</i> (short for <i>message</i>). In that variable, the code stores (literally) the words <i>Hello there</i>, plus whatever is in the <i>fname</i> variable, plus an exclamation point. So if the user typed <i>Wanda</i> in the <i>txtfname</i> textbox, the variable name <i>msg</i> will contain <i>Hello there Wanda!</i> after that line is executed.</p>
<pre>var newWindow = window.open('', 'newWindow', 'width=300,height=200');</pre>	<p>This line is a bit more advanced but still not too mind-boggling if you pick it apart. The <code>window.open</code> part uses the <code>.open()</code> method of the window object to open a new browser window. The <code>var newWindow=</code> part sets up <code>newWindow</code> as the name of the new window in JavaScript, so you can refer to the window by that name in later JavaScript code.</p> <p>Inside the parentheses:</p> <p>The first argument, <code>"</code> is two single quotation marks with nothing in between, referred to as an <i>empty string</i>. That argument can be used to specify the URL of the page that you want to open in the new window. Using an empty string, like we did, opens the window empty, so you can write your own content into it from other JavaScript code.</p> <p>The second argument inside the parentheses, 'newWindow' is the HTML target name of the window, which you could use in an HTML <code>target=</code> attribute if you wanted to. That name doesn't have to be the same as the JavaScript name. But it's often convenient to name it that way.</p> <p>The last argument inside the parentheses, 'width=300,height=200' defines the size of the window in pixels.</p>

Topic	Information
<pre>newWindow.document.write("<h1>" + msg + "</h1>");</pre>	<p>Here is where having that variable named <i>newWindow</i> comes in handy for referring to the new window. That line uses the write method of the document object to write code into the page. Specifically, it writes an h1 tag, followed by whatever text is in the msg variable, plus a closing h1 tag. So if someone typed <i>Wanda</i> into the textbox, that line of code will write this into a new page in a new window:</p> <pre><h1>Hello there Wanda!</h1></pre> <p>Hopefully, you recognize this as a simple HTML h1 heading.</p>
<pre>newWindow.document.write("Close")</pre>	<p>Once again, we use the write method of the document object to write add this bit of HTML code to the page that opens in a new window:</p> <pre>Close"</pre> <p>This creates a link that says <i>Close</i>. When clicked, it closes the window.</p> <div><h2>Your New Page as Two Lines of Code</h2><p>By the time those last two lines of code have been executed, the new window contains a new webpage. The browser, of course, renders the HTML that's in that new page just as it renders all other HTML. So that new page displays what the HTML says to display. That webpage includes the two lines of code that we had the JavaScript write into the document. Thus, the new page contains this HTML and JavaScript code:</p><pre><h1>Hello there Wanda!</h1> Close"</pre></div>

Topic	
	<p>Thus, that new page has an h1 heading that says <i>Hello there Wanda!</i> and a link that says <i>Close</i>.</p>  <p>Of course, the name <i>Wanda</i> will be whatever the user originally typed into the fname textbox. As for the Close link, when the user clicks that link, the new browser window (or tab) closes because the <code>window.close()</code> JavaScript code is executed in response to that click.</p>
<pre>newWindow.focus();</pre>	<p>This last line of executable code in our <code>shownames()</code> function applies the <code>.focus()</code> method to our New Window. When you have multiple program windows open, they can stack one atop the other. Applying the <code>.focus()</code> method to a new window ensures that the new window is on the "top of the stack" and not covered by any other open program windows.</p> <p>And if the new page opens in a new tab rather than a new window, the same concept applies. Applying the focus method to the new tab ensures that it's the selected tab and plainly visible on the screen.</p>
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

So, basically, the web browser simply does exactly what the JavaScript code told it to do. In a sense, it's kinda simple how it all works, even though the JavaScript code looks so mysterious until you get the hang of reading it. Hopefully, this assignment has helped you get a better handle on it so that you don't have to feel entirely intimidated or mystified when you see JavaScript code in the future. It's not as scary as it looks and, thanks to the Internet, you can usually look up any JavaScript keyword you encounter in code at any time.



Sneak Peek

At this point in the class, you may be wondering if `document.write()` is the only way to get content onto the page via code. No, it's not. We'll be looking at other methods in later lessons. But if you're curious about it, here's a video that shows you the whole process. Consider it a preview of things to come.

Lesson 3 Resources for Further Learning

JavaScript Terminology for Beginners (<https://sharkysoft.com/archive/1997/jsa/content/039.html>)

<https://sharkysoft.com/archive/1997/jsa/content/039.html>

Click this link for a quick review of those nerdy JavaScript buzzwords like object, property, method, variable, and function.

List of JavaScript Client Objects (<https://www.w3resource.com/javascript/object-property-method/core-object.php>)

<https://www.w3resource.com/javascript/object-property-method/core-object.php>

This is a quick resource for some of the more commonly used DOM objects

DevGuru JavaScript Quick Reference (<https://www.devguru.com/content/technologies/javascript/home.html>)

<https://www.devguru.com/content/technologies/javascript/home.html>

Here's a quick reference to JavaScript and the DOM that's organized like a book index for easy reference. It's a good candidate to add to your own browser's Bookmarks or Favorites.

JavaScript and HTML DOM Reference (<https://www.w3schools.com/jsref/default.asp>)

<https://www.w3schools.com/jsref/default.asp>

Here you'll find a good resource for looking up DOM syntax and facts as needed.

JavaScript Screen Object (<http://www.javascriptkit.com/jsref/screen.shtml>)

<http://www.javascriptkit.com/jsref/screen.shtml>

Here's a quick reference for the DOM screen object.

Window Object (<http://www.javascriptkit.com/jsref/window.shtml>)

<http://www.javascriptkit.com/jsref/window.shtml>

Clicking this link takes you to an extensive reference on the DOM window object.

window.open (<https://developer.mozilla.org/en-US/docs/DOM/window.open>)

<https://developer.mozilla.org/en-US/docs/DOM/window.open>

Here is some extensive formal documentation on the window.open method from the Mozilla Developer Network.

HTML DOM Document Object (https://www.w3schools.com/jsref/dom_obj_document.asp)

https://www.w3schools.com/jsref/dom_obj_document.asp

This page offers a brief but useful reference to the DOM document object.

JavaScript Variables (https://www.w3schools.com/js/js_variables.asp)

https://www.w3schools.com/js/js_variables.asp

This page offers a nice, quick reference to JavaScript variable syntax.

References and Cheat Sheets (<https://collections.alansimpson.me/?id=8>)

<https://collections.alansimpson.me/?id=8>

A curated collection of JavaScript cheat sheets and online references.