

JavaScript

Introduction

JavaScript is one of the easier programming languages to learn and master. However, it's used mainly in conjunction with CSS and HTML. So in addition to knowing some basic file and folder skills for your operating system, you should already be quite familiar with CSS and HTML before attempting this course. The more you know about CSS and HTML, the easier it is to understand what JavaScript is all about and how you might use it in your own work.

JavaScript Is Code

As you're probably aware, when you're creating a Web page, you're bringing two things together:

the content (the material you actually see on the Web page) and the code, which works behind the scenes to determine how and where the content appears. The code is the stuff for the computer to read and process, not for the person who is viewing the page.

The languages —HTML, CSS, and JavaScript—are kinds of code. HTML is a markup language that tells the browser what each element is. You'd use HTML code for putting text, links, and pictures on Web pages. The CSS code defines how the content looks. It's a styling language that applies the styles you choose—font, font color, formatting, and so forth—to all the pages that are ruled by that style. And JavaScript controls how the page behaves or reacts when users click something on the page. In other words, the JavaScript code determines how things behave.

JavaScript Is a Scripting Language

When you use JavaScript, you write the code into the file you use to create a Web page. Then you run it by simply opening the page in a Web browser. So you're not creating a separate, stand-alone program. Instead, JavaScript is something that runs within your Web page, alongside your HTML code and CSS code.

That's because JavaScript is something we call a scripting language (also called an interpretive language). You're only using the language to fine-tune or enhance things you've already created using some other language, such as CSS or HTML. As a result, it's pretty easy to use JavaScript, and development time tends to be short (and inexpensive). That's probably why it's so popular on the Web!

HTML Tags for JavaScript

First and foremost, since JavaScript goes in Web pages, you need to start by creating or opening a Web page. Your page right now consists of HTML tags that lay out the page. Within that HTML code, you'll be placing your JavaScript code. So next, you have to tell the Web browser where the JavaScript code starts and where it ends. In HTML5, you can use `<script>` `</script>` tags.

Obviously, the word script here stems from the fact that JavaScript is a scripting language. The `<script>` tag tells the browser that what follows is JavaScript code. And the `</script>` tag tells the browser that what follows is not JavaScript code. So you should put only JavaScript code (no content, CSS, or HTML) between those tags.

It will look like this:

```
<!DOCTYPE html> <html>
  <head>
    <title></title> </head>
  <body>
    <script>
    </script>
  </body>
</html>
```

JavaScript Comments

JavaScript allows you to include programmer comments. These are notes you write in your code, either as reminders to yourself or notes to other programmers when working in teams. You've probably already encountered comments when working with HTML.

There are two ways to write comments in JavaScript. For a single-line comment, you can start the line with two slashes,

```
// This is a one-line JavaScript comment.
```

For multiline comments, you can use `/*` to start the comment and `*/` to end the comment. (Some of you might recognize those as the same characters used to start and end comments in CSS.) Here's an example of a multiline comment in JavaScript:

```
/* This is a multiline JavaScript comment.
You don't have to mark each line as a comment.
You just have to indicate where all the comment lines end. */
```

Say "Hello World"

To dip our toes into JavaScript, we're going to use a favorite trivial example of most programmers. You'll be programming your Web page so that when the page first opens, a message comes up in a small window that reads "Hello World."

With this example, you'll get to try out the basic syntax of JavaScript methods. A method is a part of the JavaScript language that performs some operation. You'll see many methods throughout this course. JavaScript methods generally use this syntax:

```
method (parameters...)
```

Here, the method is usually a word that briefly describes what the method does, and parameters refers to one or more pieces of information that get passed to the method.

For our example, we're going to use the `alert()` method to display the message. This method uses this syntax:

```
alert("Text to Display")
```

As you can see, the method is indicated by the word `alert`. The parameters follow in the parentheses. Here, the parameter is `Text to Display`—that is, it's text that you want the alert to show. You can type in any text you want for the parameter. So the `alert` method tells the browser that you want to pop a message out in a box, and the text you put as the parameter is the content of that message.

But there's one important thing to note. JavaScript is case-sensitive, which means the keywords that define methods and other language features must be typed using specific uppercase and lowercase letters. But for now, it means that for the `alert` method to work, the word `alert` must be typed in all lowercase letters.

It's not the same for the parameter ("`Text to Display`") in this instance. The parameter is content, not code. You don't have to worry about the browser being able to "read" the content; content is for people to read. The browser only needs to read the code. So you can use uppercase, lowercase, or any combination for your display text.

So let's add this code to our page. Since we're going to have our alert box display the words `Hello World`, go ahead and type `alert("Hello World")` between your `<script>`...`</script>` tags now, as below.

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>
    <script>
      alert('Hello World');
    </script>
  </body>
</html>
```

Test the code.

Some questions that often come up the first time someone sees a JavaScript alert box include:

- Can I control the size, position, and appearance of the alert box?
- Can I control the font, color, and size of text inside the alert box?
- Can I control the icon and buttons that appear inside the alert box?

The answer to all these questions is, unfortunately, no. All you can control is the words that show inside the box. But try not to be disappointed. This is a trivial example intended only to get you starting typing JavaScript code. There's much, much more to JavaScript than little alert boxes, as you'll see in upcoming lessons.

Events

When it comes to writing JavaScript code, it's not always just a matter of controlling what the code does. Often, when the code executes is just as important. For example, sometimes you want the code to sit dormant until the user clicks or taps a button or touches the mouse pointer to some element on the

screen. Things that happen on a page while it's open are called events. The act of taking control of exactly what happens in response to some event is called event handling or handling the event.

Event handling allows you to create more-interactive pages because you get to control exactly what happens in response to every action the user takes. And JavaScript is the language you want to use to handle events.

Handling Events

So far, you were left off with a basic script that executes as soon as the Web page opens and displays an alert box. The script behaved that way because the JavaScript code is right in the body of the page (meaning it's between the `<body>` . . . `</body>` tags).

As soon as you open the page in a Web browser, the JavaScript code executes, displaying an alert box.

Most JavaScript code doesn't execute as soon as the page opens. (And most JavaScript code doesn't display an alert box either.) Most of the time, we use JavaScript code to handle events. Events are things that happen while a person is viewing and interacting with the page. For example, clicking an item on the page is an event. Touching the mouse pointer to something on the page is an event. Right-clicking an item on the page is an event. And there are many other events, as you'll learn throughout this course.

To make JavaScript code execute in response to events that happen as the user interacts with the page, we use event handlers. Many of them are like attributes that you can use right inside an HTML tag. Unlike other HTML attributes, all JavaScript event handlers start with the word `on`. For example, here are four commonly used event handlers. Note how each consists of the word `on` followed by some word (or words) that describe the event to be handled:

Event	When it happens
<code>onmouseover</code>	The user touches the mouse pointer to the item, so the mouse pointer is hovering over the item.
<code>onclick</code>	The user clicks the item (puts the mouse pointer on it and taps the primary mouse button, which is usually the left mouse button, or performs a one-finger tap on a touchscreen or touchpad)
<code>oncontextmenu</code>	The user clicks the item with the secondary mouse button (usually a right-click or CTRL + click on a single-button mouse, or a two-finger tap on a touchpad).
<code>ondblclick</code>	The user touches the mouse pointer to the item and double-clicks the primary mouse button or double-taps a touchpad.

You use event handlers inside HTML tags with the same syntax as attributes:

`eventname = "javascript code"`

where `eventname` is a valid event handler for that tag, and `javascript code` is any valid code written in the JavaScript language. HTML is not case-sensitive. So the event name (outside the quotation marks)

need not be all lowercase. For that reason, you may often see that event name typed in camel caps, such as `onClick` or `onDbClick`.

To try out a relatively simple hands-on example, follow these steps:

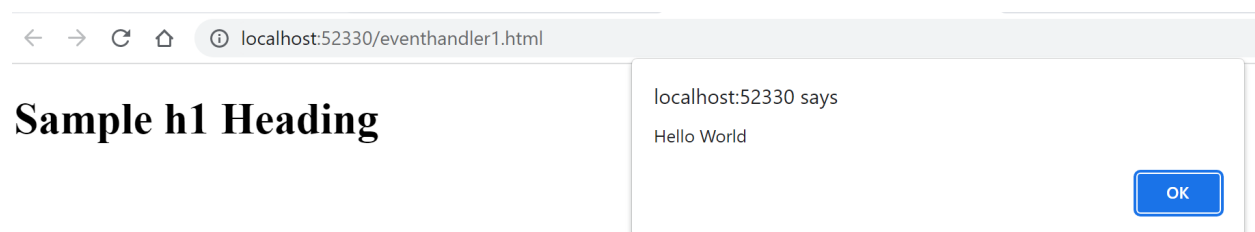
```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
  </head>
  <body>
    <h1 onclick="alert('Hello World')">Sample h1 Heading</h1>
  </body>
</html>
```

Notice that there are no `<script> . . . </script>` tags this time. That's because the `onclick=` event handler is enough to let the browser know that JavaScript code is coming. And, in fact, only JavaScript code is allowed between the quotation marks of the `onclick=" . . . "` event handler (and all other JavaScript event handlers).

Try it out now to see if you typed it correctly.

To try it out, open the html page in a browser. At first, nothing will happen. You'll just see the h1 heading on the page. That's because the JavaScript code is coupled with an `onclick` event for the h1 heading. So go ahead and click the h1 text. When you do, the JavaScript code will execute, and you'll see the alert box on the screen.

The reason that you didn't see any JavaScript code execute as soon as you opened the page is because there is no JavaScript code right in the body of the page. In this example, the only JavaScript code on the page is tied to an `onclick` event handler for the h1 element. So the JavaScript code doesn't execute until whoever is looking at the page clicks the h1 heading. When you click the heading, the JavaScript executes, and the page opens.



If your alert box opens when you click the h1 heading text, you typed the code correctly! If your code worked, click OK to close the alert box.

Other Event Handlers

In this chapter, you'll experiment with other events and coding methods to get a better feel for how you can control when JavaScript code executes. Let's try some other event handlers:

Change onclick to oncontextmenu, like this:

```
<h1 oncontextmenu="alert('Hello World')">Sample h1 Heading</h1>
```

No JavaScript code executes when the page opens. And if you click the title, nothing happens either, because the oncontextmenu event occurs only when the context menu is about to open.

The context menu (also called the shortcut menu) is the one that opens when you click the item with the secondary mouse button. That's usually the one on the right in Windows (and hence it's often called a right-click). On Mac OS you can CTRL + click the item. On some touchpads, you can put the cursor on the h1 text, and then tap the touchpad with two fingers, or click the lower-right corner on the touchpad. So it's the same code as before—an alert box displaying Hello World. By

changing onclick to oncontextmenu, you've just changed when that code executes. Click OK to close the alert box and close the browser window.

Now let's try a different code.

```
<h1 onmouseover="alert('Hello World')">Sample h1 Heading</h1>
```

This time, simply touching the mouse pointer to the heading text makes the alert box appear. That's because the onmouseover event fires as soon as the mouse pointer is hovering over the text. There's no need to click either button.

Now let's try a different code.

```
<h1 ondblclick="alert('Hello World')">Sample h1 Heading</h1>
```

That is, after you open the page in the Web browser, you have to double-click the heading to get the JavaScript code to trigger and the alert box to show.

Narrowing the Hot Spot

you may have noticed that the Sample h1 Heading text and all the space to the right of it is "hot," in the sense that clicking, right-clicking, hovering, or double-clicking that area is the same as doing so on the actual text. That may seem odd at first, but a little HTML knowledge goes a long way in explaining the mystery. In HTML, virtually all elements that start text on a new line are block elements, meaning that their default width is as wide as the browser window (or whatever element contains the block element). You can see that for yourself by putting a border around the h1 element with a little CSS code, as below.

```
<h1 ondblclick="alert('Hello World')" style="border:dashed 1px red">Sample  
h1 Heading</h1>
```

If, for whatever reason, you want only the text to be hot (not the space to its right), you can use a span tag for the event handler. A span is an inline element that's only as wide as the characters contained within it.

```
<body>
  <h1>
    <span onclick="alert('Hello World')" style="border: dashed 1px red"
      >Sample h1 Heading</span>
  >
</h1>
</body>
```

So the text is still an h1 heading because it's in `<h1> . . . </h1>` tags. But the h1 element no longer triggers the JavaScript code. The span tags inside the h1 element now hold the JavaScript code and event handler.

JavaScript Functions

Another place you can use JavaScript is between the `<head> . . . </head>` tags of a page, above the page body. JavaScript in the head section is usually stored in functions. Each function has a simple name and can contain any number of lines of code. The function can be called from anywhere in the page and, when called, executes all of the code within the function.

Functions are primarily used to organize code in a way that makes it easier to create, test, and debug the code and separate it from the HTML code for better organization. They also make it easier to reuse the same code across multiple pages in a site and multiple websites.

About Functions

A JavaScript function is one or more lines of JavaScript code that perform a specific task that can be called by name. The simplest syntax for a function is as follows:

```
function name () {
  . . .
}
```

The name is the part you get to make up. It must start with a letter but after that can contain any letters, numbers, hyphens, or underscores. You cannot use spaces or other punctuation in a function name. Function names are case-sensitive. So when naming a function, use uppercase and lowercase letters in a way that will be easy to remember in the future.

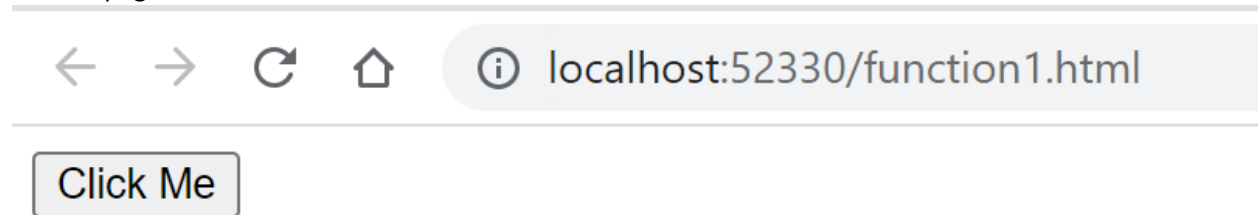
The ... in the syntax above stands for JavaScript code. The function must contain at least one line of JavaScript code. There's no upper limit to how many lines of code a function can contain.

Functions are generally placed in `<script> . . . </script>` tags in the head section of the page (that is, between the `<head> . . . </head>` tags). You can put any number of functions between a single pair of `<script> . . . </script>` tags.

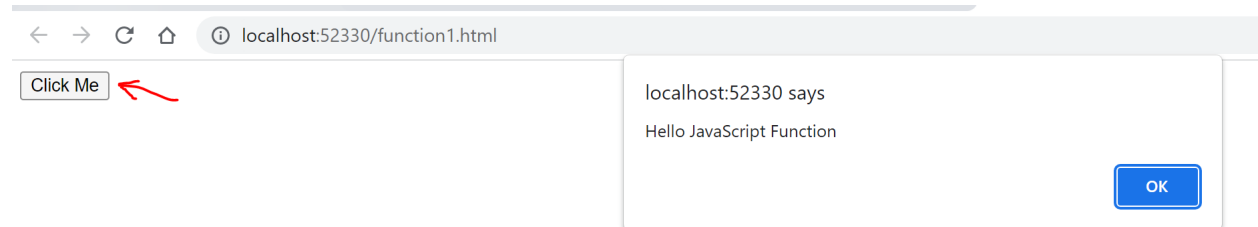
Programmers mainly use functions to organize and simplify their code. If it takes 10, 20, or more lines of code to perform some task, then all of those lines can be placed within a function. Each time you need to execute those lines of code, you just call the function by its simple name, and all of the code within the function is executed. Let's work through an example.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Function 1</title>
    <script>
      function hello() {
        alert('Hello JavaScript Function');
      }
    </script>
  </head>
  <body>
    <input type="button" value="Click Me" onclick="hello()">
  </body>
</html>
```

Save the page. Then open it in a browser again (or reload or refresh it in the browser). This time, it's not a blank page. This time it shows a button similar to the one.



The button appears because of the HTML `<input>` tag that you added to the page. We'll talk more about that `<input>` tag later in the course. For now, just be aware that inside that `<input>` tag, the `type="button"` attribute and value make it look like a button. The `value="Click Me"` attribute and value put the words Click Me on the button. (You can substitute any words you like for Click Me). The tag also contains an `onclick` event handler, and the value assigned to that is the same as the name of the function, `onclick="hello()"`.



Again, our example is somewhat trivial since our sample function contains only one line of code, and in real life, they'll probably contain many. But for now, the important thing is the concept and the syntax because they will play out over and over again.

The only thing that's important here is that the code be executed in a JavaScript function, and to call that function, we just put its name, `hello()`, in an `onclick=` event handler.

Function Parameters

A function can take a parameter, which is just values passing to the function. Parameters are specified into the parentheses. It is provided in the argument field of the function.

```
function name (param1, param2, param3) {  
    // block of code  
}
```

Example

Try it yourself:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="utf-8" />  
    <title>Functions in Parameters</title>  
    <script type="text/javascript">  
      function sayHello(name, age) {  
        // parameters  
        alert(name + ' is ' + age + ' years old.');      }  
    </script>  
  </head>  
  <body>  
    <h1>Passing the Parameters in function</h1>  
  </body>  
  <script>  
    sayHello('Steve', 23); // calling a function  
    sayHello('Alice', 21);  
    sayHello('Sam', 21);  
  </script>  
</html>
```

Default Parameters

The default parameter is nothing but to set a default value for the function parameter. If the value is not passed, it is undefined.

```
function name (param1 = value1, param2 = value2, param3 = value3)  
{  
    //Statements;  
}
```

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Functions in Parameters</title>
    <script>
      function sayHello(name, age = 20) {
        // default parameters
        document.write(name + ' is ' + age + ' years old.');      }
    </script>
  </head>
  <body>
    <h1>Default Parameters in function</h1>
  </body>
  <script>
    sayHello('Steve'); // calling a function
  </script>
</html>
```

The Return Statement

The return statement is used to return a value from the function. You need to make a calculation and receive the result.**Example:**

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Functions in Parameters</title>
    <script>
      function myfunction(a, b) {
        return a * b; //return statement
      }
    </script>
  </head>
  <body>
    <h1>Default Parameters in function</h1>
  </body>
  <script>
    document.write('Product of Two Numbers ' + myfunction(10, 30));
  </script>
</html>
```

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.

The term document can be defined as a piece of written, printed, or electronic matter that provides information. Though, for our purposes, it's basically a Web page.

And object can be defined as a material thing that can be seen and touched.

And model is a representation of a person or thing. In JavaScript, that representation consists of names that you type into your JavaScript code.

In JavaScript, we use different words to define objects. Below are some of the more commonly used object names. You'll notice that 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. The names from the object model relate directly to things you can see right on your screen when viewing any Web page.



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

Some properties are read-only (R/O), which means they give you some value, but you cannot change that value. 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-only, we say the property allows you to get the value (or the property returns the value). 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

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
<code>window.close()</code>	Closes the current window
<code>document.write(value)</code>	Types text specified by <i>value</i> into the page
<code>document.getElementById(value)</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

Hands-On DOM

Below is a page you can take for a spin in any browser.

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> . . . </script>` tags. The `<script>` tag just says to the Web browser, "Hey, there's some JavaScript code coming." The `document.write()` part uses the `write()` method of the document object to write some text into the page at that exact location. What gets written to the page depends on what you put inside the parentheses of the `write()` method. In that example, we put `screen.width` between the parentheses, which uses the `width` property of the `screen` object to get (return) the screen width.

```
<!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);
```

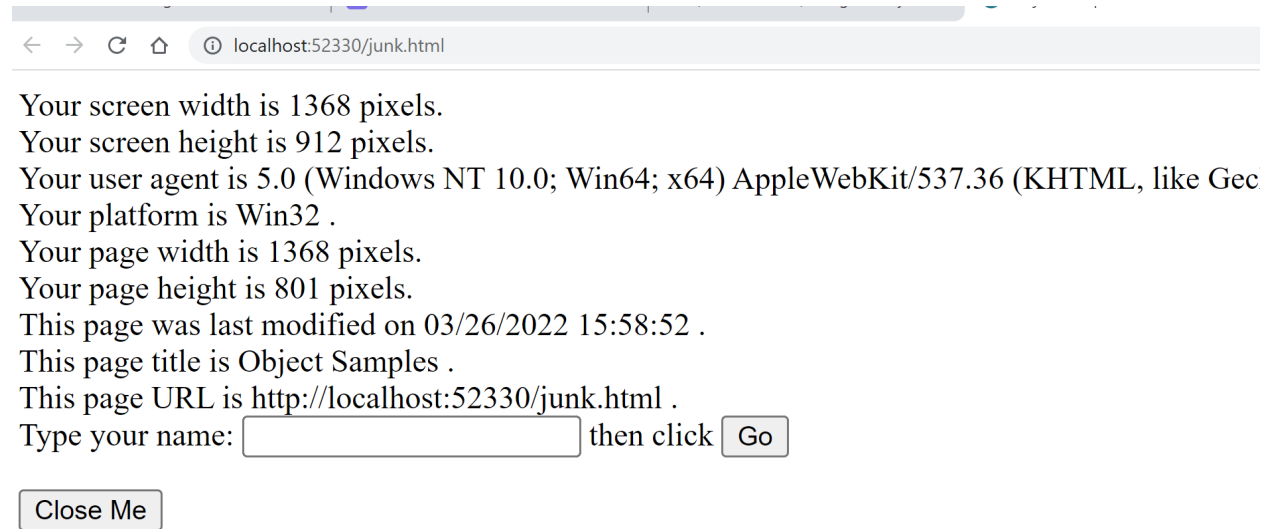
```

</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>

```

And here is the output

vfffff



A screenshot of a web browser window. The address bar shows 'localhost:52330/junk.html'. The page content includes several lines of text: 'Your screen width is 1368 pixels.', 'Your screen height is 912 pixels.', 'Your user agent is 5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gec', 'Your platform is Win32 .', 'Your page width is 1368 pixels.', 'Your page height is 801 pixels.', 'This page was last modified on 03/26/2022 15:58:52 .', 'This page title is Object Samples .', 'This page URL is http://localhost:52330/junk.html .', and 'Type your name: ' followed by a text input field, 'then click ' followed by a 'Go' button. At the bottom left, there is a 'Close Me' button.

Literals and Variables

Let me start with two very simple definitions:

- Literal: A value that never changes
- Variable: A value that can vary (change)

The syntax for defining a variable in JavaScript is simply:

var name

- The variable name must start with a letter or an underscore character (_).
- After the first character, the name can contain any letters, numbers, an underscore, or a hyphen.
- 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:

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	

Variable names are case-sensitive.

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

So if you execute

```
var tempVar
alert(tempVar)
```

You get:



Assigning a Value to a Variable

You follow the variable name with an equal sign and the value you want to put in there. So the syntax is:

name = value

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

var name = value

That's 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. It might also be good to know

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

Now try:

```
Var tempVar = "Howdy"  
alert(tempVarr)
```

Exercise

Exercise

Create an HTML file as below. Name it `variable.html`. 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.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Variables</title>
    <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();
      }
    </script>
  </head>
  <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>
</html>
```

Open `variables.html` in your browser. You should see a textbox and a button. The blinking cursor should already be in the textbox for you. Follow the commented instructions and complete the exercise.

Now, check the codes and try to understand the code and try to make some changes to see the effects. Read the notes to understand the code better.

Now,

- 1- change the window to 250 by 150 px.
- 2- change the color of the message to "red"
- 3- close the window by just moving your mouse over "close"

Dates, Number, Decision Making

The ability to make decisions is one of the key factors that separates programming languages. But before we get too deeply into these decisions, let's learn about the types of data you'll be working with when you're writing this kind of code.

Understanding Data Types

Strings (Text)

In programming language, a chunk of text is often called a string. That's short for a string of characters that have no specific numeric value. A string can be any length and can contain any characters that you can type at the keyboard. When assigning a string value to a variable, enclose the text in single or double quotation marks. Here's an example in which the code is assigning strings to variables named name, address, csz, phone, and email:

```
var name="Wanda O'Connor"  
var address="123 Oak Tree Lane"  
var csz = "Someplace, OH 43085"  
var phone="555-555-1234"  
var email="wanda555@gmail.com"
```

You can actually store anything as a string. The only time you don't want to use the string is when you have data on which you want to perform arithmetic (or date arithmetic).

Numbers

Numbers in JavaScript (and all programming languages) are quantities (also called scalar values) on which you can perform arithmetic calculations like addition, subtraction, multiplication, and division. Dollar amounts and other quantities are the most common uses. To qualify as a valid number in JavaScript, the data must meet the following requirements:

- The number can contain digits (0-9) and one decimal point (period). Commas, spaces, dollar signs, and other characters are not allowed.
- The first character in front of the number can be a hyphen to indicate a negative number. Hyphens are not allowed anywhere else.
- Never use quotation marks with numbers.

Arithmetic Operators

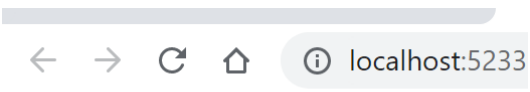
To perform basic arithmetic on numbers, use the following arithmetic operators:

Operator	Meaning
*	Multiply
/	Divide
+	Add
-	Subtract (or negative number)

Check the following code, and try to see how it works:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample Numbers</title>
  </head>
  <body>
    <script>
      //Store a number
      var qty = 5;
      //Store a number
      var price = 4.99;
      //Multiply two numbers and store result in variable
      var subtotal = qty * price;
      //Show multiplication result
      document.write(subtotal);
      //Type in a line break tag
      document.write('<br>');
      //Show multiplication result with 2 decimal places
      document.write(subtotal.toFixed(2));
    </script>
  </body>
</html>
```

When you view the page in a browser, you see the JavaScript natural math result on one line, and the same number rounded to two decimal places right below it.



24.9500000000000003
24.95

Now let's look at another, more specific use of numbers.

Dates

JavaScript provides the ability to store and work with dates. For example, it can do things like calculate the date x days ago, or x days in the future (where x is some number). It can calculate the number of days between two dates. But in order for a date to be recognized as a date, the data must be stored as a date object.

The syntax for storing a date is a little different from that for storing strings and numbers. To store the current date and time, use this simple syntax:

```
var name = new Date()
```

Here, name is a variable name of your own choosing. To store a specific date in a variable, you can use either of two different syntaxes. One is to provide the year, month, and day (in that order) as three separate values separated by commas:

```
var name = new Date(year,month,day)
```

Again, name is any variable name you get to make up yourself. Replace year with a four-digit year number. Replace month with a month number in the range of 0 to 11 (where 0 is January, 11 is December). Replace day with a number in the range of 1 to 31.

There's another syntax you can use to store a date, and it bypasses the 0 for January problem. That syntax is:

```
var name = new Date(datestring)
```

Here, datestring is a month name, followed by a space, followed by the day number, followed by a space, followed by the four-digit year. The date string should be enclosed in quotation marks. Here's an example:

```
var halloween = new Date("October 31 2013")
```

If you want to specify a time along with the date, follow the year with a space and a time expressed using military time (24-hour clock) in the format hh:mm:ss. Stay within the same pair of quotation marks. For example, this would be February 14, 2015, at 6:35 PM:

```
var valentines = new Date("February 14 2015 18:35:00")
```

Here's a sample page that illustrates each example and puts the results on the page:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample Dates</title>
  </head>
  <body>
    <script>
      //Store current date time
      var today = new Date();
      //Store dates and times using different syntaxes
      var birthday = new Date(1960, 0, 20);
      var halloween = new Date('October 31 2013');
      var valentines = new Date('February 14 2015 18:35:00');
      //Display date time values on the page on separate lines.
      document.write(today);
      document.write('<br>');
      document.write(birthday);
      document.write('<br>');
      document.write(halloween);
      document.write('<br>');
      document.write(valentines);
    </script>
  </body>
</html>
```

The output is illustrated below:

← → ↻ 🏠 ⓘ localhost:52331/dates.html

Sun Mar 27 2022 15:01:31 GMT-0700 (Pacific Daylight Time)
Wed Jan 20 1960 00:00:00 GMT-0800 (Pacific Standard Time)
Thu Oct 31 2013 00:00:00 GMT-0700 (Pacific Daylight Time)
Sat Feb 14 2015 18:35:00 GMT-0800 (Pacific Standard Time)

What's important is that each value is stored as a date—a moment in time—in a variable.

Boolean Values

JavaScript, like all programming languages, also supports Boolean values. This is the simplest data type because it can only have a value of false or a value of true. There's no maybe or any other acceptable value. If you assign a value other than true or false, the value you assign will be converted to false or

true. For example, if you assign any of the following values to a Boolean variable, then the value will be set to false:

```
0
-0
null ""
undefined NaN
```

Assigning any other value will set the value to true. Note that assigning any text in quotation marks to a Boolean value will set the value to true. So be careful not to use quotation marks in assigning a value to a Boolean value. Ideally, to set the value to false, you just want to use this value:

```
var name = false
```

Or use this syntax to set it to true:

```
var name = true
```

As always, you can replace name with a variable name of your own choosing.

Despite their simplicity, Boolean variables can be useful because there are many situations in programming in which the code has to make a decision about what to do next based on some value being true or false. As I mentioned at the start of this lesson, on any given day when you're using your computer, tablet, cellphone, or other device, the programs behind the scenes are making things happen. And they do that by making thousands of if . . . then . . . else decisions based on Boolean true and false values.

And now that you know about Boolean variables and our other types of data, you're ready to learn how you write the kind of code that makes these decisions. Join me in the next chapter to learn how.

Decision-Making

Decision-making in JavaScript and all other programming languages boils down to being able to write, in code, simple decision-making steps such as If this is the case, do this; otherwise, do that. We all make countless decisions like that in our daily lives, so this is nothing new. The trick is just learning how to express the same idea using a language that the computer can execute.

JavaScript if Statements

here are a few different ways to post if . . . then questions. Perhaps the simplest and most common is this syntax:

If (condition) code to execute;

The condition will be the name of a variable that contains a true or false value or an expression that results in a true or false result. We'll talk about what all of that means and provide many examples throughout this course. But for now, let's just focus on the syntax of the if statement. The code to

execute is one line of JavaScript code that's executed if (and only if) the condition is true. If the condition proves false, that line of code is ignored.

A couple of important things that matter a lot are:

- The word if must be typed in lowercase (typing one or both letters in uppercase is a common mistake).
- The condition must be enclosed in parentheses (forgetting one or both of them is another common error).

Let's put together a simple page to start testing and practicing if statements. Open your editor, and type or copy and paste the following code. Save the page as iftest.html in your Intro JavaScript folder.

```
<!DOCTYPE html>
<html>
  <head>
    <title>if tests</title>
  </head>
  <body>
    <script>
      //Create a Boolean variable and give it a value.
      var condition = true;
      //Use the variable as a test condition.
      if (condition) document.write('Condition is true');
    </script>
  </body>
</html>
```

After you save the page, open it in a Web browser. The page should show The condition is true because, in fact, the variable named condition contains true (assuming you typed all of the code correctly). So the line of JavaScript code that reads document.write("Condition is true") is executed, putting the words Condition is true onto the page.

Now open iftest.htm in your editor. Change the line that assigns true to the condition variable so that it assigns false to that variable instead. Do not change the if() statement or the document.write text.

```
<script>
  //Create a Boolean variable and give it a value.
  var condition = false;
  //Use the variable as a test condition.
  if (condition) document.write('Condition is true');
</script>
```

Save the page after making the change. Then open the page in a browser, or click

the Reload or Refresh icon in the browser to reload the page. If you typed all of the code correctly, the page will be blank. That's because the line of code that writes the words Condition is true on the page is executed only when the condition variable contains true. Since that variable now contains false, that line of code is ignored and, therefore, nothing is written into the page.

Using if Statements With Multiple Code Lines

The syntax you learned for if statements is very simple, and there are more ways to do it. If you want to execute two or more lines of JavaScript code when a condition provides true, you can just enclose those lines in a pair of curly braces, like this:

```
if (condition) {  
    code to execute; code to execute;  
    ...  
}
```

The syntax is basically the same as before. The curly braces just create an entire code block that is executed if the condition proves true or ignored if the condition proves false. The ... is just shorthand for "however many lines of code you want to type," and in real life you would never actually type ... into your code.

Adding an *else* Option

With this syntax, you can also create a block of code that is executed if (and only if) the condition proves false. The trick is to add the word *else* and a pair of curly braces to enclose the lines of code that are executed when the condition proves false, like this:

```
if (condition) {  
    code to execute;  
    code to execute;  
    ...  
} else {  
    code to execute;  
    code to execute;  
    ...  
}
```

How you break up the lines doesn't really matter. All that matters is the lowercase *if*, the condition in parentheses, the curly braces for code to execute if the condition proves true, and (optionally), the word *else* followed by the code to execute, enclosed in curly braces, if the condition proves false. So this is also a valid syntax:

```
if (condition) { code to execute; code to execute; ....}  
else  
{ code to execute; code to execute; ...}
```

Feel free to break lines as you see fit. But do keep in mind that most errors in typing if statements stem from forgetting parentheses or curly braces. So it's always a good idea to type up the code with all the parentheses and curly braces in place first. Then go in and fill in the actual code as a second step.

Let's try another hands-on exercise, using an if with an else in it. Open in your editor the iftest.htm page you created near the start of this lesson. Add curly braces around the `document.write("Condition is true");` statement. Then add the word `else` and `{document.write("Condition is false"); }`. How you break up the lines isn't really important, so long as you don't break a line in the middle of a word. But most people would probably type the code like this, which makes it easy to see and check on all of the required parentheses and curly braces.

```
<script>
  //Create a Boolean variable and give it a value.
  var condition = false;
  //Use the variable as a test condition.
  if (condition) {
    document.write("Condition is true");
  } else {
    document.write("Condition is false");
  }
</script>
```

Try the above code and see the results. Change the **condition to true** and check the results again.

Multiple ifs

You can nest if statements to perform different actions for different conditions using this syntax:

```
if (condition1)
{
  code that's executed if condition1 is true;
}
else if (condition2)
{
  code that's executed if condition2 is true;
}
else
{
  code that's executed if none of above are true;
}
```

You can have any number of conditions above the final `else`. And within any pair of curly braces, you can have any number of lines of code to be executed. Of course, that means there have to be multiple conditions to test. And that brings us to the condition part of the if statement:

```
if (condition)
```

Testing for Conditions

The JavaScript if statement, as in any programming language, is all about having the code make a decision about what to do next based on some situation or condition. In fact, the generic term for if . . . else-type decision-making in all programming languages is conditionals. The condition is usually a test to see if some condition (or set of conditions) is true or false and often involves comparisons.

To perform tests for conditions, we use expressions with comparison operators. For example, `score > 59` means the score is greater than 59. In this example, `score` is presumed to be a variable that contains

some number. If that number is greater than 59, then the expression returns true (the condition is true). If the score is less than or equal to 59, then the expression returns false, and the condition is false.

The greater than symbol (>) is one of several comparison operators that you can use to write expressions.

In this chapter, you'll learn about other ways to write expressions that test for true or false conditions. Let's start by examining some of the comparison operators you'll be using.

Comparison Operators

Comparison operators provide the means of comparing things so JavaScript code can make decisions. The most commonly used comparison operators in JavaScript are summarized below. I'll use x as the name of some hypothetical variable to illustrate the syntax. In real life, you'd replace x with any variable name or value.

Operator	Meaning	Example
==	is equal to	x==10
!=	not equal to	x!=10
>	is greater than	x>10
>=	is greater than or equal to	x>=10
<	is less than	x<10
<=	is less than or equal to	x<=10

You might wonder what's up with the == for is equal to. One equal sign is the assignment operator that's used to store a value in a variable. For example, take a look at this line of code:

```
var x = 10
```

Notice that we're not comparing anything there. We're creating a variable named x and assigning a value of 10 to it. In other words, we're storing the number 10 in a variable named x. Now, assuming we actually did that in some code, here are the values that the sample expressions in the table would return.

Sample Expression	Returns	Why?
<code>x==10</code>	true	Because x does equal 10
<code>x!=10</code>	false	Because x <i>does</i> equal 10, so x <i>doesn't</i> equal 10 is false
<code>x>10</code>	false	Because x <i>equals</i> 10, it's <i>not larger than</i> 10
<code>x>=10</code>	true	Because x equals 10, and that says <i>is greater than or equal to</i> 10
<code>x<10</code>	false	Because x equals 10, it's not less than 10
<code>x<=10</code>	true	Because x equals to 10, and that says <i>is less than or equal to</i> 10

Now let's take a look at that in a sample bit of JavaScript code. Try the following code and see what you get.

```
var age = 17
if (age >= 18) {
  document.write("You are an adult")
} else {
  document.write("You are a minor")
}
```

Logical Operators

JavaScript logical operators allow you to test for multiple conditions and see if all or some of them are true or not true. In the examples below, x and y are hypothetical variables used only to illustrate the syntax of creating expressions using these operators

Operator	Meaning	Example	Meaning
<code>&&</code>	and	<code>x==10 && y < 50</code>	x equals 10 <i>and</i> y is less than 50 (both are true)
<code> </code>	or	<code>x==10 y < 50</code>	x equals 10 <i>or</i> y is less than 50 (either or both are true)
<code>!</code>	not	<code>!(x==10 && y<50)</code>	Not true that x equals 10 and y is less than 50

Here's a hypothetical example of a more complex if . . . then . . . else scenario where the code is choosing to display some text based on the contents of a variable named age.

Try the following code:

```
<script>
  //Create a number variable and give it a value.
  var age = 65
  //Use the variable as a test condition.
  if (age <= 12) {
    document.write("Child fare")
  }
  else if (age > 12 && age < 60) {
    document.write("Adult fare")
  } else {
    document.write("Senior discount")
  }
</script>
```

Let's step through the code so you can see that, despite the fact that it looks like total nonsense to the untrained eye, it's actually pretty simple. Let's start with this:

```
var age =65
```

That creates a variable named age and stores the value 65 in that variable. Next, comes this bit of code:

```
if (age <= 12) {
  document.write("Child fare")
}
</script>
```

That says if age is less than or equal to 12, write "Child fare" onto the page. Since age is 65, well over 12, that doesn't happen, and next comes this part of the code:

```
else if (age > 12 && age < 60) {
  document.write("Adult fare")
}
```

That checks to see if age is greater than 12 and also if age is less than 60. However, age is not less than 60, so that's not true either, so that bit of JavaScript is skipped over. Which leaves this bit of code:

```
else {
  document.write("Senior discount")
}
```

Since neither of the other conditions proved true, that leaves only this option, and so the words Senior discount are written on to the page.

Exercise

open up your editor and type or copy and paste the following code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Interactive If</title>
    <script>
      //Show the fare when called.
      function showfare() {
        //Get the fare from the txtAge textbox.
        var age = document.getElementById('txtAge').value;
        //Create an empty string variable named msg (short for message).
        var msg = '';
        // If they didn't enter a number (is Not a Number), ask for a number.
        if (isNaN(age)) {
          msg = 'Please enter a number';
        }
        //If have is less than or equal to 12, child fare.
        else if (age <= 12) {
          msg = 'Child fare';
        }
        //Age 13 to 59 is adult fare.
        else if (age > 12 && age < 60) {
          msg = 'Adult fare';
          //Must be 60 or more if none of above are true.
        } else {
          msg = 'Senior discount';
        }
        //Put the msg text inside the paragraph named fare, using the innerHTML
property.
        document.getElementById('fare').innerHTML = msg;
      }
    </script>
  </head>
  <body>
    <!-- Ask for age -->
    Type your age:
    <input type="text" id="txtAge" style="width: 25px" autofocus /> then click
    Go
    <button onclick="showfare()">Go</button>
    <!-- Empty paragraph named fare will get content from JavaScript -->
    <p id="fare"></p>
  </body>
</html>
```

Test the program and enter different numbers.

In the code, you can see there's a function named `showfare()`. And as you may recall, JavaScript functions don't actually do anything until they're called. So let's ignore that for now and look down in the body of the page where you see this:

```
<input type="text" id="txtAge" style="width:25px" autofocus>
```

That's a textbox control into which the user can type text (or numbers). I named it `txtAge` using `id="txtAge"`. I used CSS to give it a width of 25 pixels. That part is entirely optional. However, the default width is much wider than that, and since we're only expecting the user to type a relative small number, it makes sense to show a textbox that's just wide enough to accommodate two or three characters.

The `autofocus` is an HTML5 attribute that automatically puts the cursor in the textbox as soon as the page opens.

The next key ingredient is this button control.

```
<button onclick="showfare()">Go</button>
```

That shows the Go button on the page which, when clicked, will call the `showfare()` function. Before we look at that, though, there's one more important element on the page, created by these tags.

```
<p id="fare"></p>
```

That is a paragraph that contains no text. So at first, it doesn't show anything at all on the page. However, the paragraph has a name, as defined by `id="fare"`. Which is good news, because JavaScript can find any element on the page that has an ID. So we can use that as a place for JavaScript to write text, as you'll see shortly.

Now let's go back up to the head section and look at the JavaScript function.

```
<script>
  //Show the fare when called.
  function showfare(){
    //Get the fare from the txtAge textbox.
    var age = document.getElementById("txtAge").value
```

There you can see the start of the function named **showfare**, which is called by the **Go** button on the page. The first thing it does is create a variable named `age`. And it places into the variable whatever the user has typed into the **txtAge** textbox on the page.

After that line is executed, we create another variable named `msg` (which is short for "message," though, of course, it could be any name). That one starts off as a empty string ("") because we're not

exactly sure what we want to put in it yet. The code has to decide what to put into that variable based on what the user typed into the **txtAge** textbox (which is now also in the age variable).

```
var msg = ""
```

Our first if condition is:

```
isNaN(age)
```

That strange-looking bit of code uses the built-in isNaN function to determine if the content of the age variable is Not a Number. The syntax required parentheses and the exact uppercase and lowercase letters shown. And the if condition also requires its own parentheses. So that first one is tricky to type. It won't work unless all four parentheses and the uppercase and lowercase letters are exactly right.

```
if (isNaN(age)) {  
    msg="Please enter a number"  
}
```

So, basically, that part says, "if the content of the age variable is not a number, put Please enter a number in the msg variable."

Next, the code says if the age variable contains a number that's less than or equal to 12, then put the words Child fare in the msg variable.

```
else if (age <= 12) {  
    msg="Child fare"  
}
```

And the next statement says if the age is greater than 12 and the age is also less than 60, then put the words Adult fare in the msg variable.

```
else if (age > 12 && age < 60) {  
    msg="Adult fare"
```

The next one says else (meaning nothing else has been true yet), then put the words Senior discount in the msg variable. We know, at that point, that age must be a number that's 60 or more because we've exhausted all the other possibilities with previous ifs.

```
    //Must be 60 or more if none of above are true.  
} else {  
    msg="Senior discount"  
}
```

So by the time code execution reaches here, the msg variable contains some text that's suited to whatever the user typed into the age variable. The only thing left to do is show that text on the page, and that's handled by this last line of code in the function:

```
document.getElementById("fare").innerHTML=msg
```


The `document.getElementById("fare")` part finds the element that has the ID name of fare, which in this case is a paragraph on the page. Then we use the JavaScript `innerHTML` property to put some content inside the paragraph. Even though the method `read innerHTML`, you can use it to write text or code into the element. In this case, we're just writing some text into that paragraph. And the text we're writing is whatever happens to be in the `msg` variable at that point in time. So the paragraph shows the appropriate message text based on what the user typed into the text box.

There is one more catch that here that we need to deal with in order to fully round out this sample page. If you leave the textbox blank and click Go, you get "Child fare" rather than "Please enter a number". This is because a blank textbox returns an empty string ("") which is considered equal to the number zero. And oddly enough, it's considered a valid number too (which makes no sense to most of us, but that's how it works).

Try to fix this issue (not looking at a solution below)

To deal with this bit of JavaScript weirdness, we need to add some code to the page. Currently, we use the following code to show the "Please enter a number" message if a person types in a non- number:

```
if (isNaN(age)) {  
    msg="Please enter a number"  
}
```

That `if()` needs to include some logic that checks to see if it's not a number or a zero length string. In JavaScript we use `||` for or and we use `==` for a comparison equals. To represent a zero-length string, we use `""`, a pair of quotation marks with nothing between them, not even a space. So to really put the finishing touch on the sample code, change the `if()` logic shown above to this:

```
if (isNaN(age) || age=="") {  
    msg = "Please enter a number"  
}
```

So, basically, you've written some code that makes a decision about what to show on the page, based on what the user types into that page. Most highly interactive websites and applications work on the same principles, where content that's most appropriate to show on the page at any moment is deduced from known data and `if . . . else` decisions.

Loops

Loops let you run a block of code a certain number of times.

The for loop

A for loop is made up of four statements and has the following structure:

```
for ([initialisation]; [conditional]; [iteration])  
[loopBody]
```

The initialization statement is executed only once, before the loop starts. It gives you an opportunity to prepare or declare any variables.

The conditional statement is executed before each iteration, and its return value decides whether or not the loop is to continue. If the conditional statement evaluates to a false value then the loop stops.

The iteration statement is executed at the end of each iteration and gives you an opportunity to change the state of important variables. Typically, this will involve incrementing or decrementing a counter and thus bringing the loop ever closer to its end.

The loopBody statement is what runs on every iteration. It can contain anything you want. You'll typically have multiple statements that need to be executed and so will wrap them in a block ({ ... }).

Here's a typical for loop:

A typical for loop

```
for (var i = 0 , limit = 100; i < limit ; i++) {  
    // This block will be executed 100 times  
    console.log ('Currently at ' + i );  
    // Note : the last log will be "Currently at 99 "  
}
```

The while loop

A while loop is similar to an if statement, except that its body will keep executing until the condition evaluates to false.

```
while ([ conditional ]) [ loopBody ]
```

Here's a typical while loop:

A typical while loop

```
var i = 0;  
while (i < 100) {  
    // This block will be executed 100 times  
    console . log (' Currently at ' + i );  
    i ++; // increment i  
}
```

You'll notice that we're having to increment the counter within the loop's body. It is possible to combine the conditional and incrementer, like so:

A while loop with a combined conditional and incrementer

```
var i = -1;  
while (++ i < 100) {
```

```
// This block will be executed 100 times
console . log ( ' Currently at ' + i );
}
```

Notice that we're starting at -1 and using the prefix incrementer (++i).

The do-while loop

This is almost exactly the same as the while loop, except for the fact that the loop's body is executed at least once before the condition is tested.

```
do [ loopBody ] while ([ conditional ])
```

Here's a do-while loop:

A do-while loop

```
do {
    // Even though the condition evaluates to false
    // this loop 's body will still execute once .

    alert ( ' Hi there ! ' );
} while (false);
```

These types of loops are quite rare since only few situations require a loop that blindly executes at least once. Regardless, it's good to be aware of it.

Breaking and continuing

Usually, a loop's termination will result from the conditional statement not evaluating to true, but it is possible to stop a loop in its tracks from within the loop's body with the break statement.

Stopping a loop

```
For ( vari=0 ; i<10 ; i++ )
{ if(something)
    { break;
  }
}
```

You may also want to continue the loop without executing more of the loop's body. This is done using the continue statement.

Skipping to the next iteration of a loop

```
For ( vari=0 ; i<10 ; i++ ) {

    If ( something ) {
        continue;
    }
}
```

```
// The following statement will only be executed  
// if the conditional 'something' has not been met  
console.log ( 'I have been reached' );  
}
```

Arrays

Arrays are zero-indexed lists of values. They are a handy way to store a set of related items of the same type (such as strings), though in reality, an array can include multiple types of items, including other arrays.

A simple array

```
var myArray = [ 'hello' , 'world' ];
```

Accessing array items by index

```
var myArray = [ 'hello' , 'world' , 'foo' , 'bar' ];  
console.log(myArray[3]); // logs 'bar'
```

Testing the size of an array

```
var myArray = [ 'hello' , 'world' ];  
console.log ( myArray.length ); // logs 2
```

Changing the value of an array item

```
var myArray = [ 'hello' , 'world' ];  
myArray [1] = 'changed';
```

Adding elements to an array

```
var myArray = [ 'hello' , 'world' ];  
myArray.push ( 'new' );
```

Working with arrays

```
var myArray = [ 'h' , 'e' , 'l' , 'l' , 'o' ];  
var myString = myArray.join ( '' ); // 'hello'  
var mySplit = myString.split ( '' ); // [ 'h' , 'e' , 'l' , 'l' , 'o' ]
```

Objects

Objects contain one or more key-value pairs. The key portion can be any string. The value portion can be any type of value: a number, a string, an array, a function, or even another object.

[Definition: When one of these values is a function, it's called a method of the object.] Otherwise, they are called properties.

As it turns out, nearly everything in JavaScript is an object — arrays, functions, numbers, even strings — and they all have properties and methods.

Creating an “object literal”

```
var myObject = {  
  sayHello : function() {  
    console.log('hello');  
  },  
  
  myName : 'Rebecca'  
};  
  
myObject.sayHello();           // logs 'hello'  
console.log(myObject.myName); // logs 'Rebecca'
```

Note When creating object literals, you should note that the key portion of each key-value pair can be written as any valid JavaScript identifier, a string (wrapped in quotes) or a number:

```
var myObject = {  
  validIdentifier: 123,  
  'some string': 456,  
  99999: 789  
};
```