Introduction

Welcome Back!

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.

You may want the code to sit dormant until the user clicks or taps a button, at which point an information box appears. Or perhaps when the user touches the mouse pointer to some element on the screen, it launches a game. Each of these actions are called *events*.

Anything that happens on a page while it's open is an event. 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. JavaScript is the language you want to use to handle events.

In this lesson, you'll learn how to control *when* your JavaScript runs. Come on over to Chapter 2, and we'll get right into it.

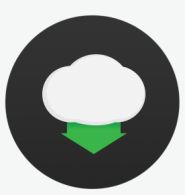
Inline JavaScript and JavaScript Functions

In Lesson 1, you created a basic script that executes as soon as the webpage opens. In this case, when you open the page your script displays an alert box. We're going to change this, because most JavaScript code doesn't execute as soon as the page opens.

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. Each of the following is an example of an event:

- Clicking an item on the page.
- Touching something on the page with the mouse pointer.
- Right-clicking an item on the page.

As you'll learn throughout this course, there are many other events you can use to launch your JavaScript.



Download

Below is the link to the ZIP file that contains the two pages you'll need to work through the exercises in this lesson:

- **Inline JavaScript:** This file is a copy of the final product you should have arrived at with *HelloWorld.htm*. This is the file that you'll use to add event handlers to.

- **JavaScript Functions:** This is a blank HTML page. You'll use this file to create your first JavaScript Function.

All you have to do is click the link below to download the zip file, save it to your *Intro JavaScript* folder and extract the two files.

Download Lesson-02_ClassProject.zip

1.8 KB (Gigabytes) ZIP

When you're ready, let's get started!

Chapter 1: Handling Events

Understanding Event Handlers

Common HTML Events

To make JavaScript code execute in response to events that happen as the user interacts with the page, we use *event handlers*. Here are some commonly used event handlers:

Event	When it happens
onload	The browser finishes loading the page.
onchange	The user changes an item on the page.
onmouseover	The user touches the mouse pointer to the item, so the mouse pointer is hovering over the item.
onclick	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)
oncontextmenu	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).
ondblclick	The user touches the mouse pointer to the item and double-clicks the primary mouse button or double-taps a touchpad.

All JavaScript event handlers start with the word *on*.

You may have noticed that each event handler consists of the word *on* followed by some word (or phrase) that describes the event to be handled. All event handlers have two things in common:

- They all start with the letters *on*.
- They never contain a blank space.

Here's a link to a handy cheat sheet that contains of all the JavaScript event handlers:

Cheatography.com - JavaScript Cheat Sheet (https://cheatography.com/davechild/cheat-sheets/javascript/)

Coding an Event Handler

Using Inline JavaScript

Event handlers are coded in relation to specific HTML elements on the page. Most event handlers are like attributes, in that you can use them right inside an HTML tag. You use event handlers inside HTML tags with the same syntax as attributes:

```
<HTML element tag eventname = "javascript code">HTML element content</HTML element tag>
```

The *eventname* needs to be a valid event handler for that tag, and the *javascript code* is any valid code written in the JavaScript language.

HTML is not case-sensitive.

This means that the HTML element tag and the event name need not be all lowercase. For that reason, you may see an event name typed in camel caps, such as onClick or onDblClick.

Let's say that you want the *Hello World* alert box, from your trivial example, to appear when a user clicks the <h1> heading. For the sake of simplicity, we'll also make *Hello World!* the <h1> heading for our page. Here's what your inline JavaScript would look like:

```
<h1 onclick="alert('Hello World')">Hello World!</h1>
```

Let's break this down so you can see how it's put together:

- The HTML element code is indicated by:

```
<h1>Hello World!</h1>
```

- The event handler is indicated by:

```
onclick="alert('Hello World')"
```

You may have noticed that once you add the onclick event, the double quotation marks (") surround all the JavaScript code (alert('Hello World')) and that the method parameters are now surrounded by single quotation marks (').

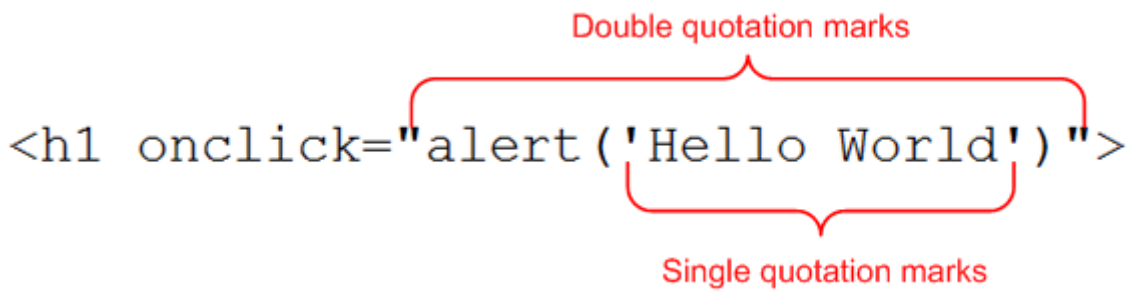
```
<h1 onclick="alert('Hello World')">
```

This is because the quotation marks that surround the method parameters as nested within the quotation marks that surround the JavaScript code.

Nesting Quotation Marks

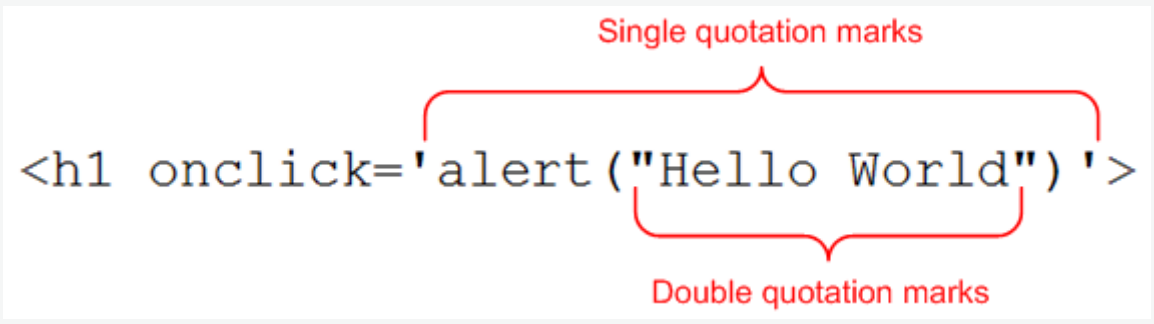
Proper Nesting

Proper nesting is important in JavaScript. Like most things, it might be easier to get it right if you understand why. When the web browser "sees" *onclick = "*, that first double quotation mark indicates where the JavaScript code starts. It will assume, from that point on, that any other code it encounters is JavaScript until it hits a closing double quotation mark. Any quotation marks inside there should be single, so as not to be confused with the quotation mark that ends all of the JavaScript code.



It's the Nesting that Matters.

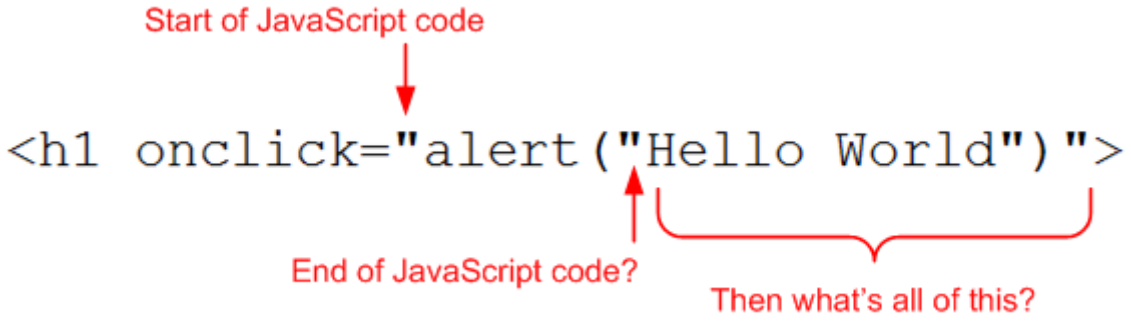
Some people prefer to put the single quotation marks on the outside and the double quotation marks on the inside. That's perfectly okay, too, because the outermost quotation marks match (they're both single quotation marks), and there are no single quotation marks between them.



The nesting is what matters, and it has to do with how the browser that's executing "sees" the code. When it sees *onclick='*, it assumes that JavaScript code continues on until it hits another single quote. Any double quotation marks inside there are assumed to be part of the code, but not the end of the code. So the code works properly.

Improper Nesting

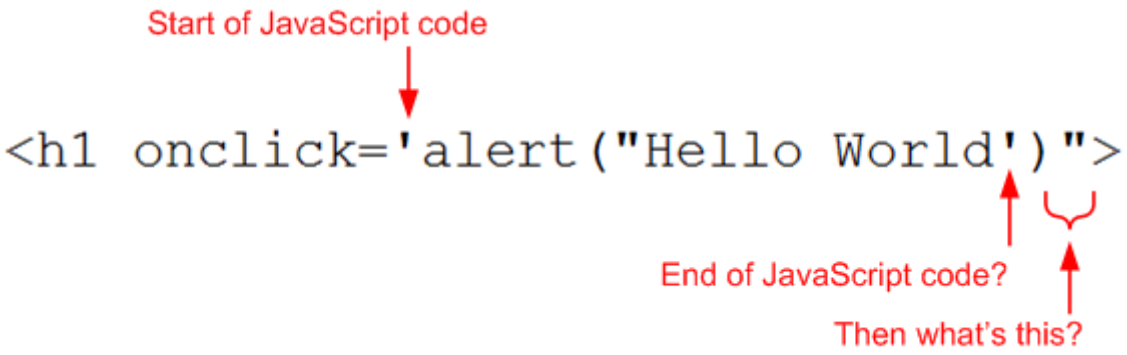
If you don't nest things properly, it will "look like" the JavaScript code has ended before it actually has ended. In the example below there are double quotation marks inside double quotation marks. This doesn't work because as soon as the browser sees the second double quotation mark, it assumes that's the end of the code and doesn't know what to do with the rest of the code after that.



Remember

It's perfectly okay to enclose all the code in single quotation marks, too. But again, you have to make sure the second single quotation mark is really the end of the JavaScript code, and any quotation marks inside are double quotation marks.

If you get things out of whack and don't nest the pairs properly, the syntax will be wrong, and the browser will think it has reached the end of the JavaScript code before it actually has.



Tip

To avoid these issues, any time you style an event handler, type both outer quotation marks right away, like this:

```
onclick=" "
```

Then put your cursor between the quotation marks and type your JavaScript code. There are two advantages to getting into the habit of typing your code this way:

- For one, you won't forget to type the closing quotation mark later (another common error), because you've already typed it.
- For another, if you need a pair of quotation marks inside there, you know you'll have to use whatever quotation you *didn't* use for the outer quotation marks. (In other words, you'd use single quotation marks in the example above because you already used double quotation marks as the outermost pair.)

Now that you understand how to code an event handler, let's go ahead and give it a try.

Chapter 2: Working with Event Handlers

onclick

With the Hello World alert box trivial example, as soon as you open the page in a web browser, the JavaScript code executes, displaying the alert box. This is because right now the JavaScript code is right in the body of the page (meaning it's between the <body> and </body> tags).

We're going to make it so that it displays when you click on the h1 heading "Hello World!"

Here are the Steps

1. Open the *InlineJavaScript.htm* file you downloaded earlier in your editor. Your code should look something like this:

```
<!DOCTYPE html>
<html>
<head>
<title></title>
</head>
<body>
<!-- The script tag marks start of JavaScript code -->
<script>
//Inside the script tags you type JavaScript code and JavaScript comments
alert("Hello World")
</script>
<!-- The /script tag marks start of JavaScript code -->
</body>
</html>
```

2. First, remove the JavaScript alert code:

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

3. Now, right under the <body> tag, type or copy and paste in the following HTML comment:

```
<!-- Example of inline JavaScript below -->
```

4. Below this comment, type or copy and paste the following inline JavaScript:

```
<h1 onclick="alert('Hello World')">Hello World!</h1>
```

5. Save the changes you've made to the *InlineJavaScript.htm* in your *Intro JavaScript* folder. You're page code, when complete, should look something like this:

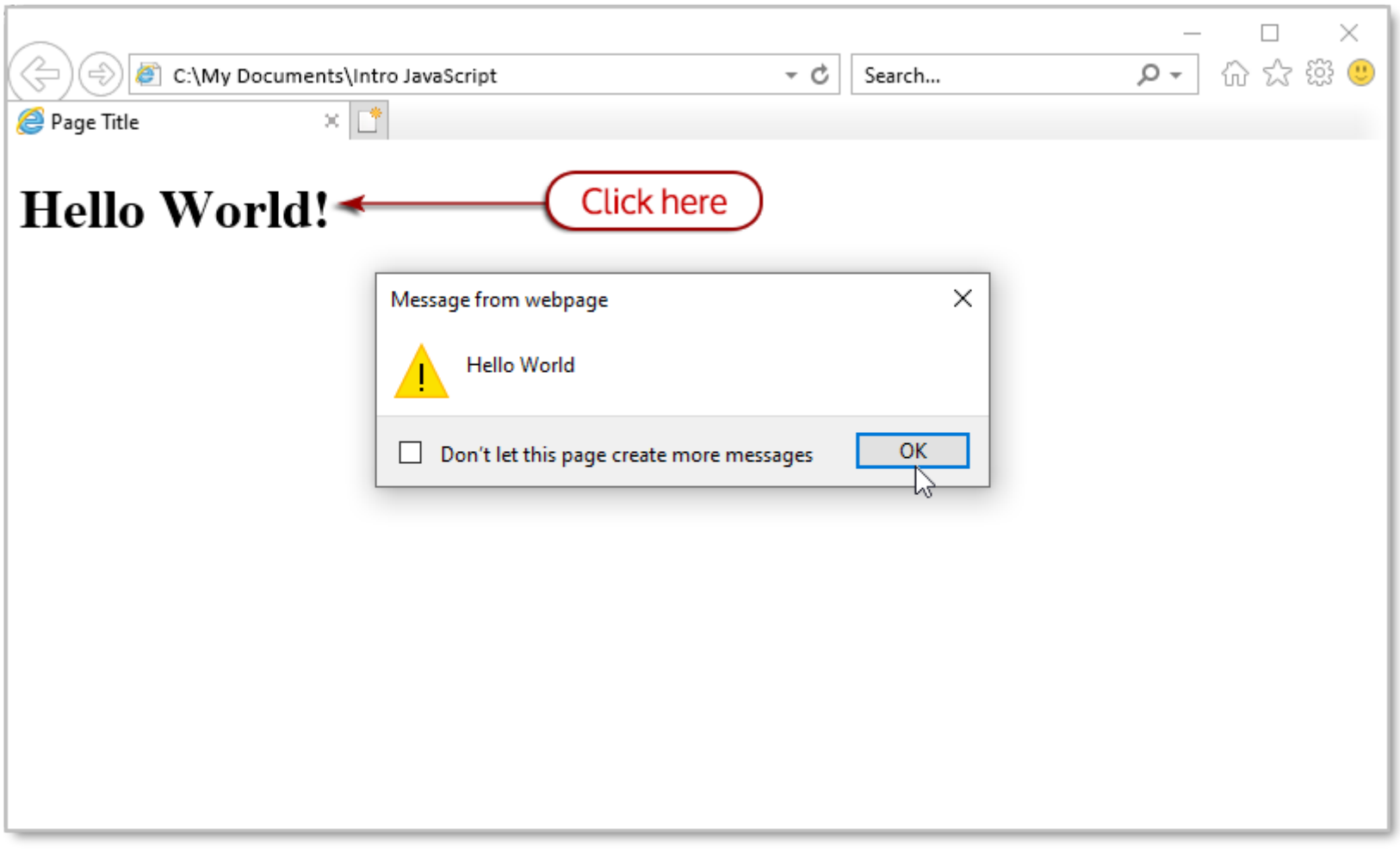
```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<!-- Example of inline JavaScript below -->
<h1 onclick="alert('Hello World')">Hello World!</h1>
<!-- The script tag marks start of JavaScript code -->
<script>
//Inside the script tags you type JavaScript code and JavaScript comments


</script>
<!-- The /script tag marks end of JavaScript code -->


</body>
</html>
```

Note

> You may have noticed that we didn't place the inline JavaScript between the <script> and </script> tags this time. That's because the onclick= event handler is enough to let the browser know that JavaScript code is coming. In fact, only JavaScript code is allowed between the quotation marks of the onclick=" . . . " event handler (and all other JavaScript event handlers).

6. Let's try it out now to see if you typed it correctly. Go ahead and open the *inlineJavaScript.htm* page in a browser. At first, nothing will happen. You'll just see the h1 heading *Hello World!* on the page. That's because the JavaScript code is coupled with an onclick event for the h1 heading.

7. To see your code in action, click the *Hello World!* h1 heading. When you do, the JavaScript code will execute, and you'll see the alert box on the screen.



- **If you click the Hello World! h1 heading text, and your alert box opens,** you typed the code correctly! If your code worked, you can click **OK** to close the alert box.

- **If you click the Hello World! h1 heading text, and nothing happens,** there might be a typo somewhere in your code. One of the most common typos with JavaScript code has to do with how you nested your quotation marks—so check there first.

As you can see, not all code has to be placed in between the <script> and </script> tags, and not all JavaScript code executes as soon as the page opens. You can use event handlers in HTML tags to determine when the code executes.

The onclick event handler is just one of many you'll be learning about in this course. Let's experiment with some other codes.

What Do Other Event Handlers Do?

oncontextmenu

The *context menu* (also called the *shortcut menu*) is the one that opens when you click the item with the secondary mouse button.

- **In Windows,** that's usually the one on the right—hence it's often called a right-click.
- **On Mac OS,** you can press 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 tap the lower-right corner on the touchpad.

Let's try this event handlers to get a better feel for how you can control when JavaScript code executes.

Here are the Steps

1. Open *inlineJavaScript.htm* in your editor.
2. Change the event handler from *onclick* to *oncontextmenu*, like this:

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

3. Save the change by pressing **CTRL + S** or **COMMAND + S**.
4. Open the page in a browser. No JavaScript code executes when the page opens.
5. Go ahead and click the *Hello World!* h1 heading. Nothing happens now either.
6. Now, right-click on the *Hello World!* h1 heading (or press CTRL + click on the Hello World! h1 heading). If you've coded correctly, the alert box should now appear.

Changing When

As you can see, the oncontextmenu event occurs only when the context menu is about to open. The JavaScript code is the same as before—an alert box displaying *Hello World*. By changing *onclick* to *oncontextmenu*, you've just changed *when* that code executes.

7. Click **OK** to close the alert box and close the browser window.

Now let's try a different code.

onmouseover

The *onmouseover* event fires as soon as the mouse pointer is hovering over the text. There's no need to click either button.

Here are the Steps

1. Open *inlineJavaScript.htm* in your editor again.

2. This time change the event handler from *oncontextmenu* to *onmouseover*, as below.

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

3. Save the page again.

4. Open the page in your browser.

5. Move the mouse pointer so it's touching the *Hello World!* h1 heading text. This time, simply touching the mouse pointer to the heading text makes the alert box appear.

6. Click **OK** to close the alert box now, and close the browser.

ondblclick

This is a variation on the *onclick* event handler. The *dbl* in between on and click is an abbreviation for double. That means, after you open the page in the browser, you have to double-click the heading to get the JavaScript code to trigger and the alert box to show. This one isn't as widely used as the others, since double-clicking is rare in webpages. But it's nice to know you do have that level of control when you need it.

Here are the Steps

1. Open *inlineJavaScript.htm* in your editor.

2. Change *onmouseover* to *ondblclick*, as below.

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

3. Save the page from your editor.

4. Now double-click on the *Hello World!* h1 heading text.

5. Once the alert box appears, click **OK** to close it.

Narrowing the Hot Spot

What is the Hot Spot?

The *Hello World!* h1 heading text is "hot" and, you may have noticed, so is all the space to the right of it. This means that clicking, right-clicking, hovering, or double-clicking the area to the right of the h1 heading is the same as doing so on the actual text.

HTML Elements are Block Elements

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).

Let's add a little code to our page, so you can see the hot spot for yourself.

Here are the Steps

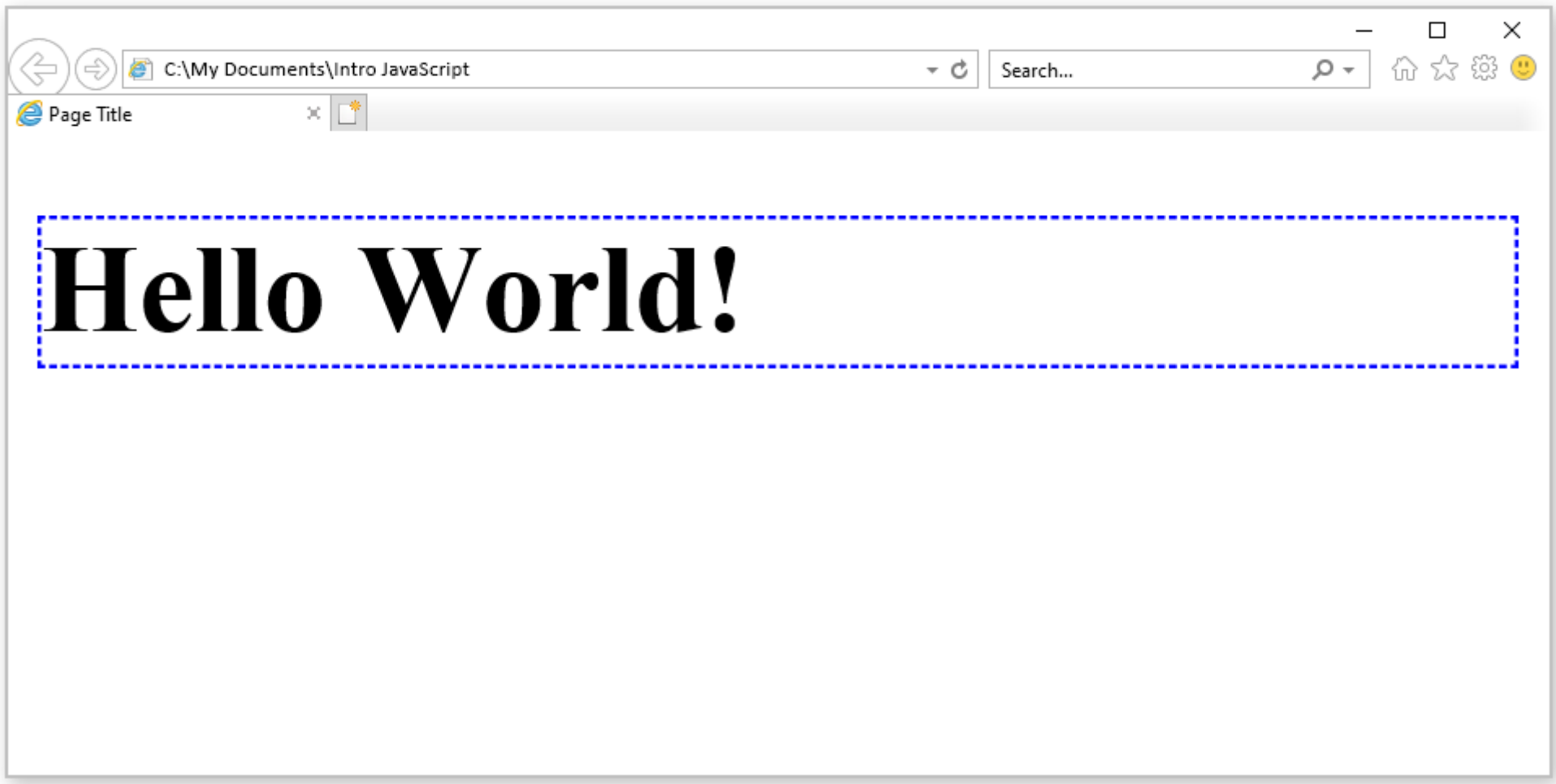
1. Open *InlineJavaScript.htm* in your editor.

2. Add a border around the h1 element with the following CSS code:

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

3. Save the page after adding that CSS inline style.

4. Then open the page in a browser. You'll see a blue, dashed border around the entire h1 element. And that way you'll be able to see how wide it really is.



Everything is "hot" within that dashed blue border.

Regardless of the event handler you've coded, if you engage your even handler anywhere within that box it will launch the JavaScript you've coded. In our example, with the ondblclick event handler in play, if you double-click anywhere inside that border the alert box will open. That's true whether the border is visible or not.

Adding a Span Tag

You can use event handlers with any tags. 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.

Let's go back to our original onclick event handler to test this, and we'll also use a span for the event handler.

Here are the Steps

1. Open *inlineJavaScript.htm* in your editor.

2. Remove the inline JavaScript and CSS code from the h1 tag.

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

3. Your h1 tag should now only have the phrase *Hello World!* displayed.

```
<h1>Hello World!</h1>
```

4. Now, add a span with an onclick event handler to show the alert box around the phrase *Hello World!* (inside the h1 tags) like this:

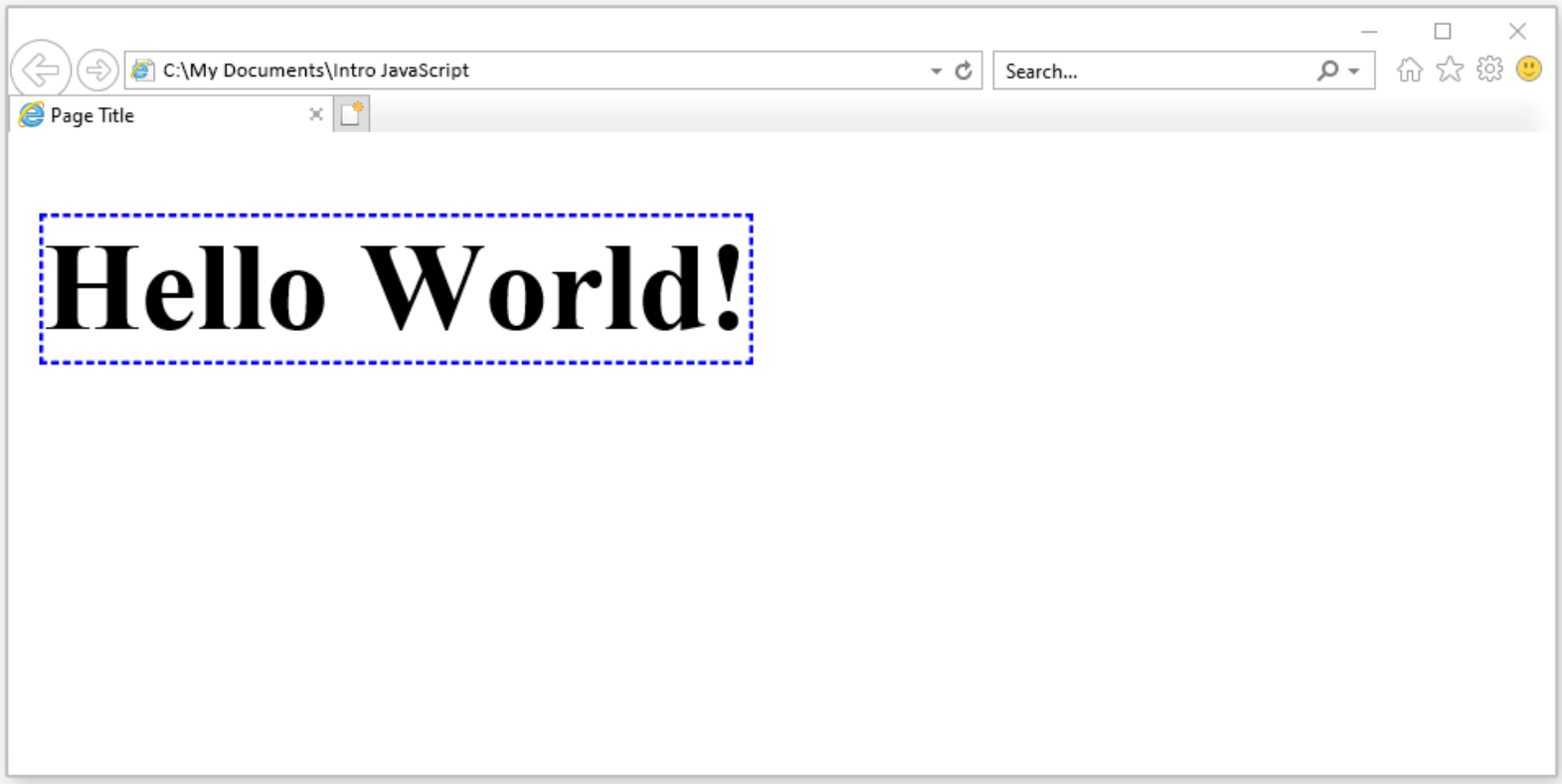
```
<h1>
<span onclick="alert('Hello World')" style="border:dashed 1px blue">Hello World!</span>
</h1>
```

5. Save the code in your editor.

6. Open or reload or refresh the page in the browser.

Sneak Peek

You'll see that while text is still an h1 heading (because it's in <h1> and </h1> tags), the h1 block element no longer triggers the JavaScript code. The hot spot now only encompasses the text. This is because the span tags inside the h1 element now hold the JavaScript code and event handler.



Now, if you click to the right of the *Hello World!* h1 heading text, it won't trigger the JavaScript code, because the whole h1 block is no longer the hotspot. The span is the hot spot now, so you have to actually click somewhere on the words *Hello World!* to get the alert box.

Execution is Everything

The point is that you have extremely fine control over when your JavaScript executes—and we've only scratched the proverbial surface. But no matter how advanced or complex you want your code and page interactivity to be, it all boils down to controlling *what* the JavaScript code does and *when* it does it. And the *when* part is all about event handlers.

Chapter 3: JavaScript Functions

Understanding JavaScript Functions

So far, you've worked with JavaScript code that's in the page body between <script> and </script> tags. You've also used inline JavaScript with event handlers inside HTML tags. Yet another place you can use JavaScript is between the <head> and </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. In this chapter, you'll learn how to create functions.

Function Syntax

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(){
...
}
```

Let's break down this syntax:

Text equivalent start.

Topic	Information
Function	The word *function* is the indicator that a function is about to be provided.
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, or underscores.

Topic	Information
()	The open and closing parentheses are a required part of the syntax and, as you'll learn later, can hold parameters through which values are passed to the function.
{. . .}	The area between the curly brackets, is where the JavaScript code resides. 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.

Read the topic in the first column. Then read the second column for the information.

Text equivalent stop.

Functions are generally placed in <script> and </script> tags in the head section of the page (that is, between the <head> and </head> tags).

```
<head>
<title>Page Title</title>
<script>
function name(){
...
}
</head>
</script>
```

We know it's a JavaScript function because it's in <script> and </script> tags (where JavaScript code gets written), and it starts with the word *function*. You can put any number of functions between a single pair of <script> and </script> tags.

Programmers 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.

Coding a JavaScript Function

Naming Your Function

The *name* part is something you get to make up. You could call your function *hello* or *goober* or *gomer* or *razzmatazz* or almost anything at all. There are only a few things to keep in mind:

- **It must start with a letter.** After that can contain any letters, numbers, or underscores.
- **No spaces or punctuation.** You cannot use spaces or other punctuation in a function name.
- **Function names are case-sensitive.** When naming a function, use uppercase and lowercase letters in a way that will be easy to remember in the future.

But for this example, we're going to use the name *hello*.

Make an Empty Function

This will be an empty function because we're not going to add between the opening and closing curly braces right now. But that's where you'll write any code that the function should execute when called.

Here are the Steps

1. Open the JavaScriptFunction.htm page you downloaded earlier in your editor.
2. Put the cursor after the </title> tag, and press **ENTER**.
3. Type *<script>*, press **ENTER** a couple of times again.
4. The type *</script>* to get an empty block of JavaScript code up in the head section of the page. You should end up with something like this:

```
<!DOCTYPE html>
<html>
<head>
 <title>Page Title</title>
 <script>



 </script>
</head>
<body>
</body>
</html>
```

Now you can add your function.

So now you have an area in the page into which you can place any number of JavaScript functions—between those <script> and </script> tags.

5. Put the cursor between the <script> and </script> tags.

6. Type *function hello(){}.* There should be no space between *name*, the left and right parentheses, and the left and right curly brackets.

7. Now, place your cursor between the curly brackets and press **ENTER** a couple of times. You should end up with an empty function named *hello*, as below.

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
<script>
function hello(){
}
</script>
</head>
<body>
</body>
</html>
```

Typing the </script> tag right after the typing the <script> tag, and typing the closing curly brace after the opening curly brace, are good habits to get into because it means you'll be less likely to forget to type them later. And forgetting the closing curly brace or closing </script> tag are very common errors for experienced programmers as well as for beginners.

Building Out Your Function

Adding Your Function

Right now, the function you just created doesn't actually do anything, because there's no code between the curly braces. You can put any number of lines of code in there. For now, let's keep it simple and add one line.

Here are the Steps

1. Put the cursor between the opening and closing curly braces of the function.

2. Type *alert("Hello JavaScript Function");* so now your code should look like this:

```
<!DOCTYPE html>
<html>
<head>
 <title>Page Title</title>
 <script>
  function hello(){
    alert("Hello JavaScript Function");
  }
 </script>
</head>
<body>
</body>
</html>
```

End-of-Line Semicolons are Optional

You might notice that we put a semicolon at the end of the line of code. Technically, this is not necessary, because each line of code in a JavaScript can end in a line break, or a semicolon, or both. But many programmers will put the semicolons in simply because they are used to doing that from other languages, and/or doing so makes them feel more confident in their code. Since you're likely to see this style out there in the real world, we'll follow suit and help you get familiar with this JavaScript code format.

3. Go ahead and save the changes you just made to *JavaScriptFunction.htm*.

4. To open the page in your browser, double-click its icon on the *Intro JavaScript* folder. This time you'll get nothing but an empty white page. Don't worry, this is exactly what we want!

Calling a Function

With JavaScript functions, you don't want them to do anything until you actually *call* them. Typically, you call a function from an event handler in a tag. Calling the function executes all the code in the function.

Let's give it a try!

Here are the Steps

1. Switch to (or open) *JavaScriptFunction.htm* in your editor.

2. Place your cursor in the body of the page (between the <body> and </body> tags).

3. Type (or copy and paste) the following tag:

```
<input type="button" value="Click Me" onclick="hello()">
```

Sneak Peek

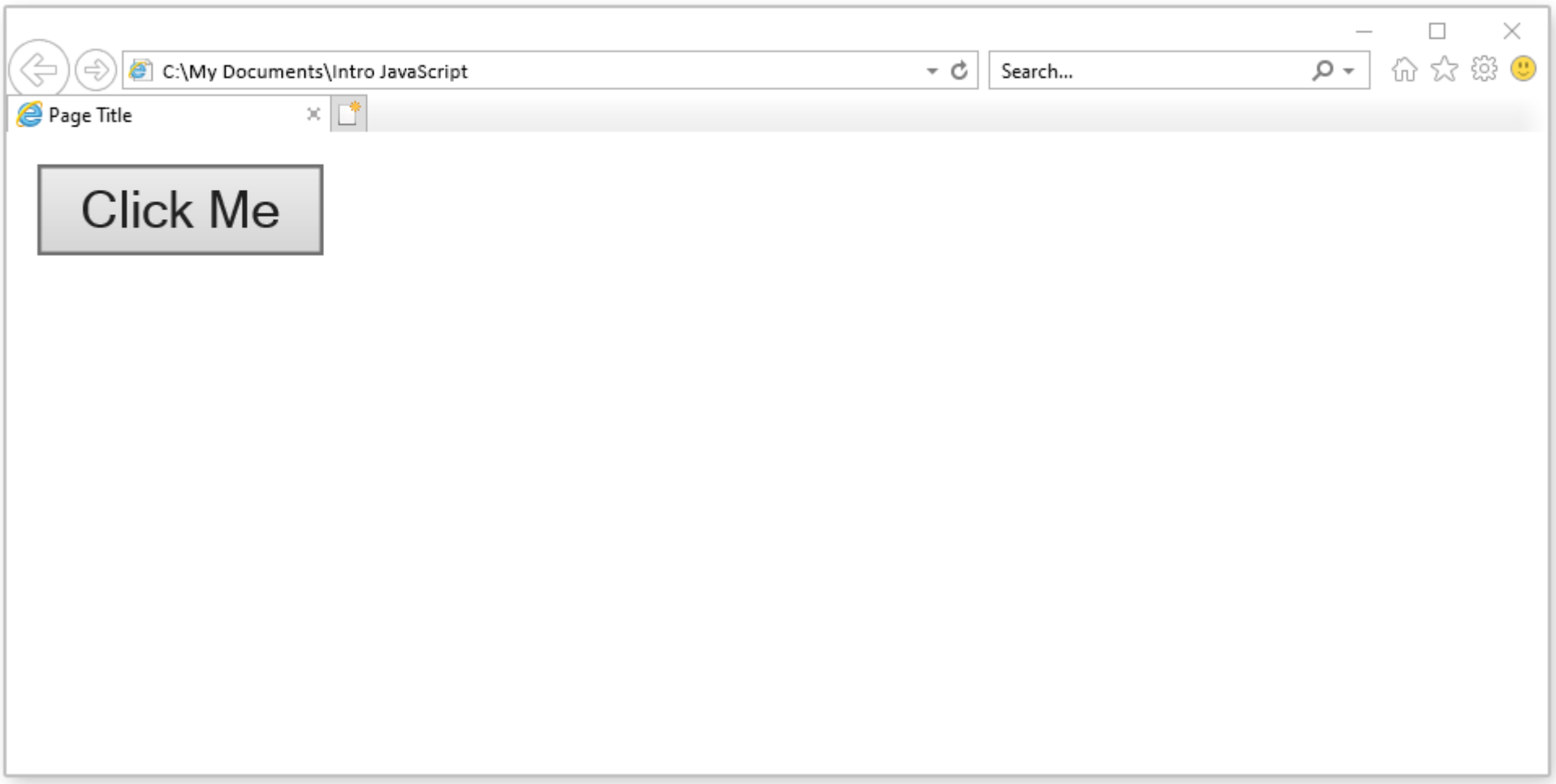
The complete *JavaScriptFunction.htm* page should look like this in your editor:

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

4. Save the page.

5. Then open it in a browser again (or reload or refresh it in the browser). This time, the page isn't blank. This time it shows a button similar to the one below (the exact appearance of the button will vary

depending on the browser you're using). In the browser, the input tag and attributes render a button on the page.



Input Tag and Attributes

The button appears because of the HTML <input> tag that you added to the page. This example is different from our earlier onclick example because we used an <input> tag rather than an <h1> tag. However, you can use the onclick event handler in just about any tag that creates visible element on the page. 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.

<input> Tag

If this is the first time you've ever seen an <input> tag, don't worry about that. It's just a tag that creates an element, and you'll be seeing plenty more examples in upcoming lessons.

type = "button"

Inside the <input> tag the type="button" make it look like a button. You cannot control the look and feel of the button.

value = "Click Me"

The value="Click Me" put the words *Click Me* on the button. You can control the words that appear inside the box, so you can substitute any words you like for *Click M*e.

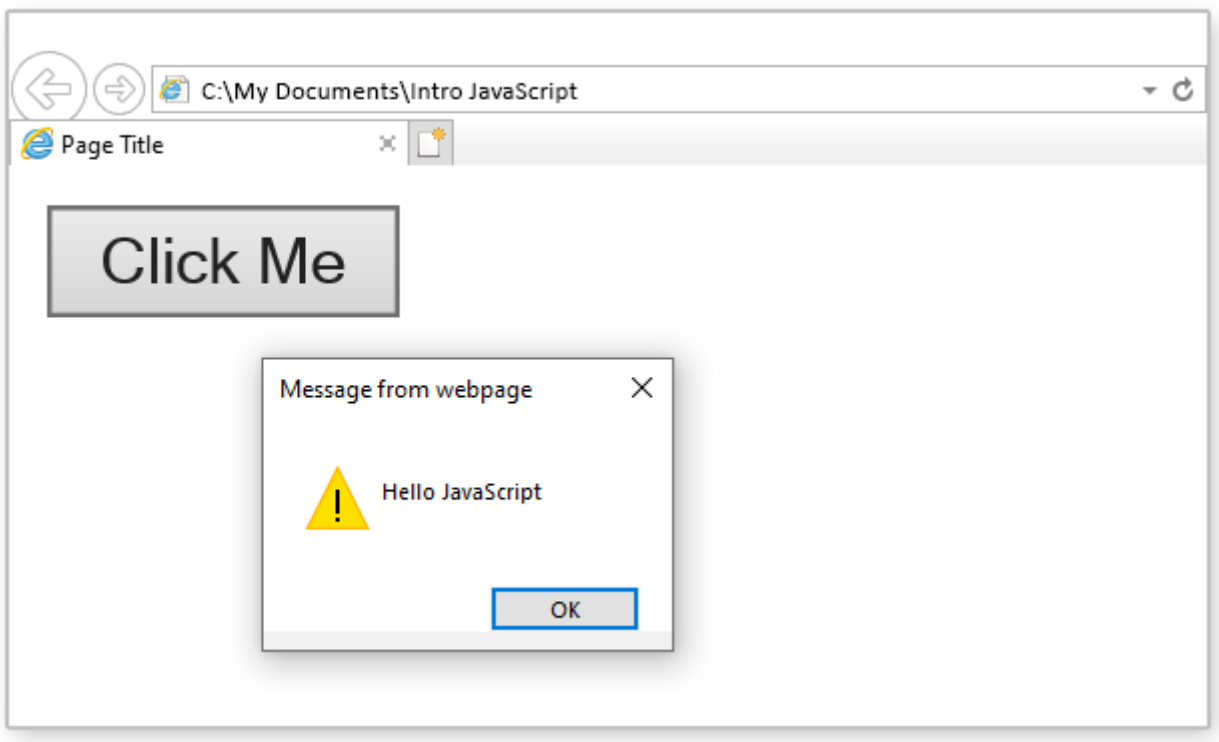
onclick = "hello()"

The <input> tag also contains an onclick event handler, and the value assigned to that is the same as the name of the function, onclick="hello()".

Remember

JavaScript is case-sensitive, so watch all of your uppercase and lowercase letters carefully!

6. Go ahead and click the "Click Me" button. An alert box should appear. The onclick event handler in the input tag calls the hello() function when you click the button, so the alert box appears on the screen. As usual, the exact appearance of that alert box depends on the browser you're using.



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 that's not important. What's important here is that you were able to control *when* the function was executed by tying it to an event handler in an HTML tag.

Right now your focus should be on understanding the concept and the syntax because they will play out over and over again—not just throughout this course, but for your entire development career. The better you understand some of this basic stuff that's used throughout the language, the more you'll be able to see the simplicity behind more-advanced examples you'll be encountering later.

Now, let's wrap things up for this lesson and summarize what you've learned.

Review

This lesson was mostly about controlling *when* your JavaScript executes. To summarize the key points:

- **Handing Events:** Here you learned that taking full advantage of JavaScript's many capabilities requires not only that you control *what* your JavaScript code does, but *when*. JavaScript code that's in the body of the page between <script> and </script> tags executes when the page first opens in the browser. To control exactly when a piece of JavaScript code executes, we use event handlers inside HTML tags.

- **Working with Event Handlers:** The event handler defines what the user must do to the element to make the code execute. In this section you experimented with a number of different event handlers, and customized your event hot spot.

- **JavaScript Functions:** You've discovered some syntax and organizational issues that are at the very core of learning and using JavaScript successfully. You can organize JavaScript code into chunks called *functions*. Functions are placed between <script> and </script> tags in the head section above the page body (between the <head> and </head> tags). Each function is identified by the word *function* followed by a name of your own choosing. All the code that's executed when the function is called must be placed in curly braces after the function name. Functions are executed when called by event handlers in tags.

Don't forget to complete the Lesson 2 assignment, and you'll also want to test your knowledge by taking the Lesson 2 quiz after completing this lesson. And then we'll see you in Lesson 3, where you'll learn more about adding interactivity to your page with JavaScript.

Lesson 2 Assignment

Create and Call a JavaScript Function

For this assignment, you're going to create a page that contains the following elements:

1. A javascript function.

2. A button, that when clicked, executes that function.

Here are the Steps

1. Create a new web page (.htm or .html) with your code editor with all the basic "boilerplate" tags.
2. Add a page title of "Assignment 2"(or feel free to get creative).
3. Add a JavaScript function named myfunction() inside the head tags. Don't forget the script tags!
4. Inside the function, add code to display the words "You did it!" in an alert box.
5. Add an HTML button to the body of the web page.
6. Add an event handler that says to call the function named myfunction() when someone clicks that button.
7. Test the page in a browser to verify that it works.

If you have any trouble, you can watch the following video doing it with Visual Studio Code:

Note

You don't need to use the Visual Studio Code Editor (https://code.visualstudio.com/) But many people use it.

Download Box

To check your code, you can download the correct code here:

Download L02-Assignment.zip

907 Bytes (Gigabytes) ZIP

Lesson 2 Resources for Further Learning

Global Event Attributes (https://www.w3schools.com/tags/ref_eventattributes.asp)

https://www.w3schools.com/tags/ref_eventattributes.asp

Here's a good site to add to your browser's Bookmarks or Favorites, as it lists all of the event attributes that you can use in JavaScript.

JavaScript Syntax (https://www.w3schools.com/js/js_syntax.asp)

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

Here is a good quick reference on JavaScript syntax and the use of semicolons.

JavaScript Cheat Sheet (https://cheatography.com/davechild/cheat-sheets/javascript/)

https://cheatography.com/davechild/cheat-sheets/javascript/

Here you'll find a handy downloadable and printable JavaScript cheat sheet of many aspects of the JavaScript language.

JavaScript (https://developer.mozilla.org/en-US/docs/Web/JavaScript)

https://developer.mozilla.org/en-US/docs/Web/JavaScript

This link takes you to the home page for Mozilla's thorough JavaScript documentation and related links and tutorials.

HTML Reference—(HTML5 Compliant) (https://www.w3schools.com/tags/default.asp)

https://www.w3schools.com/tags/default.asp

If you don't already have a good reference for looking up HTML tags on the fly, you'll want to add this one to your browser's Bookmarks or Favorites because everything in Web development revolves around those HTML tags.

How to Enable JavaScript in Your Browser (https://enable-javascript.com/)

https://enable-javascript.com/

Most websites use some JavaScript, so JavaScript is enabled, by default, in all browsers. But if you ever come across a browser that won't do any JavaScript at all, this page should help to adjust settings accordingly.

Where to Place JavaScript (http://www.tizag.com/javascriptT/javascriptheadnbody.php)

http://www.tizag.com/javascriptT/javascriptheadnbody.php

Here's another good tutorial that reinforces some things you learned in today's lesson.