



Romel Cabiling ▾



[Home](#)

[Home](#) > [My courses](#) > [121 - WEB101 / CCS3218](#) > [13 JavaScript Basics](#) > [Lesson Proper for Week 13](#)

Lesson Proper for Week 13

WHAT IS JAVASCRIPT?

JavaScript is a programming language that adds interactivity and custom behaviors to our sites. It is a client-side scripting language, which means it runs on the user's machine and not on the server, as other web programming languages such as PHP and Ruby do. That means JavaScript (and the way we use it) is reliant on the browser's capabilities and settings. It may not even be available at all, because either the user has chosen to turn it off or because the device does not support it, which good developers keep in mind and plan. JavaScript is also what is known as a dynamic and loosely typed programming language.

JavaScript is a lightweight but incredibly powerful scripting language. We most frequently encounter it through our browsers, but JavaScript has snuck into everything from native applications to PDFs to eBook's. Even web servers themselves can be powered by JavaScript.

As a dynamic programming language, JavaScript doesn't need to be run through any form of compiler that interprets our human-readable code into something the browser can understand. The browser effectively reads the code the same way we do and interprets it on the fly.

JavaScript is also loosely typed. All this means is that we don't necessarily have to tell JavaScript what a variable is. If we're setting a variable to a value of 5, we don't have to programmatically specify that variable as a number; that is, 5 is a number, and JavaScript recognizes it as such.

What JavaScript Can Do

Most commonly we'll encounter JavaScript as a way to add interactivity to a page. Whereas the "structural" layer of a page is our HTML markup, and the "presentational" layer of a page is made up of CSS, the third "behavioral" layer is made up of our JavaScript. All of the elements, attributes, and text on a web page can be

accessed by scripts using the DOM (Document Object Model). We can also write scripts that react to user input



accessed by scripts using the DOM (Document Object Model), we can also write scripts that react to user input, altering either the contents of the page, the CSS styles, or the browser's behavior on the fly.

You've likely seen this in action if you've ever attempted to register for a website, entered a username, and immediately received feedback that the username you've entered is already taken by someone else (FIGURE 21-1). The red border around the text input and the appearance of the "sorry, this username is already in use" message are examples of JavaScript altering the contents of the page. Blocking the form submission is an example of JavaScript altering the browser's default behavior. Ultimately, verifying this information is a job for the server—but JavaScript allows the website to make that request and offer immediate feedback without the need for a page reload.

Whoops! Some errors occurred.

- That username is already in use.
- Email confirmation doesn't match

Username Must be at least 4 characters

Email

Confirm Email

Password

Confirm Password

FIGURE 21-1. JavaScript inserts a message, alters styles to make errors apparent, and blocks the form from submitting. It can also detect whether the email entries match, but the username would more likely be detected by a program on the server.

In short, JavaScript allows you to create highly responsive interfaces that improve the user experience and provide dynamic functionality, without waiting for the server to load up a new page. For example, we can use JavaScript to do any of the following:

- Suggest the complete term a user might be entering in a search box as he types. You can see this in action on Google.com (FIGURE 21-2).



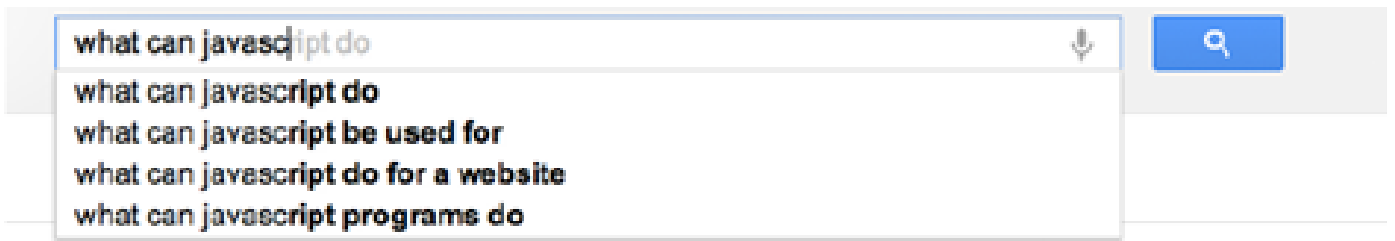


FIGURE 21-2. Google.com uses JavaScript to automatically complete a search term as it is typed in.

- Request content and information from the server and inject it into the current document as needed, without reloading the entire page—this is commonly referred to as “Ajax.”
- Show and hide content based on a user clicking a link or heading, to create a “collapsible” content area (FIGURE 21-3).

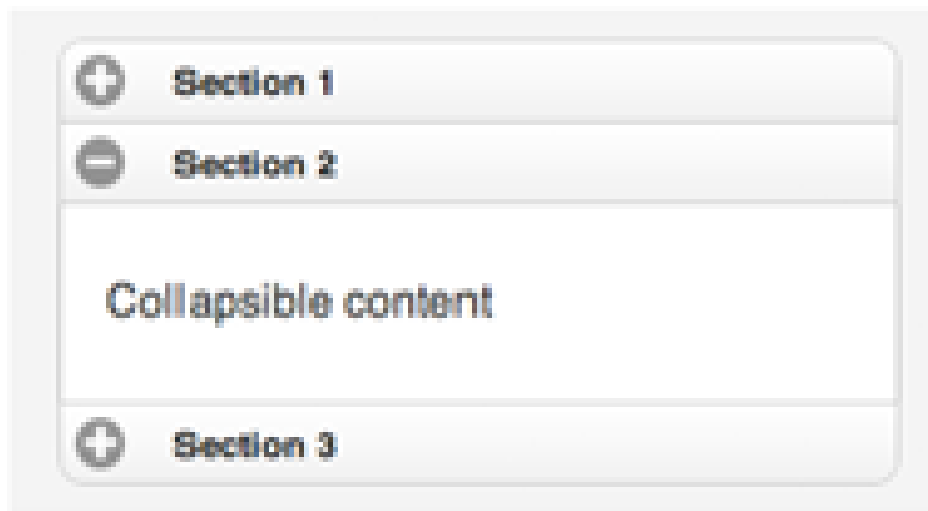


FIGURE 21-3. JavaScript can be used to reveal and hide portions of content.

- Test for browsers’ individual features and capabilities. For example, one can test for the presence of “touch events,” indicating that the user is interacting with the page through a mobile device’s browser, and add more touch-friendly styles and interaction methods.
- Fill in gaps where a browser’s built-in functionality falls short, or add some of the features found in newer browsers to older browsers. These kinds of scripts are usually called shims or polyfills.
- Load an image or content in a custom-styled “lightbox”—isolated on the page with CSS—after a user clicks a thumbnail version of the image (FIGURE 21-4).





FIGURE 21-4. JavaScript can be used to load images into a lightbox-style gallery.

ADDING JAVASCRIPT TO A PAGE

As with CSS, you can embed a script right in a document or keep it in an external file and link it to the page. Both methods use the **script** element.

NOTE

For documents written in the stricter XHTML syntax, you must identify the content of the script element as CDATA by wrapping the code in the following wrapper:

```
<script type="text/javascript">
  // <![CDATA[
    JavaScript code goes here
  // ]]>
</script>
```

Embedded Script

To embed a script on a page, just add the code as the content of a **script** element:



To embed a script on a page, just add the code as the content of a **script** element.

```
<script>
```

```
    ... JavaScript code goes here
```

```
</script>
```

External Scripts

The other method uses the **src** attribute to point to a script file (with a *.js* suffix) by its URL. In this case, the **script** element has no content:

```
<script src="my_script.js"></script>
```

The advantage to external scripts is that you can apply the same script to multiple pages (the same benefit external style sheets offer). The downside, of course, is that each external script requires an additional HTTP request of the server, which slows down performance.

Script Placement

The **script** element can go anywhere in the document, but the most common places for scripts are in the **head** of the document and at the very end of the **body**. It is recommended that you don't sprinkle them throughout the document, because they would be difficult to find and maintain.

For most scripts, the end of the document, just before the **</body>** tag, is the preferred placement because the browser will be done parsing the document and its DOM structure:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
    <meta charset="utf-8">
```

```
</head>
```

```
<body>
```

```
    ...contents of page...
```

```
    <script src="script.js"></script>
```

```
</body>
```

```
</html>
```

Consequently, that information will be ready and available by the time it gets to the scripts, and they can execute faster. In addition, the script download and execution blocks the rendering of the page, so moving the script to the bottom improves the perceived performance.

However, in some cases, you might want your script to do something before the body completely loads. Putting it in the **head** will result in better performance.



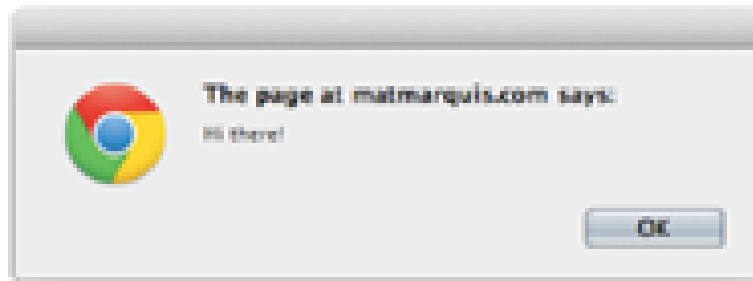
THE ANATOMY OF A SCRIPT

Originally, JavaScript's functionality was mostly limited to crude methods of interaction with the user. We could use a few of JavaScript's built-in functions (FIGURE 21-5) to provide user feedback, such as **alert()** to push a notification to a user, and **confirm()** to ask a user to approve or decline an action. To request the user's input, we were more or less limited to the built-in **prompt()** function.

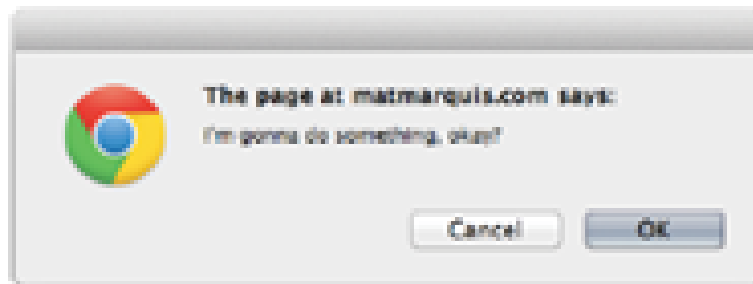
In order to take advantage of these interaction methods, we have to first understand the underlying logic that goes into scripting. These are logic patterns common to all manner of programming languages, although the syntax may vary. To draw a parallel between programming languages and spoken languages: although the vocabulary may vary from one language to another, many grammar patterns are shared by the majority of them.



```
alert("Hi there");
```



```
confirm("I'm gonna do something, okay?");
```



```
prompt("What should I do?");
```

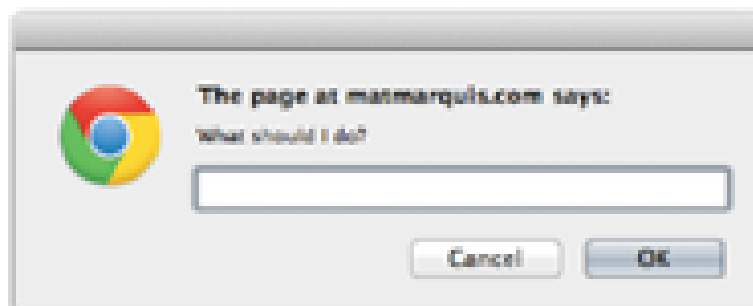


FIGURE 21-5. Built-in JavaScript functions: **alert()** (top), **confirm()** (middle), and **prompt()** (bottom).

The Basics

It is important to know that JavaScript is case-sensitive. A variable named **myVariable**, a variable named **myvariable**, and a variable named **MYVariable** will be treated as three different objects.

Also, whitespace such as tabs and spaces is ignored, unless it's part of a string of text and enclosed in quotes. All of the character spaces added to scripts such as the ones in this chapter are for the benefit of humans—they make reading through the code easier. JavaScript doesn't see them.

§ Statements



A script is made up of a series of statements. A statement is a command that tells the browser what to do. Here is a

A script is made up of a series of statements. A statement is a command that tells the browser what to do. Here is a simple statement that makes the browser display an alert with the phrase "Thank you":

```
alert("Thank you.");
```

The semicolon at the end of the statement tells JavaScript that it's the end of the command, just as a period ends a sentence. According to the JavaScript standard, a line break will also trigger the end of a command, but it is a best practice to end each statement with a semicolon.

§ Comments

JavaScript allows you to leave comments that will be ignored at the time the script is executed, so you can provide reminders and explanations throughout your code. This is especially helpful if this code is likely to be edited by another developer in the future.

There are two methods of using comments. For single-line comments, use two slash characters (//) at the beginning of the line. You can put single-line comments on the same line as a statement, as long as the comment comes after the statement. It does not need to be closed, as a line break effectively closes it.

```
// This is a single-line comment.
```

Multiple-line comments use the same syntax that you've seen in CSS. Everything within the /* */ characters is ignored by the browser. You can use this syntax to "comment out" notes and even chunks of the script when troubleshooting.

```
/* This is a multiline comment. Anything between these sets of characters will be completely ignored when the script is executed. This form of comment needs to be closed. */
```

I'll be using the single-line comment notation to add short explanations to example code, and we'll make use of the alert() function we saw earlier (FIGURE 21-5) so we can quickly view the results of our work.

§ Variables

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar(\$) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.
4. It may not contain special characters (e.g., ! , / \ + * =).

§ Data Types

JavaScript provides different data types to hold different types of values like numbers, strings, objects and more.

There are two types of data types in JavaScript.

1. Primitive data type



1. Primitive data type

There are five types of primitive data types in JavaScript. They are as follows:

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

2. Non-primitive (reference) data type

The non-primitive data types are as follows:

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

JavaScript is a dynamic type language, means you don't need to specify type of the variable because it is dynamically used by JavaScript engine. You need to use var here to specify the data type. It can hold any type of values such as numbers, strings etc.

§ Operators

JavaScript operators are symbols that are used to perform operations on operands. An operator is a mathematical symbol that produces a result based on two values (or variables).

There are following types of operators in JavaScript.

1. Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on the operands. The following operators are known as JavaScript arithmetic operators.

Operator	Description	Example
+	Addition	10+20 = 30
-	Subtraction	20-10 = 10
*	Multiplication	10*20 = 200
/	Division	20/10 = 2
%	Modulus (Remainder)	20%10 = 0
++	Increment	var a=10; a++; Now a = 11



++	Increment	var a=10; a++; Now a = 11
--	Decrement	var a=10; a--; Now a = 9

2. Comparison (Relational) Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

3. Bitwise Operators

Bitwise operators are **operators** (just like +, *, &&, etc.) that operate on ints and uints at the binary level. This means they look directly at the binary digits or bits of an integer.

The bitwise operators perform bitwise operations on operands. The bitwise operators are as follows:

Operator	Description	Example
&	Bitwise AND	(10==20 & 20==33) = false
	Bitwise OR	(10==20 20==33) = false
^	Bitwise XOR	(10==20 ^ 20==33) = false
~	Bitwise NOT	(~10) = -10
<<	Bitwise Left Shift	(10<<2) = 40
>>	Bitwise Right Shift	(10>>2) = 2
>>>	Bitwise Right Shift with Zero	(10>>>2) = 2

4. Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
	Logical OR	(10==20 20==33) = false



	Logical OR	(10==20 20==30) = false
!	Logical Not	!(10==20) = true

5. Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
=	Multiply and assign	var a=10; a=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

6. Special Operators

The following operators are known as JavaScript special operators.

Operator	Description
(?:)	Conditional Operator returns value based on the condition. It is like if-else.
,	Comma Operator allows multiple expressions to be evaluated as single statement.
delete	Delete Operator deletes a property from the object.
in	In Operator checks if object has the given property
instanceof	checks if the object is an instance of given type
new	creates an instance (object)
typeof	checks the type of object.
void	it discards the expression's return value.
yield	checks what is returned in a generator by the generator's iterator.

§ if/else statements

The JavaScript if-else statement is used to execute the code whether condition is true or false. There are three forms of if statement in JavaScript.

1. If Statement - It evaluates the content only if expression is true. The signature of JavaScript if statement is given below.

```
if(expression){
```

```
//content to be evaluated
```



```
//content to be evaluated
```

```
}
```

2. If else statement - It evaluates the content whether condition is true or false. The syntax of JavaScript if-else statement is given below.

```
if(expression){
```

```
//content to be evaluated if condition is true
```

```
}
```

```
else{
```

```
//content to be evaluated if condition is false
```

```
}
```

3. If else if statement - It evaluates the content only if expression is true from several expressions. The signature of JavaScript if else if statement is given below.

```
if(expression1){
```

```
//content to be evaluated if expression1 is true
```

```
}
```

```
else if(expression2){
```

```
//content to be evaluated if expression2 is true
```

```
}
```

```
else if(expression3){
```

```
//content to be evaluated if expression3 is true
```

```
}
```

```
else{
```

```
//content to be evaluated if no expression is true
```

```
}
```

§ Switch

The **JavaScript switch statement** is used *to execute one code from multiple expressions*. It is just like else if statement that we have learned in previous page. But it is convenient than *if..else..if* because it can be used with numbers, characters etc.

The signature of JavaScript switch statement is given below.

```
switch(expression){
```

```
case value1:
```



case value1:

code to be executed;

break;

case value2:

code to be executed;

break;

.....

default:

code to be executed if above values are not matched;

}

§ Loops

The JavaScript loops are used *to iterate the piece of code* using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.

There are four types of loops in JavaScript.

1. for loop - The **JavaScript for loop** *iterates the elements for the fixed number of times*. It should be used if number of iteration is known. The syntax of for loop is given below.

for (initialization; condition; increment)

{

code to be executed

}

2. while loop - The **JavaScript while loop** *iterates the elements for the infinite number of times*. It should be used if number of iteration is not known. The syntax of while loop is given below.

while (condition)

{

code to be executed

}

3. do-while loop - The **JavaScript do while loop** *iterates the elements for the infinite number of times* like while loop. But, code is *executed at least* once whether condition is true or false. The syntax of do while loop is given below.

do{

code to be executed



code to be executed

```
}while (condition);
```

§ Functions

JavaScript functions are used to perform operations. We can call JavaScript function many times to reuse the code.

Advantage of JavaScript function

There are mainly two advantages of JavaScript functions.

1. **Code reusability:** We can call a function several times so it save coding.
2. **Less coding:** It makes our program compact. We don't need to write many lines of code each time to perform a common task.

The syntax of declaring function is given below.

```
function functionName([arg1, arg2, ...argN]){  
    //code to be executed  
}
```



◀ Preliminary Activity for Week 13

Jump to...



Analysis, Application, and Exploration for Week 13 ▶



Navigation

Home



Dashboard

Site pages

My courses

121 - CC106

121 - BPM101 / DM103

121 - OAELEC2

121 - ITE3

121 - MUL101

121 - ITSP2B

121 - WEB101 / CCS3218







Participants



Grades

General



- 01 Introduction to Internet and World Wide Web
- 02 Full Stack Development Overview
- 03 The Basics of HTML
- 04 Mark Up Text
- 05 HTML Link, Image and Table
- 06 - Preliminary Examination
- 07 HTML Block, Inline Element and Forms Control
- 08 Introduction To Cascading Style Sheets
- 09 The Font Style Properties
- 10 The List and Table style properties
- 11 Advanced CSS
- 12 - Midterm Examination
- 13 JavaScript Basics
-  Preliminary Activity for Week 13
-  **Lesson Proper for Week 13**
-  Analysis, Application, and Exploration for Week 13
-  Generalization for Week 13
-  Evaluation for Week 13
-  Assignment for Week 13
- 14 JavaScript Objects
- 15 Introduction to PHP
- 16 PHP Control Statement and Function
- 17 PHP Array and String

Courses

Fair Warning

NOTICE: Please be reminded that it has come to the attention of the Publishing Team of eLearning Commons that learning materials published and intended for ***free use only by students and faculty members within the eLearning Commons network were UNLAWFULLY uploaded in other sites without due and proper permission.***

PROSECUTION: Under Philippine law (Republic Act No. 8293), copyright infringement is punishable by the following: Imprisonment of between 1 to 3 years and a fine of between 50,000 to 150,000 pesos for the first offense. Imprisonment of 3 years and 1 day to six years plus a fine of between 150,000 to 500,000 pesos for the second offense.

COURSE OF ACTION: Whoever has maliciously uploaded these concerned materials are hereby given an ultimatum to take it down within 24-hours. Beyond the 24-hour grace period, our Legal Department shall initiate the proceedings in coordination with the National Bureau of Investigation for IP Address tracking, account owner identification, and filing of cases for prosecution.



2nd Semester Enrollment

A banner for 2nd Semester Enrollment. The background is a blue-tinted image of a multi-story building with a 'BCP' sign on top. Overlaid text includes 'visit www.bcp.edu.ph' in blue, 'Enrollment registration is now Ongoing' in large red letters, 'For 2nd Semester SY 2021 - 2022' in white on a blue background, and 'We are accepting new students, returnees and transferees.' in white. A quote 'Be trained to be the best, Be linked to success' is next to a circular logo. At the bottom, there is an email icon and 'bcp-inquire@bcp.edu.ph' and a phone icon and '(8)442-8601 | (8)518-8050'.

visit www.bcp.edu.ph

Enrollment registration is now Ongoing





For 2nd Semester SY 2021 - 2022

We are accepting new students, returnees and transferees.

"Be trained to be the best,
Be linked to success"

 bcp-inquire@bcp.edu.ph  (8)442-8601 | (8)518-8050

Activities

-  Assignments
-  Forums
-  Quizzes
-  Resources

Bestlink College of the Philippines
College Department

Powered by [eLearning Commons](#)

