

JavaScript Introduction

JAVASCRIPT is a powerful client-side programming language that allows for the creation of interactive and dynamic websites. Understanding the basic principles of JavaScript is an essential skill for Web developers.



Objective:



Demonstrate basic JavaScript programming fundamentals: use of decisions, loops, arrays, event handlers, and validating JavaScript code.

Key Terms:



alert function	function parameter list	parseInt and parseFloat functions
arrays	functions	prompt function
assignment statement	if-else	relational operators
break statement	JavaScript (JS)	reserved words/keywords
comment statements	literals	string/text literals
decisions	local variables	subscript
document object	logical operators	switch statement
escape sequence	loop structures	variables
expression	NaN	while loop structure
for loop structure	null	write method
function	numeric literals	
function argument	object	
function body	object methods	
function header	operator	

Understanding JavaScript

JavaScript (JS) is a cross-platform, object-oriented script language from Netscape and is easier and faster to code than compiled languages (e.g., C and C++) but often takes longer to process than compiled languages. In webpage content, it is coded in HTML. Style information is maintained via CSS. While CSS3 allows animations and transitions to be placed on a

webpage, the content in a webpage is static. Elements of dynamic content are possible by using programs on a Web server. Therefore, when data in a document needs to be changed, a request is sent to the Web server, which then sends back new data to the client. The main problem with this approach is the time taken for the Web server roundtrip.

JAVASCRIPT PROGRAMMING FUNDAMENTALS

The main purpose of JavaScript is to write statements that access the webpage and change its behavior and content. JavaScript uses objects, methods, and properties to accomplish this task. Interactivity may be added in a webpage by using JavaScript. It is designed for use in webpages and has many features that make working with Web content easy. JavaScript code is placed inside a webpage inside `<script>` and `</script>` tags. It may also be placed in a separate file and linked to HTML files. The Web browser software interprets HTML, CSS, and JavaScript code. It is important to know about JavaScript components; variables and functions; expressions and operators; and code debugging.

JavaScript Components

JavaScript components include various entities in a JavaScript program. Each provides a specific functionality in the language.

Comment Statements

Comment statements are comments placed in JavaScript code to provide documentation within a set of program code. Comment statements are not converted into machine code. While comment statements are not required in a program, good programming standards specify that programs should contain comment lines with details (e.g., the program author and purpose of the program). Two ways to create comment lines in JavaScript are:

- ◆ The `//` symbol comments a line in a set of JavaScript code.
- ◆ `/*` starts a block of comment code. `*/` ends the comment block. There may be multiple lines of code between the start comment block and the end comment block characters.

Reserved Words/Keywords

Reserved words/keywords are terms with pre-defined meanings in the language. For example, the reserved word “alert” is used to display data. The complete list of reserved words in JavaScript can be found at <http://www.javascripter.net/faq/reserved.htm>. It is important to ensure that no programmer-defined entities have the same name as a reserved word.

Literals

Literals are entities that do not change value. For instance, each time code is executed, the value remains the same. The two types of literals used in JavaScript are numeric literals and string/text literals.



FURTHER EXPLORATION...

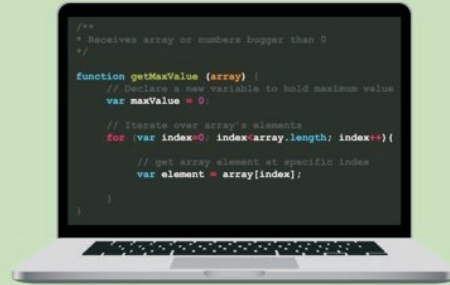
ONLINE CONNECTION: Strings and Patterns in JavaScript

JavaScript contains many features to work with string data. There are functions that can convert a string value into upper-case and lowercase. Individual characters can be retrieved from a string. Also, JavaScript supports the use of pattern matching for string variables. Pattern matching is implemented using regular expressions. To learn more, visit:

http://www.w3schools.com/jsref/jsref_obj_string.asp

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

JAVASCRIPT



JavaScript (JS) is a cross-platform, object-oriented script language from Netscape and is easier and faster to code than compiled languages (e.g., C and C++). However, it often takes longer to process than compiled languages. In webpage content, it is coded in HTML, and style information is maintained via CSS.

Numeric literals are entities written as numbers in a program. For example, 12, -678, and 2.34 are all considered numeric literals. They may be positive, 0 (zero), or negative. They may be whole numbers or numbers with values following the decimal places. Numeric literals should only consist of numeric digits with an optional decimal point and numeric digits after the decimal point. They cannot contain the comma symbol or a currency symbol.

String/text literals are entities that contain a mixture of alphabets, numbers, and symbols. String literals must be surrounded by double quotes or single quotes. For example, “Spring 2015,” “Jim,” “JAKHS(&★,” and “123.45” are all considered string literals. The string literal “123.45” holds a numeric literal inside the double quotes. However, the system will treat it as a string literal as it is surrounded by double quotes. [NOTE: In this material, we always use double quotes to set up string literals.]

Variables

Variables are programmer-defined entities in a program that can hold data. The value held in a variable can change during the course of program execution. Variables are required to have a valid name.

Variable names are case sensitive. Rules exist for setting up a valid variable name. The name of a valid variable name:

- ◆ May be composed of alphabets, numbers, and the “_” (underscore) character
- ◆ May not begin with a number (Therefore, naming a variable name 3rdTest is not valid.)

- ◆ May not contain an embedded blank (So a variable may not be named test 3. Instead, it should be named test_3. The under-score can be used to take the position of a space to visually separate the various parts of the variable name.)
- ◆ May not have the same name as reserved words or objects (To avoid conflicts with reserved words, developers create variable names that indicate their function within the program.)

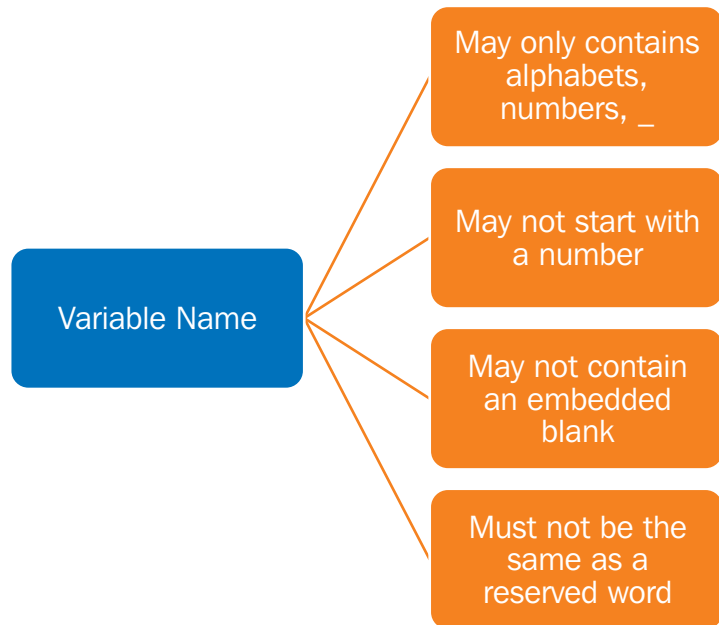


FIGURE 1. Variable names are case sensitive. There are rules for setting up a valid variable name.

JavaScript is a loosely typed language. Variables may be set up without specifying the type of data to be maintained in the variable. The data type of an object can be String, Number, Boolean, Array, or Object.

There is no requirement to declare a JavaScript variable before it is used. However, it is considered good programming practice to declare variables before they are used in a program. A variable declaration statement contains the keyword “var” followed by a space and the name of the variable.

Functions

Functions are items used in JavaScript to “bundle” sections of code that can be reused. Programmers can use functions provided by JavaScript. Developers may create their own functions as well. Functions are invoked as follows:

- ◆ Key in the name of the function.
- ◆ Follow key in with the (symbol (e.g., left parenthesis).
- ◆ Then key in the data required by the function or the **function argument**. Not all functions require arguments.
- ◆ Finally, end with the) symbol (e.g., right parenthesis).

Variables and Functions

In JavaScript code, developers may set up variables, manipulate their values, and then make use of them. An **assignment statement** is a declaration that populates a variable.

An **alert function** is a feature that displays values in variables.

```

1  /*examples of assignment
2  statements*/
3
4  var myname, current_year, my_gpa
5
6  myname = "John" //"John" is placed in the myname variable
7  current_year = 2015
8  my_gpa = 3.9

```

FIGURE 2. Lines 1 and 2 show comment statements using the block comment characters of /* and */ to begin and end a comment block, respectively. Line 6 also shows a comment placed at the end of a line of JavaScript code. Line 4 uses the “var” keyword to declare three variables. Lines 6, 7, and 8 show assignment statements. In line 6, the string literal “John” is placed in the myname variable. In line 7, the numeric literal 2015 is placed in the current_year variable, and line 8 shows the placement of a float literal in the my_gpa variable.

```

<!DOCTYPE html>
<head>
  <meta charset= "utf-8" />
  <title>Myfirst HTML5 page</title>
  <script>
    /*examples of assignment
    statements*/

    var myname, current_year, my_gpa//declarations

    myname = "John" //Place "John" in myname
    current_year = 2015
    my_gpa = 3.9

    alert(myname)
    alert("The current year is " + current_year)
    alert("My gpa is " + my_gpa)
  </script>
</head>
<body>

  <p>Testing JavaScript</p>
</body>
</html>

```

FIGURE 3. This image shows the earlier code inside an HTML document. JavaScript statements are placed inside a <script> block. After the comments, declaration, and assignment statements, there are three alert statements. Each of these is an invocation of the alert function, with one argument passed into the function.

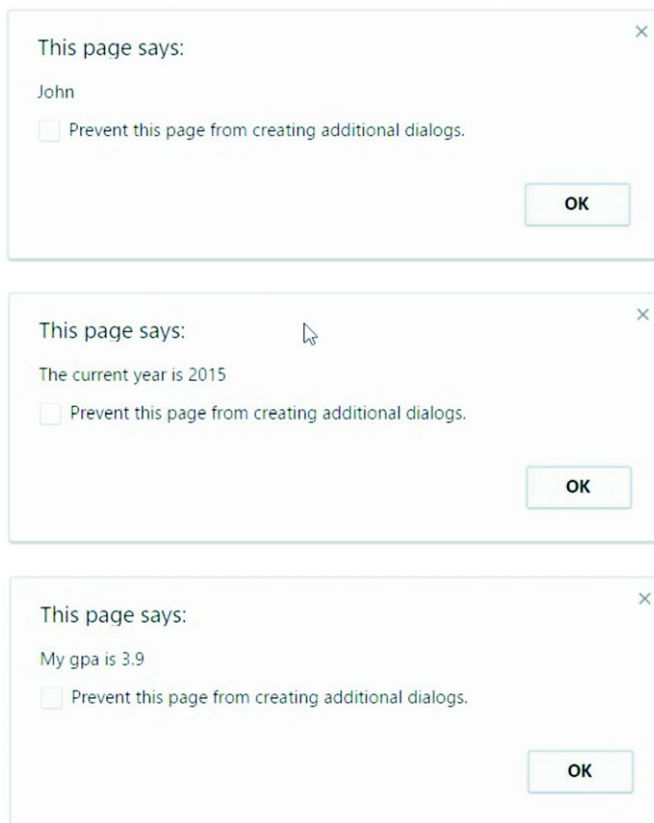


FIGURE 4. Here are the results of these alert statements. The alert function displays a dialog box that must be responded to before the program can proceed. Output from the alert function is disruptive to the end user experience and is used in this case to illustrate the inner workings of the variable assignment statement and is infrequently used in real-life applications.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset= "utf-8" />
    <title>Myfirst HTML5 page</title>
    <script>
      var student_name
      student_name = prompt("Enter student's name")
      alert("Student Name is " + student_name)
    </script>
  </head>
  <body>

    <p>Testing JavaScript</p>

  </body>
</html>
```

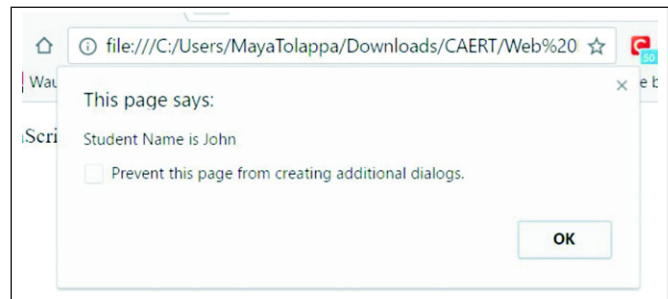
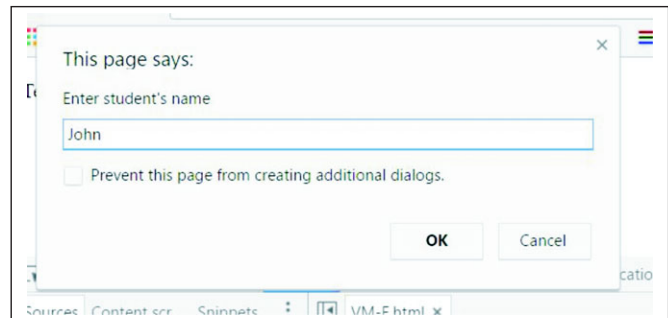


FIGURE 5. This JavaScript code uses the prompt function to obtain a value from the user that it displays via the alert statement. The argument to the prompt function is displayed in the message box as the prompt to the user.

In the second alert function invocation, a string literal “The current year” and a variable `current_year` are concatenated (the operation of joining character strings end-to-end) using the “+” operator. This operation creates a string entity that contains the string literal and the contents of the variable.

The **prompt function** is a feature that populates variables. In previous examples, variables were populated with literals, and variables can be populated using the prompt function. This function asks the user for a value that is placed in a variable.

Referring to FIGURE 5: Note that data provided by the user is always treated as a string value. To use it as a number, it must be converted from a string to a numeric value. The **parseInt** and the **parseFloat functions** are features that accomplish the conversion from a string to a numeric value.

```

<!DOCTYPE html>
<head>
  <meta charset= "utf-8" />
  <title>Myfirst HTML5 page</title>
  <script>
    var stu_hours, stu_gpa
    stu_hours = prompt("Enter hours taken by student")
    stu_hours = parseInt(stu_hours)

    stu_gpa = prompt("Enter student's GPA")
    stu_gpa = parseFloat(stu_gpa)
    alert("Student is taking " + stu_hours +
          " hours.\nStudent's GPA is " + stu_gpa)
  </script>
</head>
<body>

  <p>Testing JavaScript</p>
</body>
</html>

```

This page says:

Enter hours taken by student

☐ Prevent this page from creating additional dialogs.

OK Cancel

This page says:

Enter student's GPA

☐ Prevent this page from creating additional dialogs.

OK Cancel

This page says:

Student is taking 12 hours.
Student's GPA is 3.95

☐ Prevent this page from creating additional dialogs.

OK

FIGURE 6. The user is requested to enter two numeric values. These values are converted into numeric values using the parse functions. The text highlighted in yellow represents an escape sequence. It is a “short-cut” that tells the system to place a new-line character. This action is shown in the display produced by the alert function in the last row of the table where the output is on two lines.

Expressions and Operators

The assignment statements discussed so far consist of a variable name, the assignment operator (=), and the value to be placed in the variable. However, an assignment statement may contain expressions instead of a single value. An **expression** is a set of operands connected by operators. An **operator** is a symbol used to perform a procedure or a task.

JavaScript has operators that work with string values and numeric values. For strings, JavaScript supports the + operator. This serves as a concatenation or string-append operator.

Numeric operands support many operators. These include operators to add, subtract, divide, and multiply.

Modulus operator: JavaScript supports the “%” operator. This is the modulus operator that retrieves the remainder after a division operation.

NOTE: When multiple operators are used in a mathematical expression, they are evaluated in the same order as they are in a mathematical expression.

NOTE: The highlighted assignment statements indicate the expression portion of the statement.

Assignment Statement	Description
<code>var test_weight</code> <code>test_weight = 45.6</code>	A simple expression in which the value of 45.6 is placed in the test_weight variable.
<code>var test_weight</code> <code>test_weight = 45.6 + 10.5</code>	This expression is evaluated in two steps. <ul style="list-style-type: none"> • 10.5 is added to 45.6 to obtain the value of 56.1 • 56.1 is placed in test_weight
<code>var test_weight</code> <code>test_weight = 45.6 - 15</code>	This expression is evaluated in two steps. <ul style="list-style-type: none"> • 15 is subtracted from 45.6 to obtain the value of 30.6 • 30.6 is placed in test_weight
<code>var test_weight</code> <code>test_weight = 45.6 + 5 - 2</code>	This expression is evaluated in multiple steps. <ul style="list-style-type: none"> • 5 is added to 45.6 to obtain the value of 50.6 • 2 is subtracted from this number to yield 48.6 • 48.6 is then placed in test_weight

NOTE: The evaluation of the expression on the left-hand side (LHS) can be a *multistep process*.

FIGURE 7. These are examples of assignment statements in which an expression is used to populate a variable.

Assignment Statement	Value of the final_string variable	Description
<code>var final_string</code> <code>final_string = "Jane"</code>	Jane	Right-hand side value is assigned to variable
<code>var final_string</code> <code>final_string = "Jane" + "Doe"</code>	Jane Doe	The two strings are concatenated together. "Doe" is added to the end of "Jane." Note the blank at the end of "Jane." This ensures there is a blank space between the first and last name.
<code>var final_string, class_name</code> <code>class_name = "WEB140"</code> <code>final_string = "JavaScript"</code> <code> + class_name</code>	JavaScript WEB140	The two operands being concatenated are a string constant (within double quotes) and a variable. As class_name is referred to without quotes, it is assumed to be a variable and JavaScript locates the variable and plugs its current value into the expression. So, the actual concatenation statement is: <code>final_string = "Java Script "</code> <code> + "WEB140"</code>
<code>var string1 = "Jane"</code> <code>string1+= "Doe"</code>	Jane Doe	The += performs a concatenation. This operation can be written as follows: <code>var string1 = "Jane"</code> <code>string1 = string1 + "Doe"</code>

FIGURE 8. These are examples of string operator usage. The + can be used to add two string operands. The += operator serves as an append operator.

Operator	Use Examples	Description
+	<pre>var sc1, sc2, tot_sc sc1=100 sc2=87 tot_sc = sc1 + sc2</pre>	<p>tot_sc has a value of 187.</p> <p>The two Right-Hand Side Operands are specified without any quotes and are therefore assumed to be variables. The current values in the two variables are retrieved and the addition performed. The final value is then placed in the tot_sc variable.</p>
-	<pre>var net net = 50000 - 200</pre>	<p>The Right-Hand Side Expression is evaluated. 200 is subtracted from 50000 and the value 49800 is placed in the net variable</p>
*	<pre>var abc = 12 * 5</pre>	<p>The "*" symbol is used for multiplication since "X" is a valid variable name. 12 is multiplied by 5 and the value 60 is placed in abc.</p>
/	<pre>var avg = 125/5</pre>	<p>The "/" is the divide symbol.</p>
%	<pre>var minutes = 65 var remainder remainder = minutes % 60</pre>	<p>The "%" is the modulus operator. When it is used, it performs a division operation and places the value in the remainder of the operation in the variable in the Left Hand Side</p>
++	<pre>var num_students = 10 num_students++</pre>	<p>This adds 1 to the variable. It is the same as the following statements:</p> <pre>var num_students = 10 num_students = num_students + 1</pre> <p>In the above bold statement, the Right Hand Side is evaluated first. The value in the num_students variable is retrieved and 1 is added to to it. The new value is then placed back in the num_students variable, in effect adding 1 to the variable.</p>
--	<pre>var num_students = 10 num_students--</pre>	<p>Subtracts 1 from the variable. Works the same as the statements shown below:</p> <pre>var num_students = 10 num_students = num_students - 1</pre>
+=	<pre>var num_students = 10 num_students += 5</pre>	<p>The += statement performs two operations. It performs the addition operation followed by the assignment operation. It can also be coded as follows:</p> <pre>var num_students = 10 num_students = num_students + 5</pre>
-= *= /=		<p>All of these are combination operators that perform the specified arithmetic operation, followed by the assignment operation.</p>

FIGURE 9. These are examples of using arithmetic operators. The +, -, and / operators serve the same purpose as they do in "real" math operations. For multiplication, JavaScript uses "*" instead of "X" since "X" is a valid variable name.

Objects Used in JavaScript

As a reminder, the main purpose of JavaScript is to write statements that access the webpage and change its behavior and content. JavaScript uses objects, methods, and properties to accomplish this task. In JavaScript, almost everything is an object. JavaScript refers to elements in a webpage as “objects.” An **object** is a collection of properties; a property is an association between a name (or key) and a value. Objects are standalone entities, with properties and type. For example, the webpage’s body element is accessed via an object called “document.” JavaScript is a case-sensitive language. Therefore, this object cannot be referred to as “Document.”

To perform various actions on the object, developers make use of the object’s properties and methods. For example, the **document object** is an item that has properties that can be used to alter the background color of the HTML document—methods used to display text on the screen.

Object properties are set using assignment statements. **Object methods** are functions, and an object’s method invocation looks like a function invocation.

```
1 <!DOCTYPE html>
2 <head>
3   <meta charset= "utf-8" />
4   <title>My first HTML5 page</title>
5   <script>
6     document.title = "Hello World!"
7     document.write("Hello World")
8   </script>
9 </head>
10 <body>
11
12
13 </body>
14 </html>
```

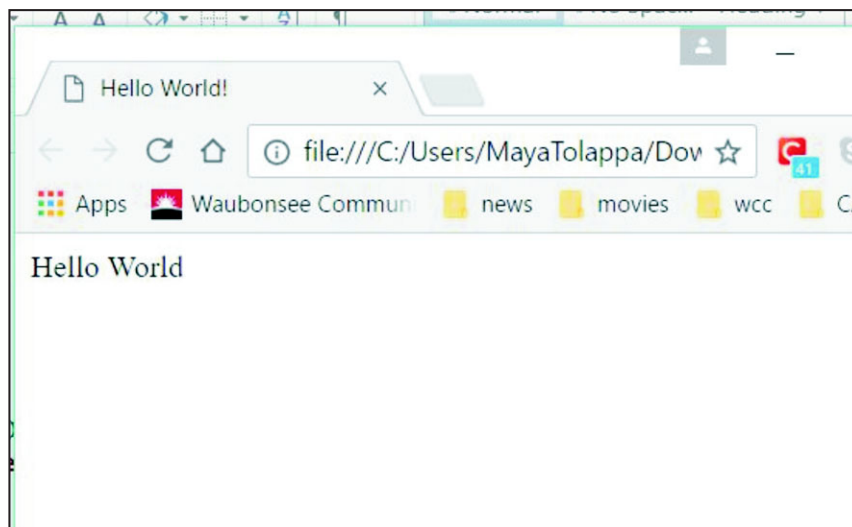


FIGURE 10. The “.” operator is used to reference an object’s methods and properties. In line 4, the title of the page is set to “My first HTML5 page.” In line 6, the title property of the document object is set to “Hello World.” In the browser view shown beneath the code, it is evident that the JavaScript setting overrides the value set in the HTML code. Line 7 uses the “write” method of the document. The *write method* renders text in the webpage.

Debugging JavaScript Code

JavaScript errors can be hard to debug. When the browser's JavaScript interpreter encounters an error, it does not display an error message to avoid disrupting the end user experience. However, errors in the JavaScript code may cause other issues. Therefore, it is essential to learn to debug JavaScript errors.

Debugging in Google Chrome browser: After displaying the page in the Google Chrome browser window, the developer clicks on the “Customize and Control Google Chrome” button on the top left side of the browser window. Then he or she clicks on “More Tools” and selects “Developer Tools.” The browser window changes to “Show Settings.”

```
<!DOCTYPE html>
<head>
  <meta charset= "utf-8" />
  <title>Myfirst HTML5 page</title>
  <script>
    /*examples of assignment
    statements*/

    var myname, current_year, my_gpa//declarations

    myname = "John" //Place "John" in myname
    current_year = 2015
    my_gpa = 3.9

    alert(myname)
    alert("The current year is " + current_year)
    alert("My gpa is " + my_gpa)
  </script>
</head>
<body>

  <p>Testing JavaScript</p>
</body>
</html>
```

FIGURE 11. The highlighted line contains an error. There is an embedded space in the variable name. When this file is opened in Google Chrome, none of the alert statements work. Rather than disrupt the end-user experience by displaying obscure JavaScript errors, the browser software ignores the statements completely.

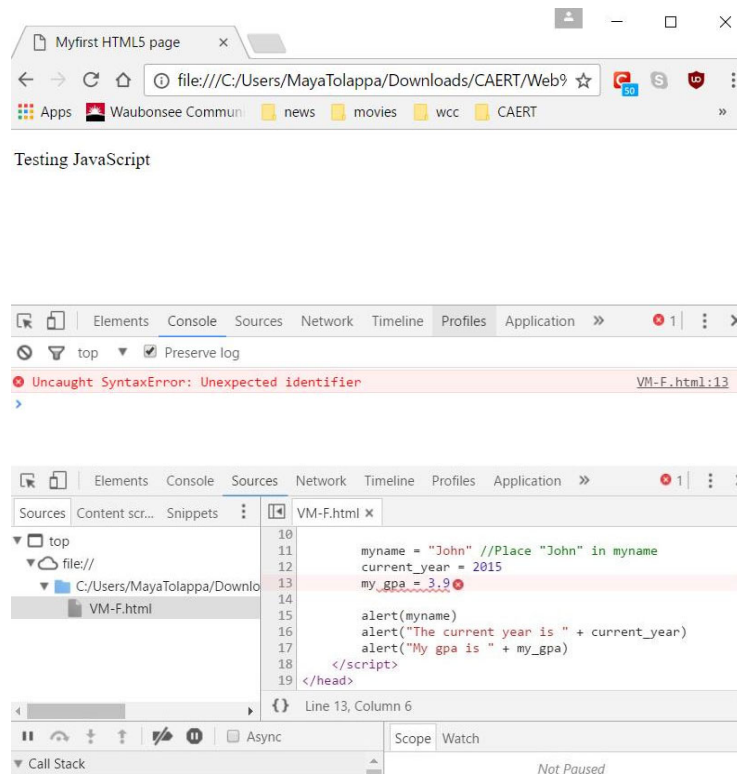


FIGURE 12. This figure shows the developer window with the error code. In the left-hand side pane, the elements tab shows the elements in the page. Clicking on the “Console” tab, JavaScript errors are shown, with the line number that contains the error. Clicking on the line number link takes the developer to the line number with the error.

Debugging CSS in Firefox Browser: To view debugging information, the developer clicks on the “Tools” menu option and selects “Web Developer” and “Inspector.” If the menu bar is not visible, the developer right-clicks on the title bar and selects the “Menu Bar” option.

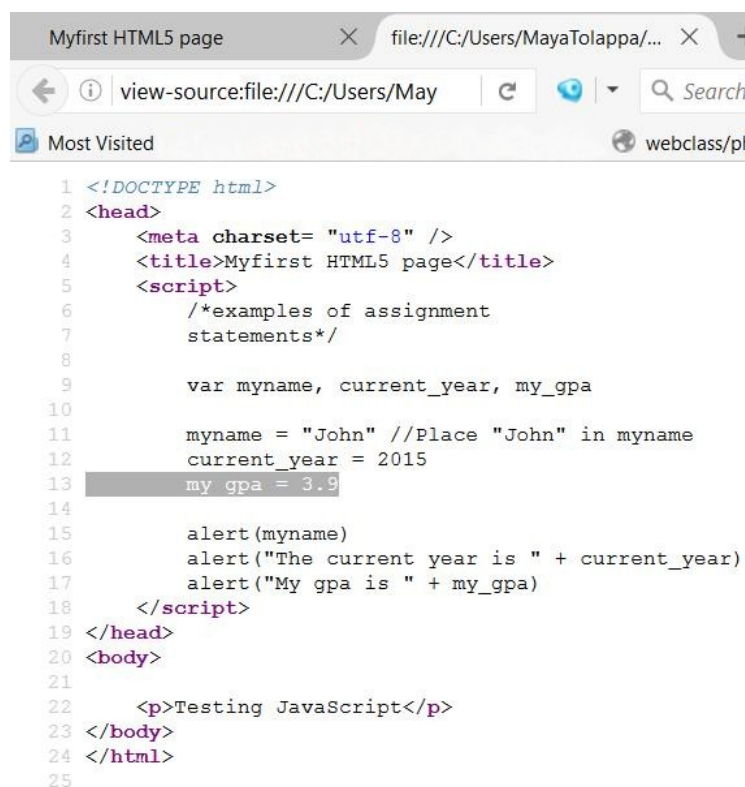
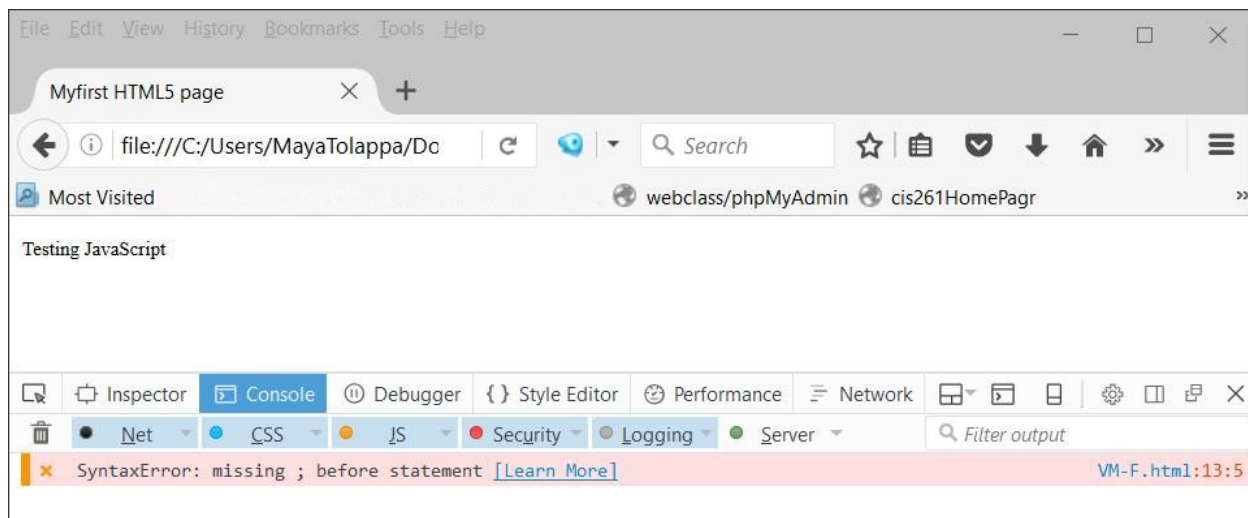


FIGURE 13. Clicking on the “Console” tab causes the line number with the error to be displayed. By clicking on the line number link, a new tab is opened with the code in it. The line in error is now highlighted.

IMPLEMENT LOGIC WITH DECISIONS, LOOPS, AND ARRAYS

Decision Structures

Decisions are features that alter the flow of code execution in JavaScript. Decisions are created using relational operators. **Relational operators** are features that compare the values in two variables. When two operands are compared, the result of the operation is true or false.

Operator	Description	Example
<	Less than	var1 < var2
>	Greater than	var2 > var2
==	Equal to	age1 == 25 If the 2 operands are of different data types, JavaScript will attempt to make then the same data type.
<=	Less than equal to	age <= 20
>=	Greater than equal to	age >= 90
!=	Not equal to	age != 19
===	Equivalent to	Both the operands must be the same data type and also equal. 5 === "5" is false, since they are not both the same data type. 5 == "5" is true, since they are equal
!==	No equivalent to	

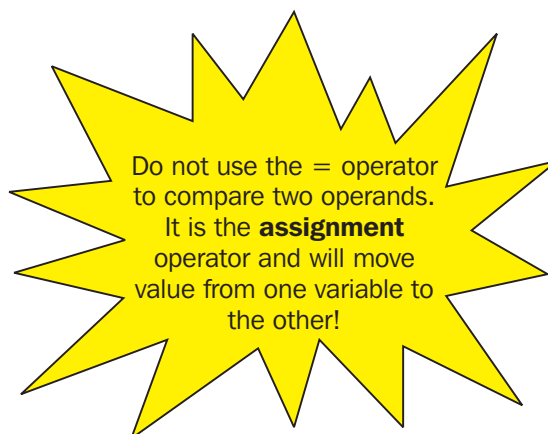


FIGURE 14. Relational operators used in JavaScript compare the values in two variables.

As JavaScript is a loosely typed language, it is important to compare the datatype and the value held in the variables when making a comparison. The “equivalent to” operator must be used instead of the “equal to” operator to check datatype and content. Developers do not use the “=” operator to compare values, as it is the assignment operator and will result in modifying the value in the variable.

Decision Statements Using “If-Else”

Decision structures are implemented in JavaScript using two structures: “if-else” structure and the “switch” structure. **“If-else”** is a decision statement; it is a conditional statement used to perform different actions based on different conditions. When coding the “if” structure, the “if” keyword is used with two operands and a relational operator. If the comparison returns a value of true, statements in the “if” body are implemented. When the comparison returns a false, statements in the else block are executed.

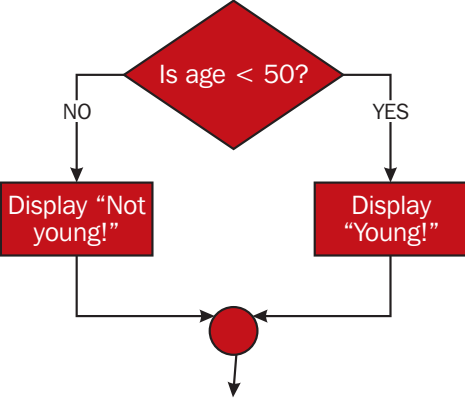
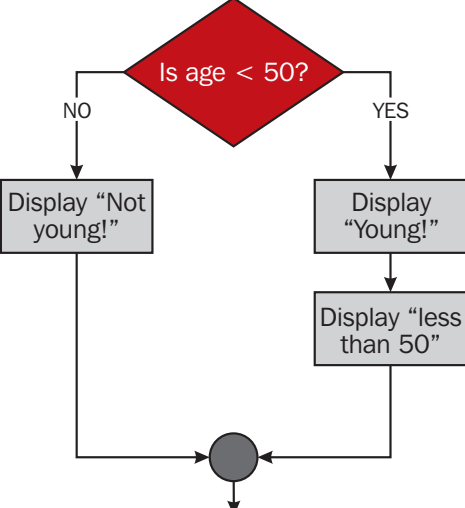
Logic Structure	JavaScript Code
	<pre>var age = 45 if (age < 50) alert("Young!") else alert("Not young!")</pre>
	<pre>var age = 45 if (age < 50) { alert("Young") alert("Less than 50") } else alert("Not young")</pre>

FIGURE 15. These are examples of using the “if” structure. The condition tested by the “if” statement is enclosed by a pair of parentheses. This is a syntactical requirement of the language. Indenting statements within an “if” structure helps make the program easier to read and follow. If multiple statements need to be executed when the condition is true, they are enclosed in a set of { } (braces).

When writing code, it is important to ensure that variables have values before they are used in decision structures. If a variable has no value in it, the condition will not return correct results. To avoid such problems, variables may be compared to the keyword null. In JavaScript, **null** is the intentional absence of any object value and must be defined (otherwise it will be

JavaScript Code	
<pre> 1 <!DOCTYPE html> 2 <head> 3 <meta charset= "utf-8" /> 4 <title>Myfirst HTML5 page</title> 5 <script> 6 var stu_name, stu_gpa 7 8 alert("stu_name contains " + stu_name) 9 10 if (stu_name == null) 11 alert("Nothing in stu_name?") 12 else 13 alert("Student name is " + stu_name) 14 15 stu_gpa = stu_gpa + 0.1 16 alert("student's gpa=" + stu_gpa) 17 </script> 18 </head> 19 <body> 20 21 <p>Testing JavaScript</p> 22 </body> 23 </html> </pre>	<div> <p>This page says:</p> <p>stu_name contains undefined</p> <p><input type="checkbox"/> Prevent this page from creating additional dialogs.</p> </div> <div> <p>This page says:</p> <p>Student name is undefined</p> <p><input type="checkbox"/> Prevent this page from creating additional dialogs.</p> </div> <div> <p>This page says:</p> <p>student's gpa=NaN</p> <p><input type="checkbox"/> Prevent this page from creating additional dialogs.</p> </div>

FIGURE 16. These are examples of JavaScript code remonstrating (arguing) the use of the keywords “null and NaN.”

undefined). NOTE: “Null” is for objects, and “undefined” is for variables, properties, and methods.

JavaScript makes use of another keyword called NaN. **NaN** is an abbreviation for the term “Not a Number” and is a value placed in a numeric variable if an attempt is made to place a non-numeric value in a numeric variable.

Referring to FIGURE 16:

The `stu_name` variable is declared in line 6 but no value is assigned to it. Line 8 attempts to display the `stu_name` variable. Chrome displays a message saying that the variable contains “undefined.”

Line 10 checks whether the value in the variable is equal to the keyword “null.” This returns a “true,” as the variable has no value assigned to it and the second display is shown by Chrome.

Consider the `stu_gpa` variable. No value is assigned to it. In line 15, the value 0.1 is added to it. Since it contains no value, adding to it does not create a numeric value, and Chrome displays it as having a NaN value.

Variables must be numeric to participate in an arithmetic operation. However, variables cannot be compared to NaN. Instead, JavaScript provides a function called `isNaN`. This function takes in a variable as an argument and returns a true or false depending upon the contents of the variable.

JavaScript supports Boolean variables that can hold values of true or false. [NOTE: Both true and false are reserved words in JavaScript.]



FIGURE 17. This example shows the remonstrating use of the isNaN function. When a user enters a value via the prompt function, the value is converted into a number using the parseFloat or the parstInt function. However, if the user enters a non-numeric value, which cannot be converted by the parseFloat or parseInt functions, NaN is placed in the variable that was to be populated via the prompt statement.

Decisions Using Switch Statements

A **switch statement** is a process to simplify large if-else statement structures. When an if-else statement is coded to check one of many different values in a single variable, it is a good candidate for conversion to a switch statement.

"If-Else" Code	"Switch " Code
<pre> 1 var inptest 2 3 inptest = prompt("Enter number of tests taken") 4 inptest = parseInt(inptest) 5 if (inptest == 4) 6 alert("You attended all 4 tests, you get an A!") 7 else 8 if (inptest == 3) 9 alert("You took 3 tests - B") 10 else 11 if (inptest == 2) 12 alert("You took 2 tests - C!") 13 else 14 if (inptest == 1) 15 alert("You took 1 test - D!") 16 else 17 alert("You got an F!") </pre>	<pre> 1 var inptest 2 3 inptest = prompt("Enter number of tests taken") 4 inptest = parseInt(inptest) 5 switch (inptest) 6 { 7 case 4: 8 alert("You took 4 tests- A") 9 break 10 case 3: 11 alert("You took 3 tests - B") 12 break 13 case 2: 14 alert("You took 2 tests - C!") 15 break 16 case 1: 17 alert("You took 1 test - D!") 18 break 19 default : 20 alert("You got an F!") 21 } </pre>

FIGURE 18. This figure shows an if-else structure and the switch statement that implements the same logic. Note the use of the break statement. A break statement is used to "jump out" of a switch statement and out of a loop. A switch construct is not structured in the same way as an "if" statement. In a nested "if" statement, once a match has been made, all other statement in the "if" statement are ignored. This is not true for the switch statement. When the value in the switch variable matches one of the case values, control passes into that case statement. All statements from there on are executed, even other case-related statements. The break statement serves to "break out" of the switch statement.

With the switch statement, developers ensure that in the case statements no relational operators are specified, such as the \geq , \leq . Only strict equality is allowed.

Logical Operators

Logical operators are tools used to test for *true* or *false*. When multiple comparisons need to be performed in an “if” statement, logical operators are used. The “&&” operator is used to denote the “AND” logical operator, and the “||” operator is used to denote the “OR” logical operator. The “!” operator is used for the “NOT” operation.

Loop Structures

Loop structures are a means of replicating a set of code repeatedly in JavaScript. Broadly speaking, there are two types of loops: pre-test loops and post-test loops.

In pretest loops, the loop condition is tested before entering the loop body. Consequently, if the loop condition is not met, the loop body statements may not be executed at all. The minimum number of times a pretest loop’s body statements are executed is 0.

In post-test loops, the loop condition is tested after entering the loop body. Consequently, if the loop condition is not met, the loop body statements will execute at least once. The minimum number of times a pretest loop’s body statements are executed is 1. [NOTE: This lesson will confine discussions to pre-test loops as all post-test loops can be written as pre-test loops, but all post-test loops cannot be converted into pre-test loops.]

We will confine discussions in this lesson to pre-test loops since all post-test loops can be written as pre-test loops, but all post-test loops cannot be converted into pre-test loops.

JavaScript Code	
1	<code>if (attendance >= 90 && totalPoints >=90)</code>
2	<code>document.write("You got an A!***")</code>
3	
4	<code>if (attendance >= 90 totalPoints == 100)</code>
5	<code>document.write("OK")</code>

FIGURE 19. These examples demonstrate the use of logical operators. In the first example, the message is printed out only if the variable *attendance* is 90 or more and the *totalPoints* variable is 90 or more. Both the conditions need to be true for the complete “if statement” to be considered true. In the second example, the message “OK” is printed if *attendance* is greater than or equal to 90 or *totalgrade* is equal to 100.

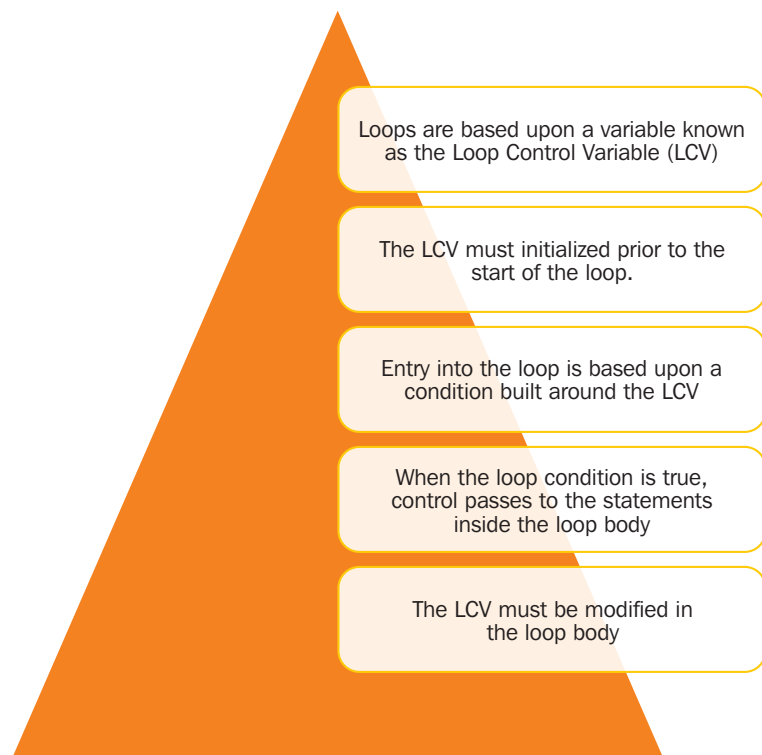


FIGURE 20. All pre-test loop structures consist of five elements. These five elements are displayed here. When developers code a pre-test loop, they should be able to identify these five loop elements within your code.

The **while loop structure** is a feature that implements a pre-test loop in JavaScript.

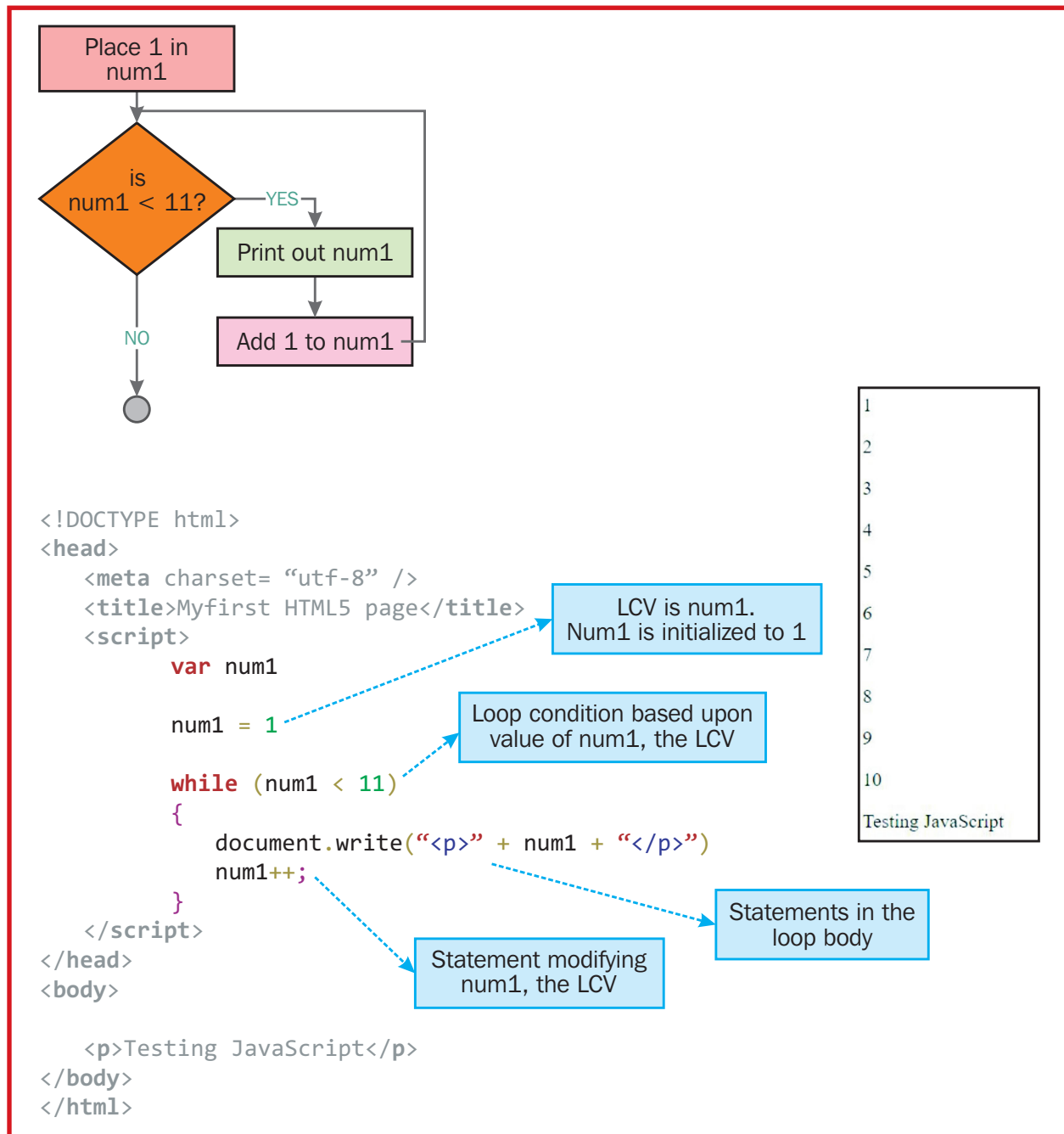


FIGURE 21. This is an example of a while loop that prints out numbers 1 to 10 on a webpage. The flowchart shows that after the last statement in the loop body have been executed, control reverts back to the loop condition statement.

Data validation can be performed using while loops. The user is requested to enter a value. As long as the user-entered value is deemed to be invalid, code in the data validation loop iterates. The loop code consists of statements that display error messages and request the user to enter a value again.

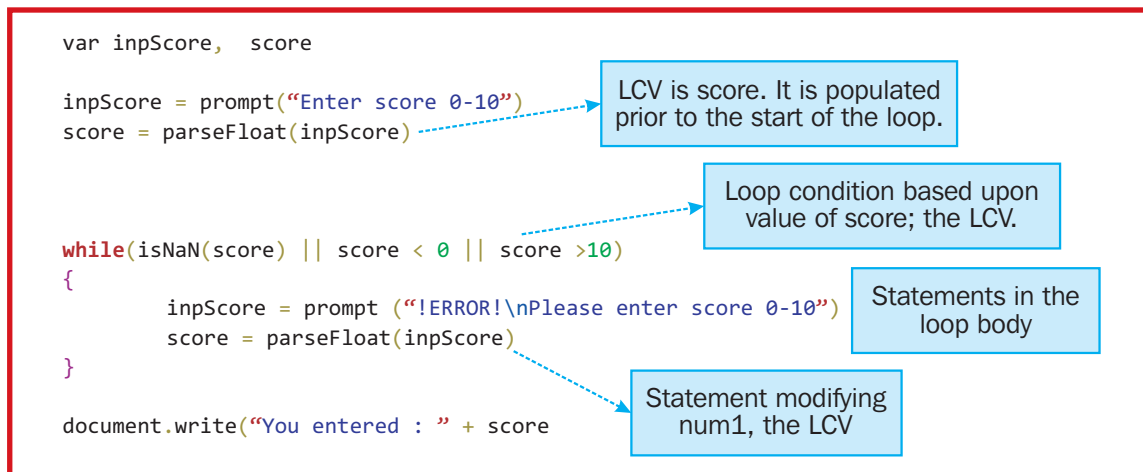


FIGURE 22. In this code the loop control variable—"score"—is initialized prior to the loop via prompt statements. If the data is invalid, control passes to the loop. Data is considered invalid if the `parseFloat` function fails and places NaN in the score variable, or if the conversion is successful, but the numeric value is not in the correct range. If the user enters valid data the first time they are prompted, the loop statements will never be executed.

If the score variable contains invalid data, the loop is entered. In the loop, an error message is displayed, and the user is once again requested to enter data. A **for loop structure** is a pre-test loop construction used when the number of iterations is known ahead of time. The "for" loop structure contains all the elements of a pre-test loop. The loop control variable (LCV) is initialized, the condition is specified, and the increment value is specified in the "for" header followed by the loop body.

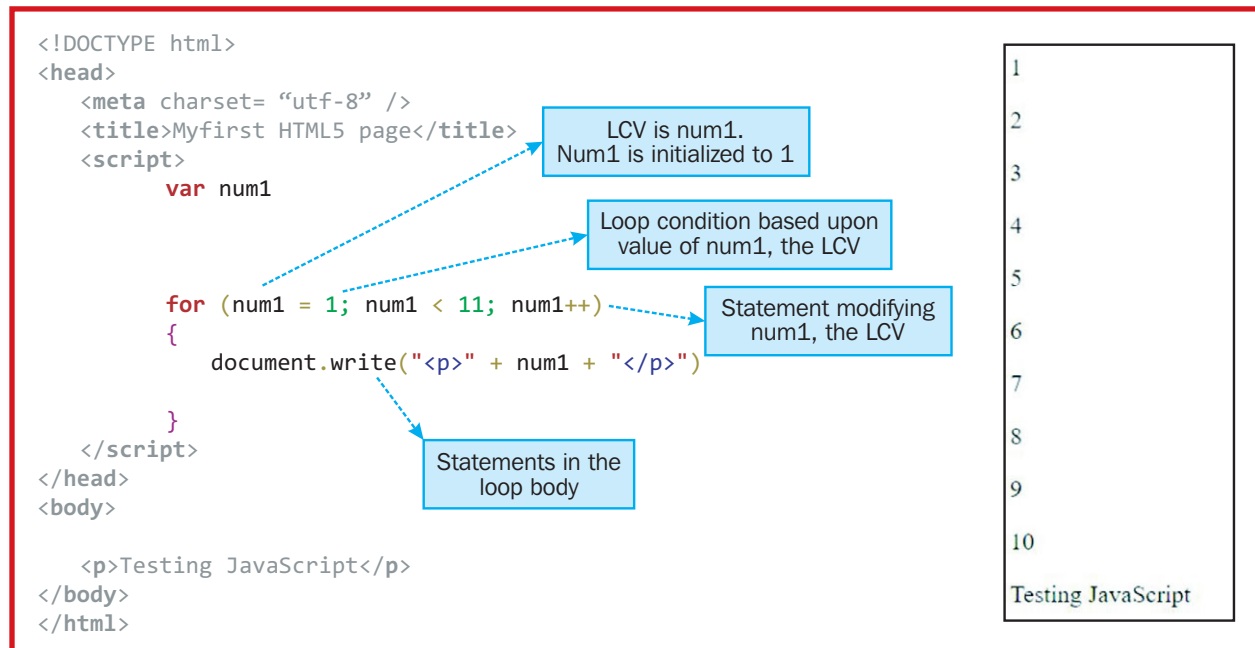


FIGURE 23. Here is a "for" loop that prints out numbers 1 to 10. Execution proceeds in the "for" loop in a fashion that is similar to execution of a "while" loop. The loop control variable (LCV), num1, is initialized to 1. Since the LCV is less than 11, control passes to body of the loop. The loop body statement, the "document.write" statement is executed. The LCV, num1, is incremented by the value specified in the increment clause on the "for" header and num1 is incremented by 1. Control passes back to the "for" header, the condition is checked again to decide whether to pass control into the loop. [NOTE: The point is that the loop control variable is incremented at the foot of the loop.]

Arrays

Arrays are a mechanism to hold multiple values in JavaScript. Each individual value in an array is referred to as an array element. All the elements in the array share the same name. They are differentiated by their positions within the array. This is analogous to the pages in a book. To read a specific page, the page number is required. An array is like a book, and each individual element in the array is like a page in the book. The unique identifier for each element in the array is its **subscript**—similar to page number in a book.

An array is a JavaScript object called “Array.” Unlike the document object, an array is not automatically available when a webpage is opened in the browser. It must be created before it can be used.

JavaScript Code	
<pre>1 <!DOCTYPE html> 2 <head> 3 <meta charset= "utf-8" /> 4 <title>Myfirst HTML5 page</title> 5 <script> 6 var scores = new Array(3) 7 var stu_ids = new Array (10, 23, 44, 76, 1) 8 var st_names = ["Jane", "John", "James", "Lily"] 9 10 document.write("<p>Scores : " + scores + "</p>") 11 document.write("<p>stu_ids : " + stu_ids + "</p>") 12 document.write("<p>st_names : " + st_names + "</p>") 13 </script> 14 </head> 15 <body> 16 17 </body> 18 </html></pre>	<pre>Scores :., stu_ids :10,23,44,76,1 st_names :Jane,John,James,Lily</pre>

FIGURE 24. Line 6 creates an array called “scores” with three elements. Note the use of the “new” keyword, followed by the name of the object, “Array.” When JavaScript executes the above statement, an array with three elements is set up. All three elements have the name “scores,” but they have different subscripts. When an array is set up, elements are not automatically initialized. They are set to “undefined” value. Line 7 shows an array initialized with a list of values; it is a five-element array with the values specified in the list. Line 8 creates a four-element array without the use of the “new” or “Array” keywords. Instead the “[]” symbols are used to set up the array elements.

Also shown in Figure 24 are the statements to display the contents of the three arrays. The scores array contains no values, and only the “,” symbols are displayed to indicate that there are three uninitialized elements in the array.

Since an array contains multiple elements within, each element is treated as an individual variable and is processed accordingly. An individual element is retrieved from an array using its position within the array. This position number is called a “subscript.” In JavaScript, the subscript value for the first element in the array is 0, not 1. The valid subscript values for a three-element array are 0, 1, and 2, with the last valid subscript for a three-element array being 2. However, arrays in JavaScript are open ended. Even though the above array is set up to have three elements, its size can be dynamically increased. Arrays have a property called “length”

that returns the number of elements in the array. In addition, arrays are usually processed using “for” loops. The size of the array is used to set up the number of iterations that the “for” loop requires.

JavaScript Code	
<pre>1 <!DOCTYPE html> 2 <head> 3 <meta charset= "utf-8" /> 4 <title>JavaScript</title> 5 <script> 6 var scores = new Array(3) 7 var st_names = ["Jane", "John", "James", "Lily"] 8 9 scores[0] = 100 //first element 10 scores[1] = 50 //second element 11 scores[2] = 78 //third element 12 13 scores[3] = 200 //append 4th element to the array 14 15 document.write("<p>Scores :"+ scores + "</p>") 16 17 for(i = 0; i < st_names.length; i++) 18 document.write("<p>" + st_names[i] + "</p>") 19 </script> 20 </head> 21 <body> 22 23 </body> 24 </html></pre>	<div>Scores :100,50,78,200</div> <div>Jane</div> <div>John</div> <div>James</div> <div>Lily</div>

FIGURE 25. Here is code that accesses individual elements in an array. Consider lines 9 to 11. They place values in the three elements in the array. In line 13, the subscript “3” refers to the fourth element. This statement automatically increases the array to a four-element array, and the value 200 is placed in the fourth element. Lines 17 to 18 set up a “for” loop where the Loop Control Variable “i” is initialized to 0 and can take up values up to 1 less than the length property of the st_names array. This means that for a four-element array, “i” takes up value 0, 1, 2, and 3. Elements from the array are accessed by subscripts in line 18 and displayed out in the document.

JavaScript arrays support many useful methods. As seen in Figure 26, the “indexOf” method takes an argument and attempts to locate it inside the array. If it is found, the subscript of the element in the array is returned. In this example, the indexOf method returns a value of 0 for the argument “Zak” since it is the first element in the array. When the indexOf method is used with “Jane” as the argument, the value is returned is -1, which is an invalid subscript, indicating that the element is not present in the array. The push method adds an element to the end of the array and serves as an append method. The pop method removes the first element in the array. The sort method sorts the elements of the array in ascending order.

These methods are based upon the following array: `var st_names = ["Zak", "Joe", "Tim"]`

Method	Description	Usage	Output
indexOf	Returns the position of an element in the array. If the element is not in the array, it returns a -1.	<pre>var indexof_zak, indexof_jane indexof_zak = st_names.indexOf("Zak") document.write("<p>Position of Zak in array = " + indexof_zak + "</p>") indexof_jane = st_names.indexOf("Jane") document.write("<p>Position of Jane in array = " + indexof_jane + "</p>")</pre>	Position of Zak in array = 0 Position of Jane in array = -1
push	Adds one or more elements to the end of an array and returns the length of the array.	<pre>st_names.push("Mac") document.write("<p>" + st_names + "</p>")</pre>	Zak,Joe,Tim,Mac
pop	Removes the last element from an array and returns that element.	<pre>st_names = ["Zak", "Joe", "Tim"] st_names.pop() document.write("<p>" + st_names + "</p>")</pre>	Zak,Joe
sort	Transposes the elements of an array: the first array element becomes the last and the last becomes the first.	<pre>st_names = ["Zak", "Joe", "Tim"] st_names.sort()</pre>	Joe,Tim,Zak

FIGURE 26. In this example, the “indexOf” method takes an argument and attempts to locate it inside the array. If it is found, the subscript of the element in the array is returned. In this example, the indexOf method returns a value of 0 for the argument “Zak” since it is the first element in the array. [NOTE: In the examples that demonstrate the pop and sort methods, the array contents are reinitialized, so that the operations work on the original contents of the array.]

USE EVENT HANDLERS IN JAVASCRIPT

Function Creation: In JavaScript, events are handled through coding functions. Therefore, developers need to understand the process of function creation before working with event handlers.

Defining and Invoking a Function

A **function** is a mechanism to “bundle” a set of code and use it multiple times. JavaScript contains many important functions, such as the alert and prompt functions. Additionally, developers can create their own function definitions. Developer-created functions must be defined before they can be invoked. A function definition is composed of the following elements.

Function Header

A **function header** is the first line in a complete function definition and begins with the reserved word “function.” It is composed of sections:

- ◆ A function must have a unique name. Rules for setting up a function name are the same as those for creating a variable name. The function name is preceded by the reserved word “function.”

- ◆ A **function parameter list** is a register that contains the number and data type of parameters required by the function.

Function Body

A **function body** is a statement that executes when the function is invoked. Statements in the function body are placed within { } symbols. By convention, statements within { } are indented.

- ◆ Functions may or may not return a value. If a value is to be returned by the function, the “return” keyword is placed in the function body.
- ◆ Function definitions are typically placed in the head tag. It is important to note that code inside a function is never executed automatically. A function can be invoked after it has been defined. If a function is defined and never invoked, it is never executed.

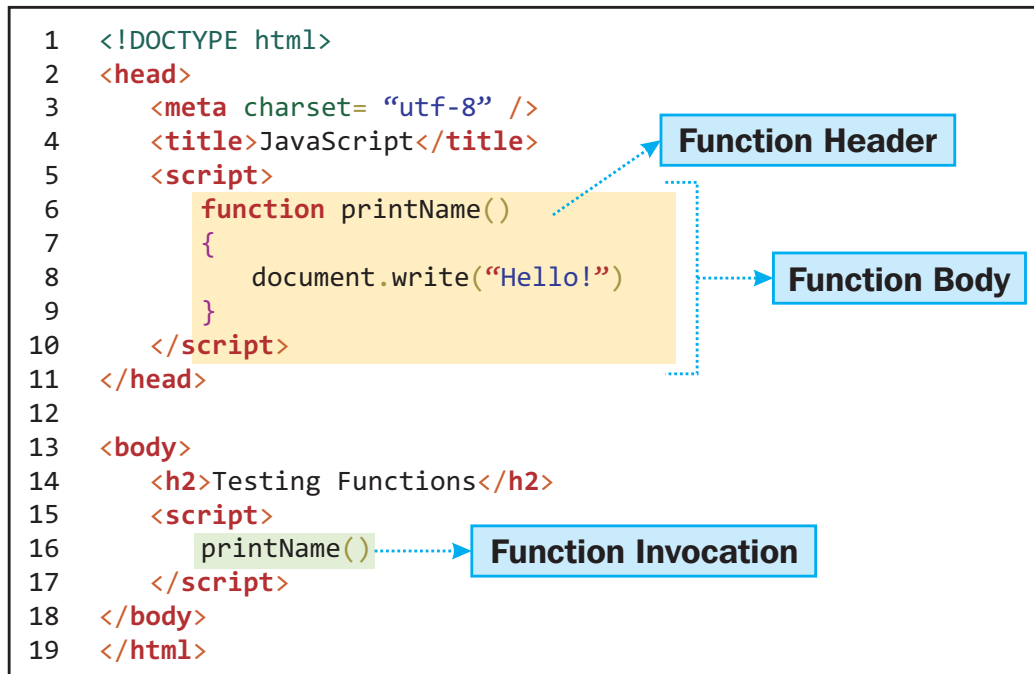


FIGURE 27. Lines 6 to 8 define the function in the head tag of the document, with line 6 serving as the function header. Function definitions begin with the reserved word “function.” Following the word “function,” there is a blank, followed by the name of the function. In this example, the name of the function is “printName.” Following the name of the function is a pair of parentheses. In this example, there is nothing between the parentheses, indicating that the function takes in no parameters. The code within the function is enclosed within a set of braces { }. Statements inside the braces represent the body of the function. Here, the function writes out text to the document object. Here, the function writes out text to the document object. Line 16 invokes this function. To invoke a function, the function name is written out, along with an empty pair of parentheses after it. When a function is invoked, JavaScript looks in all the JavaScript code in the document for a function with that name. When the function is located, control is transferred to the function and all the code in the function is executed. When the end brace “}” in the function is encountered, control is passed back to the statement that invoked it.

Functions that Take in Parameters

Some functions, such as the JavaScript “alert” function, take in parameters and perform some action with it. Developers, too, can write functions that take in a parameter. When a function has been written to take in a parameter, it must be invoked with an argument. When multiple parameters are specified in a function’s header, it must be invoked with the same number of arguments.

Functions may take in multiple parameters. Suppose that a function takes in two parameters. In the function header’s parameter list, commas separate the parameters. When multiple arguments are specified in a function invocation, commas too, separate them. The order in which they are specified in the invocation is important. By position, the arguments in the invocation statement map to the parameters in the function header.

```
1 <!DOCTYPE html>
2 <head>
3   <meta charset= "utf-8" />
4   <title>JavaScript</title>
5   <script>
6     function printName(username)
7     {
8       document.write("Hello!" + username)
9     }
10  </script>
11 </head>
12
13 <body>
14   <h2>Testing Functions</h2>
15   <script>
16     printName("Jonathan")
17   </script>
18 </body>
19 </html>
```

FIGURE 28. Lines 6 to 8 define the function. The main difference between the previous function and this one is that in this function, after the name of the function, `printName`, there is a parameter list with a parameter name specified. The name of the parameter is “username.” Within the code for the function, this parameter is written out. Line 16 invokes this function, and in the invocation, after specifying the name of the function, a string literal “Jonathan” is coded within parentheses. This is the argument to the function invocation. When the function is invoked, the argument value is mapped to the parameter in the function. So, “Jonathan” maps to `username` in the function. When the function executes, “Jonathan” is substituted for `username`.

```
1 <!DOCTYPE html>
2 <head>
3   <meta charset= "utf-8" />
4   <title>JavaScript</title>
5   <script>
6     function printName(firstname, lastname)
7     { document.write("Hello" + firstname +
8       " " + lastname)
9     }
10  </script>
11 </head>
12
13 <body>
14   <h2>Testing Functions</h2>
15   <script>
16     printName("Jane", "Doe")
17   </script>
18 </body>
19 </html>
```

FIGURE 29. The first argument “Jane” maps to the `firstname` parameter. The second argument, “Doe” maps to the second parameter in the function header’s parameter list, `lastname`. These parameters are used in the function body and displayed out.

Functions that Return Values

Local variables are variables defined inside a function and may only be used inside the function. In a function that takes in parameters and performs some sort of calculation, the function can return the final answer back to the originating statement via the return keyword. Functions that return values may participate in an assignment statement's right-hand side and are treated as an operand.

```
1  <!DOCTYPE html>
2  <head>
3    <meta charset= "utf-8" />
4    <title>JavaScript</title>
5    <script>
6      function calc_average(num1, num2)
7      {
8        var num_average
9        num_average = (num1 + num2 ) / 2
10       return num_average
11     }
12   </script>
13 </head>
14
15 <body>
16   <h2>Testing Functions</h2>
17   <script>
18     stu_average = calc_average(10, 12)
19     document.write("Average = " + stu_average)
20   </script>
21 </body>
22 </html>
```

FIGURE 30. Lines 6 to 11 contain the function definition for the `calc_average` function. The function takes in two parameters. In the function, a variable called `num_average` is declared and the average of these two parameter variables is assigned to it. The `num_average` variable is local to the `calc_average` function and may not be used outside it. Line 10 contains the reserved word “return” followed by the name of the variable that is to be returned from the function. In this case, the `num_average` variable is returned from the function. Line 18 invokes the function. However, this invocation looks different from the previous invocations. This invocation looks like an assignment statement. A variable, `stu_average`, appears on the left-hand side of the assignment statement and the function invocation is on the right hand side of the assignment statement. The value it returned by the function is placed in the variable in the left hand side of the assignment statement.

Event Handlers

The power of JavaScript lies in its ability to handle events in an HTML document. Code to handle events is placed within the tag definition as an attribute.

Event	Attribute	Description
Load	onload	It occurs when the HTML document is loaded.
Click	onclick	It occurs when the user clicks on something on the page. This is typically coded for images, buttons, and link tags.
MouseOver	onmouseover	It occurs when the mouse moves over an element. This event is handled for images and links.
MouseOut	onmouseout	It occurs after the MouseOver event has occurred and the mouse moves away from the element it was positioned over.
Focus	onfocus	It occurs when an element in a document gets focus.
Blur	onblur	It occurs after the onfocus event has concluded and the element loses focus.
Submit	onsubmit	It occurs when a form is sent for submission. In this element, checks are conducted to see if the form elements are valid prior to being sent to a program on a server.
Reset	onreset	When a form is reset, this event is triggered.
Data in control does not conform to validity properties.	oninvalid	It occurs when data is entered in a form control/element that does not meet validity constraints. For example, a required field is left empty and the “submit” button is clicked.

FIGURE 31. Events, along with the attribute that handles the event, are displayed. For example, the onload attribute of a <body> element can be coded to point to a JavaScript function.

To handle an event, a function definition is coded, usually within the <head> element. Next, the name of the function is specified in an element’s open tag as the value to an attribute, where the attribute name is the event to be handled.

While CSS3 styles can be used to provide interactivity for elements using the hover pseudo-class, JavaScript provides a wider range of actions. In particular, JavaScript provides additional capabilities for the <a> anchor element. It provides the capability for handling the click event.

Normally, when the contents of an <a> tag are clicked, the page navigates to another page. By specifying an event handler for the onclick attribute, this default action can be changed. The function that handles this event must return the Boolean

```

1  <!DOCTYPE html>
2  <head>
3      <meta charset= "utf-8" />
4      <title>JavaScript</title>
5      <script>
6          function load_event_handler()
7          {
8              alert("Welcome to my site")
9          }
10     </script>
11 </head>
12
13 <body onload = "load_event_handler()">
14     <h2>Testing Events</h2>
15
16 </body>
17
18 </html>

```

FIGURE 32. Lines 6 to 9 define a function called load_event_handler that takes in no parameters. Line 13 uses this function as the value for the body elements’ onLoad attribute. The “()” symbols are included to indicate that the attribute value is a function. Since the name of the function is an attribute, it is surrounded by double quotes. When the load event is detected, the load_event_handler function is executed.

JavaScript Code

```

1 <!DOCTYPE html>
2 <head>
3   <meta charset= "utf-8" />
4   <title>JavaScript</title>
5   <script>
6     function click_event_handler()
7     { var response
8       response = confirm("Do you wish to leave?")
9       return response
10    }
11  </script>
12 </head>
13
14 <body>
15   <h2>Testing Events</h2>
16   <a href="http://www.google.com"
17     onClick = "return click_event_handler()">
18     click here to move to another page
19   </a>
20 </body>
21 </html>

```

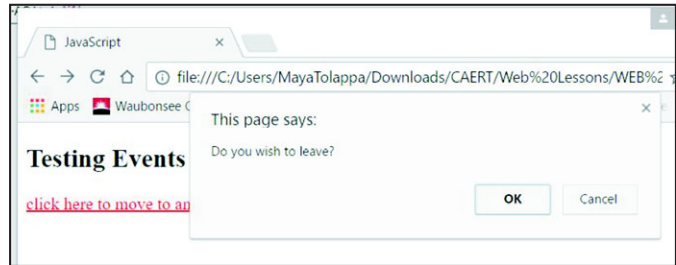


FIGURE 33. Here is an example of the onclick event handler for an anchor element. Line 8 makes use of the “confirm” function. This function displays a message to the developer with the buttons “OK” and “Cancel.” Clicking on these buttons returns a true or a false value, respectively. The value returned by the function is the value returned from the confirm function. If the user clicks on the “OK” button, the confirm function returns a true. This value is stored in the response variable that is returned from the function. On the other hand, if the user clicks on the “Cancel” button, a value of false is returned from the function. When the click event handler of an <a> element returns false, navigation to the target URL is cancelled.

value. If the click event returns true, the window navigates to the anchor destination. If the click event returns false, the window does not move to the anchor destination and remains at the current page.

Maintaining JavaScript Code in a Separate File

In a website, it is possible that a specific JavaScript function is used in multiple pages. In such cases, these JavaScript functions are placed in a separate file. Webpages that need to execute functions in this file can link to it. This is similar to the maintenance of website styles in a CSS file.

By convention, files that contain only JavaScript code are maintained in files with the “.js” extension. By using the “src” attribute of the <script> element, webpages are able to link to the files.

JavaScript File Called site_scripts.js

```

1 function click_event_handler()
2 { var response
3   response = confirm("Do you wish to leave?")
4   return response
5 }

```

HTML File that Uses site_scripts.js

```

1 <!DOCTYPE html>
2 <head>
3   <meta charset= "utf-8" />
4   <title>JavaScript</title>
5   <script src="site_scripts.js">
6
7   </script>
8 </head>
9
10 <body>
11   <h2>Testing Events</h2>
12   <a href="http://www.google.com"
13     onClick = "return click_event_handler()">
14     click here to move to another page
15   </a>
16 </body>
17 </html>

```

FIGURE 34. Here is a file that contains a JavaScript function. Line 5 in the HTML file points to the .js file via the “src” attribute. No path information is shown in the “src” attribute because the HTML file and the JavaScript file are in the same folder.

DETECT JAVASCRIPT WITHIN CODE

HTML Forms

Forms are created in a webpage using the `<form>` element. A form may contain textboxes, checkboxes, radio buttons, and other form elements or controls. Controls must have a value specified for the name attribute. Mutually exclusive radio buttons share the same value for the name attribute. Some of the controls have “required” placed as an attribute. In an HTML5 document, the submit button will not be processed unless these controls have content in them. This validation is part of HTML5, and no JavaScript code is needed to perform this validation.

Control	HTML Code
Text Box	<code><input type="text" name="txtname" value="" required maxlength = "15" ></code>
Number within a Range	<code><input type="text" name="txtpnts" value="" required maxlength = "15" ></code>
Button	<code><input type="submit" value="Send data to host"></code>
Option Buttons or Radio Buttons	<code><input type="radio" name="gender" value="male" required>Male <input type="radio" name="gender" value="female">Female</code>
List Box (with the items AOS, CIS, WEB, and ACC)	<code><select name="Major" required> <option value="" >-Select a major-</option> <option value="aos">AOS</option> <option value="cis">CIS</option> <option value="web">WEB</option> <option value="acc">ACC</option> </select></code>

FIGURE 35. Various controls that may be placed in a form are displayed here. Note that for the `<select>` element, the first `<option>` element does not have a value attribute setting. If the submit button is clicked without any selection made in the drop-down list box, the first element is selected by default. Since its value attribute is empty, the required setting is violated, and HTML5 will flag an error.

The `<form>` element contains an action attribute that specifies the name of the webserver program to which the form data should be sent. The `<form>` element supports an “onsubmit” event. Code can be placed in this event to provide logic that needs to execute before form data submission.

To check form validation code, developers require a webserver program that can process data sent into it provided all validation checks have been met. The URL <http://posttestserver.com/post.php> may be used as a “dummy” webserver program for testing purposes. This is a server program online. When form submission is successful, this site displays back the form data sent in.

All HTML controls in the file should have a name attribute and an id attribute. The server program uses the name “attribute” to identify the input controls, and the id attribute is used by JavaScript to address individual elements in a webpage. While the values in these two attributes do not have to be the same, it is common practice to place the same value in both of these attributes for each element in a webpage.

```

1  <!DOCTYPE html>
2  <head>
3      <meta charset= "utf-8" />
4      <title>JavaScript</title>
5  </head>
6
7  <body>
8      <form name="frm_test" action="http://posttestserver.com/post.php"
9      method="post"    >
10
11      <h3>Let us try out some HTML controls</h3>
12
13      <p>Here is a text box.</p>
14      <p>Enter name of course: <input type="text" name="txtname"
15      id = "txtname" value= "" required maxlength = "15" </p>
16
17      <p>Here are a set of option buttons:</p>
18      <p><input type="radio" name="gender" id = "gender"
19      value="m" required
20      oninvalid = "disp_gender_error()"
21      onclick = "clear_validity()">Male
22
23      <input type="radio" name="gender"
24      value="f" onclick = "clear_validity()">Female
25      </p>
26
27      <p>Select one of the following majors:</p>
28      <p><select name="Major" id = "Major" required
29      oninvalid = "disp_major_error()"
30      onclick = "setCustomValidity('')">
31          <option value= "" >-Select a major-</option>
32          <option value = "aos">AOS</option>
33          <option value = "cis">CIS</option>
34          <option value = "web">WEB</option>
35          <option value = "acc">ACC</option>
36      </select>
37      </p>
38      <input type="submit" value="Send data to host">
39
40  </form>
41 </body>
42 </html>

```

course:

Please fill out this field.

☒ Male ☐ Female

Please select one of these options.

-Select a major- ▼

Please select an item in the list.

FIGURE 36. Here is HTML code that displays a form. Line 9 indicates that form data is to be sent to <http://posttestserver.com/post.php>. Since this is an HTML5 file, form validation (such as absence of a required field), the HTML5 interpreter in the browser automatically performs this action. Error messages produced by HTML5 are shown below the code.

Error messages produced by HTML5 form validation logic and are generic in nature. To create custom error messages, JavaScript code needs to be written for the “oninvalid” event for each of the form controls. The function specified in the oninvalid attribute is the event handler function automatically invoked when HTML5 detects that the control is in an invalid state.

```

1  <!DOCTYPE html>
2  <head>
3    <meta charset= "utf-8" />
4    <title>JavaScript</title>
5    <script src = "site_scripts.js"></script>
6  </head>
7
8  <body>
9    <form name="frm_test" action="http://posttestserver.com/post.php"
10     method="post">
11
12     <h3>Let us try out some HTML controls</h3>
13
14
15     <p>Here is a text box.</p>
16     <p>Enter name of course: <input type="text" name="txtname"
17       id = "txtname" value= "" required maxlength = "15"
18       oninvalid = "disp_txtname_error()"
19       onclick = "setCustomValidity('')"></p>
20
21
22     <p>Here are a set of option buttons:</p>
23     <p><input type="radio" name="gender" id = "gender"
24       value="m" required
25       oninvalid = "disp_gender_error()"
26       onclick = "clear_validity()">Male
27
28       <input type="radio" name="gender"
29       value="f" onclick = "clear_validity()">Female
30     </p>
31
32     <p>Select one of the following majors:</p>
33     <p><select name="Major" id = "Major" required
34       oninvalid = "disp_major_error()"
35       onclick = "setCustomValidity('')">
36       <option value= "" >-Select a major-</option>
37       <option value = "aos">AOS</option>
38       <option value = "cis">CIS</option>
39       <option value = "web">WEB</option>
40       <option value = "acc">ACC</option>
41     </select>
42     </p>
43     <input type="submit" value="Send data to host">
44
45   </form>
46 </body>
47 </html>

```

FIGURE 37. Line 5 indicates that the JavaScript validation code is maintained in a file called site_scripts.js. The lines highlighted in green contain values for the oninvalid and the onclick event handler attributes. The required attribute is specified for these elements. Therefore, the control is deemed to be invalid when the field is left empty when the “submit” button is pressed. The function specified in the onclick attribute is invoked when the user enters a value in the field. The oninvalid attribute is set for radio button group and the list box. Their onclick attribute values are not JavaScript function invocations. Instead, they are JavaScript code. Event handlers can be function invocations or lines of JavaScript code.

The event handler function needs to obtain a reference to an element in the webpage. JavaScript uses the element's id attribute value to do so. JavaScript can retrieve an element by using the `getElementById` method of the document object. This method takes in the id value as an argument and returns the element. Once the element has been retrieved, the "setCustomValidity" method of the control is invoked, and a string argument is sent to it in its invocation. The string that is sent is the error message displayed when HTML5 determines that an error has occurred.

```

1 function disp_txtname_error()
2 {
3     var txtname = document.getElementById("txtname")
4     txtname.setCustomValidity("Name cannot be blank")
5 }
6
7 function disp_gender_error() {
8     var radgender = document.getElementById("gender")
9     radgender.setCustomValidity("Please select a gender")
10 }
11
12 function disp_major_error() {
13     var lst_major = document.getElementById("Major")
14     lst_major.setCustomValidity("You must select a major!")
15 }
16
17 function clear_validity(){
18     var radgender = document.getElementById("gender")
19     radgender.setCustomValidity("")
20 }

```

FIGURE 38. Here is a JavaScript file with the functions invoked in the HTML file. Line 3 of the file declares a variable called `txtname` and uses the `getElementById` method of the document object. The method is sent in the id value of an element from the webpage. Since the argument to the method invocation is "txtname," it retrieves the textbox with that id value from the webpage and places it in the `txtname` variable that is an object variable.

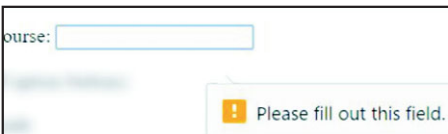
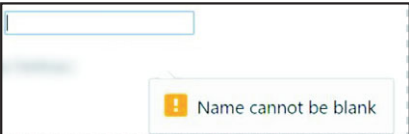
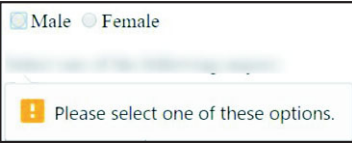
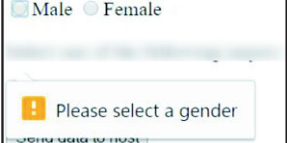
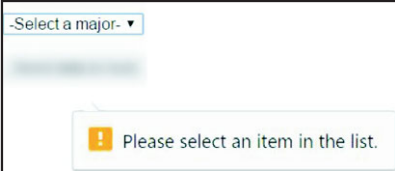
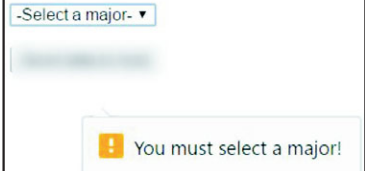
Default HTML5 Message	Custom Error Message Generated via JavaScript Script
	
	
	

FIGURE 39. Here are the default error messages displayed by HTML5 and the custom messages by the JavaScript script.

Summary:



JavaScript (JS) is a cross-platform, object-oriented script language from Netscape and is easier and faster to code than compiled languages (e.g., C and C++) but often takes longer to process than compiled languages. Comment statements are comments placed in JavaScript code to provide documentation within a set of program code. Reserved words/keywords are terms with pre-defined meanings in the language. For example, the reserved word “alert” is used to display data. Literals are entities that do not change value. Numeric literals are entities written as numbers in a program.

Checking Your Knowledge:



1. How are variables declared in JavaScript?
2. How does an array differ from a normal variable?
3. Describe the steps to code a function that writes out a list on a webpage.
4. How does JavaScript generate custom error messages for HTML5 form elements?
5. Where are decision statements used in a webpage?

Expanding Your Knowledge:



Should semi-colons be used to end JavaScript statements? Semi-colons are used in the C++ and Java languages to terminate statements. JavaScript syntax looks similar to C++ syntax, with the exception that semi-colons are not needed at the end of JavaScript statements. In contrast, JavaScript uses a technology called “ASI” (Automatic Semicolon Insertion) that inserts semi-colons as needed when parsing JavaScript code. There is controversy in the IT industry about whether semi-colons should be used in JavaScript or not. This lesson plan does not use semi-colons at the end of statements. To learn more about this matter go to the DailyJS article, JavaScript and Semicolons, at <http://dailyjs.com/2012/04/18/640-semicolons/>.

Web Links:



Eloquent JavaScript

<http://eloquentjavascript.net/>

Henry's HTTP Post Dumping Server

<https://posttestserver.com/>

Reserved Words in JavaScript

<http://www.javascripter.net/faq/reserved.htm>