Module : 5 Introduction to JavaScript: Functions, DOM, Forms, and Event Handlers

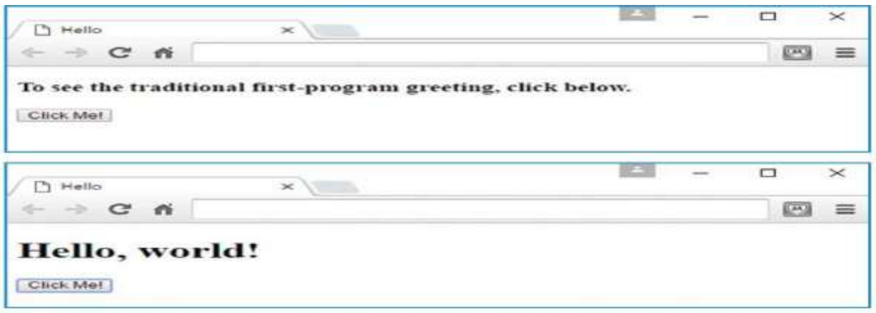
History of JavaScript

HTML's first version, designed by Tim Berners-Lee from 1989 to 1991, was fairly static in nature. Except for link jumps with the a element, web pages simply displayed content, and the content was fixed. In 1995, the dominant browser manufacturer was Netscape, and one of its employees, Brendan Eich, thought that it would be useful to add dynamic functionality to web pages. So he designed the JavaScript programming language, which adds dynamic functionality to web pages when used in conjunction with HTML.

It took Eich only 10 days in May 1995 to implement the JavaScript programming language—a truly remarkable feat. Marc Andreessen, one of Netscape's founders, originally named the new language Mocha and then LiveScript. But for marketing purposes, Andreessen really wanted the name JavaScript. In December 1995, Andreessen got his wish when Netscape obtained a trademark license from Java manufacturer, Sun Microsystems, and LiveScript's name was changed to JavaScript. JavaScript is not all that similar to Java. Actually, C++ and other popular programming languages are closer to Java than JavaScript is. In 1996, Netscape submitted JavaScript to the Ecma International standards organization to promote JavaScript's influence on all browsers (not just Netscape's browser). Ecma International used JavaScript as the basis for creating the ECMAScript standard. In 1998, Netscape formed the Mozilla free-software community, which eventually implemented
Firefox.

Hello World Web Page

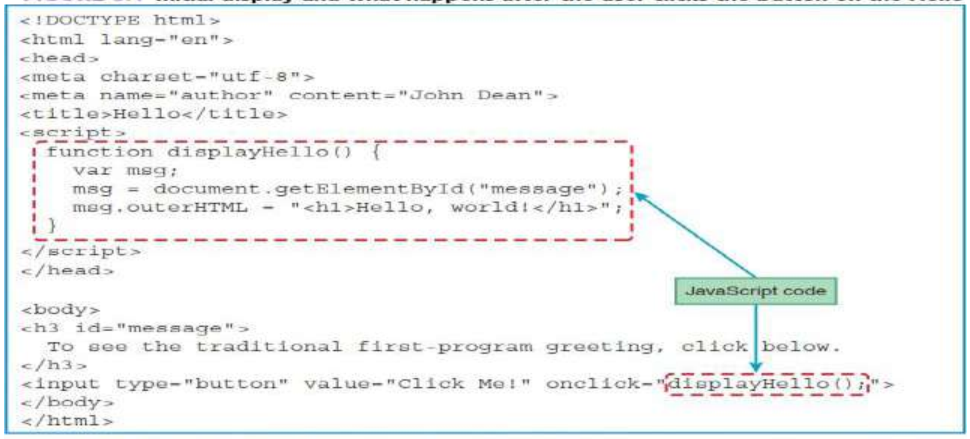
For the book's web pages that use JavaScript, as in the Hello web page shown in Figure 8.1, you'll need to enter the web page's URL in a browser and interact with the web page to fully appre ciate how the web page works. The book's preface provides the URL where you can find the book's web pages. So go ahead and find the Hello web page's URL, enter it in a browser, click the web page's button, and be amazed as the "To see the …" text gets replaced by the large "Hello, world!" text.



To see the traditional first-program greeting, click below.

Hello, world!

FIGURE 8.1 Initial display and what happens after the user clicks the button on the Hello



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Hello</title>
<script>
  function displayHello() {
    var msg;
    msg = document.getElementById("message");
    msg.outerHTML = "<h1>Hello, world!</h1>";
  }
</script>
</head>

<body>
<h3 id="message">
  To see the traditional first-program greeting, click below.
</h3>
<input type="button" value="Click Me!" onclick="displayHello();">
</body>
</html>
```

JavaScript code

FIGURE 8.2 Source code for Hello web page

Now take a look at the Hello web page's source code in FIGUre 8.2. You can see that there's not much JavaScript—it's just the code in the script container and the code that follows the onclick attribute. The rest of the web page is HTML code.

Buttons

To keep things simple, we'll start with just one type of button, and here's its syntax:

```
<input type="button"
value="button-label"
onclick="click-event-handler">
```

Note that the code is a void element that uses the input tag. As its name suggests, the input tag implements elements that handle user input. You might not think of a button as user input, but it is—the user chooses to do something by clicking a button. the input element's most common attributes—type, value, and onclick.

The input element is used for different types of user input, and its type attribute specifies which type of user input. More formally, the type attribute specifies the type of control that's being implemented, where a control is a user input entity such as a button, text control, or check box. In the Hello web page source code, note that the type attribute gets the value button, which tells the browser to display a button. If you don't provide a type attribute, the browser will display a text control, because that's the default type of control for the input element. For now, just know that a text control is a box that a user can enter text into, and this is what a (filled-in) text control (with a prompt at its left) looks like:

Because it uses a box, many web developers refer to text controls as "text boxes." We use the term "text control" because that's the term used more often by the HTML standards organizations.

The input element's value attribute specifies the button's label. If you don't provide a value attribute, the button will have no label. If you want a button with no label, rather than just omit ting the value attribute, we recommend that you specify value="". That's a form of self-documentation, and it makes your code more understandable.

The input element's onclick attribute specifies the JavaScript instructions that the JavaScript engine executes when the user clicks the button. What's a JavaScript engine, you ask? A JavaScript engine is the part of the browser software that runs a web page's JavaScript. In the Hello web page, the onclick attribute's value is JavaScript code that "handles" what's supposed to happen when the user clicks the button. Clicking the button is considered to be an event, so the onclick attribute's JavaScript code is known as an event handler.

Functions

A function in JavaScript is similar to a mathematical function. A mathematical function receives arguments, performs a calculation, and returns an answer. For example, the sin(x) mathematical function receives the x argument, calculates the sine of the given x angle, and returns the calcu lated sine of x. Likewise, a JavaScript function might receive arguments, will perform a calculation, and might return an answer. Here's the syntax for calling
a function:

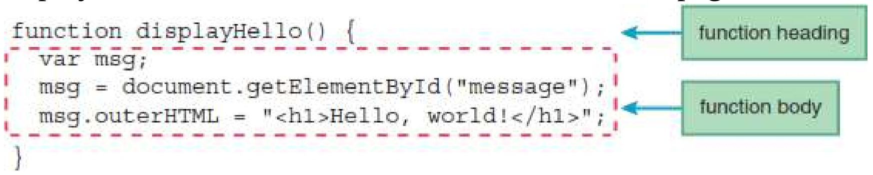
```
function-name(zero-or-more-arguments-separated-by-commas);
```

Hello web page button has an onclick attribute with a value of displayHello();. That's a JavaScript function call, and its syntax matches the preceding syntax. Note the parentheses are empty because there's no need to pass any argument values to the displayHello function. If there were arguments,

they would need to be separated by commas, and proper style suggests that you insert a space after each comma.
Here's the syntax for a function definition:

```
function function-name(zero-or-more-parameters-separated-by-commas) {
  statement-1;
  statement-2;
  …
  last-statement;
}
```

And here's the displayHello function definition from the Hello web page:



```
function displayHello() {
  var msg;
  msg = document.getElementById("message");
  msg.outerHTML = "<h1>Hello, world!</h1>";
}
```

The parentheses in the function heading are empty because the function call's parentheses are empty (the function call was displayHello();). If there are arguments in the function call, then you'll normally have the same number of parameters in the function heading one parameter to receive each argument's value.

Normally, function definitions should be placed (1) in a script container in the web page's head container or (2) in an external JavaScript file. Looking at the previous code fragment, you can see three lines in the function's body. Each line is a JavaScript statement, where a statement performs a task. Note the semicolons at the end of all three statements. Semicolons are required at the end of a JavaScript statement only if the JavaScript statement is followed by another JavaScript statement.

Variables

Here's the function's first statement:

```
var msg;
```

The msg thing is a variable. You should already be familiar with variables in algebra. You can think of a variable as a box that holds a value. In this case, the msg variable will hold a string that forms a message. Before you use a variable in JavaScript code, you should use var to declare the variable in a declaration statement. For example:

```
var name;
var careerGoals;
```

Words that are part of the JavaScript language are known as keywords. The word var is a keyword. On the other hand, name and careerGoals are not keywords because they are specific to a particular web page's code and not to the JavaScript language in general. In that function, besides var, can you identify another keyword? The function heading uses the word function and function is a keyword. With most of JavaScript's keywords, it's illegal for you as a programmer to redefine them to mean something else. So you cannot use "function" as the name of a variable. Those keywords are "reserved" for the JavaScript language, and they are known as reserved words.

In most programming languages, when you declare a variable, you specify the type of values that the variable will be allowed to hold—numbers, strings, and so on. However, with

JavaScript, you do not specify the variable's type as part of the declaration. The variable's type is determined dynamically by the type of the value that's assigned into the variable. For example

name = "Mia Hamm";
careerGoals = 158;

JavaScript is known as a loosely typed language, or a dynamically typed language, which means that you do not declare a variable's data type explicitly, and you can assign different types of values into a variable at different times during the execution of a program.

Identifiers

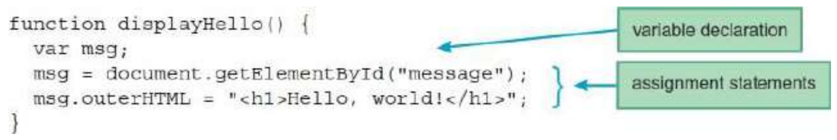
In the Hello web page, displayHello was the identifier for the function name, and msg was the identifier for a variable. In naming your variables and functions, the JavaScript engine requires that you follow certain rules. Identifiers must consist entirely of letters, digits, dollar signs ($), and/or underscore (_) characters. The first character must not be a digit. If you do not follow these rules, your JavaScript code won't work.

Coding-convention rules are narrower than the preceding rules. Coding conventions suggest that you use letters and digits only, not dollar signs or underscores. They also suggest that all letters should be lowercase except the first letter in the second word, third word, and so on. That's referred to as camel case, and here are a few examples: firstName, message, daysInMonth. Notice that the identifiers' words are descriptive.

If any of the coding conventions are broken, it won't affect your web page's ability to work properly, but your code will be harder to understand and maintain. If your code is harder to understand and maintain, that means programmers who work on the code in the future will have to spend more time in their efforts, and their time costs money. Normally, programmers spend more time working on old code (making bug fixes and making improvements) rather than writing new code, and all that work on old code costs money.

Assignment Statements and Objects

Once again, here's the Hello web page's displayHello function:



```
function displayHello() {
  var msg;
  msg = document.getElementById("message");
  msg.outerHTML = "<h1>Hello, world!</h1>";
}
```

In the function's body, the first statement is a variable declaration for the msg variable. After you declare a variable, you'll want to use it, and the first step in using a variable is to put a value inside it. An assignment statement puts/assigns a value into a variable. As you can see in the preceding example, the function body's second and third statements are assignment statements. The assignment operator (=) assigns the value at the right into a variable at the left. So in the first assignment statement, the document.getElementById("message") thing gets assigned into the msg variable. In the second assignment statement, the "<h1>Hello, world!</h1>" thing gets assigned into the msg.outerHTML variable. An object is a software entity that represents something tangible.

Behind the scenes, all of the elements in a web page are represented as objects. When a browser loads the Hello web page, the browser software generates objects for the head element, the body element, the h3 element, and so on. There's also an object associated with the entire web page, and that object's name is document. Each object has a set of related properties, plus a set of behaviors. A property is an attribute of an object. A behavior is a task that the object can perform.

The document object contains properties like the web page's type. Most web pages these days (and all the web pages in this book) have a value of HTML5 for the document object's type property.

```
<!DOCTYPE html>
```

The html value indicates that the document object's type is HTML5. To access an object's property, you specify the object name, a dot, and then the property name. So to access the current web page's document type, use document for the object name, . for dot, and doctype for the property. Here's the JavaScript code:

```
document.doctype
```

Remember that an object is not only a set of properties, but also a set of behaviors. One of the document object's behaviors is its ability to retrieve an element using the element's id value. To retrieve an element, the document object uses its getElemementById method. To call an object's method, you specify the object name, a dot, the method name, and then parentheses around any arguments you want to pass to the method.

```
document.getElementById("message")
```

See how the method call includes "message" for its argument? In executing the method call, the JavaScript engine searches for an element with id="message". There is such an element in the Hello web page, and here it is:

```
<h3 id="message">
  To see the traditional first-program greeting, click below.
</h3>
```

So the getElementById method call retrieves that h3 element.
The HTML5 standard says that an id attribute's value must be unique for a particular web page. You might recall how we used an id attribute to identify a target for a link within a web page.

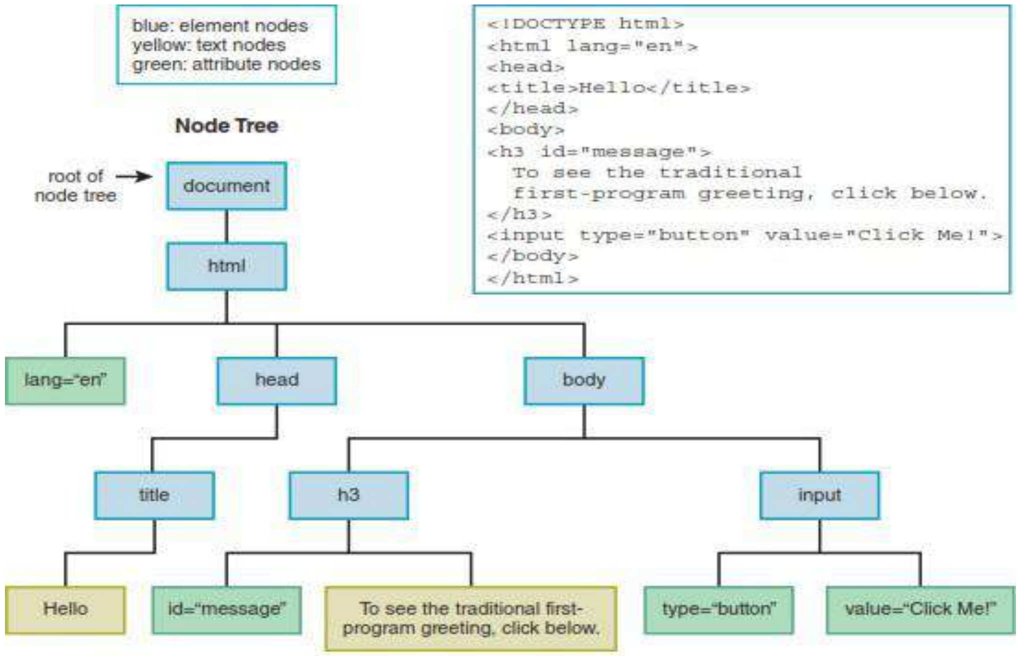
```
function displayHello() {
  var msg;
  msg = document.getElementById("message");
  msg.outerHTML = "<h1>Hello, world!</h1>";
}
```

In the displayHello function, you can see that the getElementById method call is on the right hand side of an assignment statement, so the method's returned value (the h3 element's object) gets assigned into the variable at the left of the assignment statement. After msg gets the h3 element's object, that object gets updated with this assignment statement:

```
msg.outerHTML = "<h1>Hello, world!</h1>";
```

Document Object Model

The Document Object Model, which is normally referred to as the DOM, models all of the parts of a web page document as nodes in a node tree. A node tree is similar to a directory tree, except instead of showing directories that include other directories (and files), it shows web page elements that include other elements (and text and attributes). Each node represents either (1) an element, (2) a text item that appears between an element's start and end tags, or (3) an attribute within one of the elements. The figure's code is a stripped-down version of the Hello web page code shown earlier, with some of its elements and attributes (e.g., the meta and script elements) removed. The node tree shows blue nodes for each element in the web page code.



```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Hello</title>
</head>
<body>
<h3 id="message">
  To see the traditional
  first-program greeting, click below.
</h3>
<input type="button" value="Click Me!">
</body>
</html>
```

FIGURE 8.3 **Node tree for simplified Hello web page**

The term dynamic HTML refers to updating the web page's content by manipulating the DOM's nodes. Assigning a value to an element object's outerHTML property (as in the Hello web page) is one way to implement dynamic HTML.

The DOM provides different ways to access the nodes in the node tree. Here are three common techniques:

1. You can retrieve the node tree's root by using document (for the document object) in your code and then use the root object as a starting point in traversing down the tree.
2. You can retrieve the node that the user just interacted with (e.g. a button that was clicked) and use that node object as a starting point in traversing up or down the tree.
3. You can retrieve a particular element node by calling the document object's getElementById method with the element's id value as an argument.

Forms and How They're Processed: Client-Side Versus Server-Side

A form is a mechanism for grouping input controls (e.g., buttons, text controls, and check boxes) within a web page.

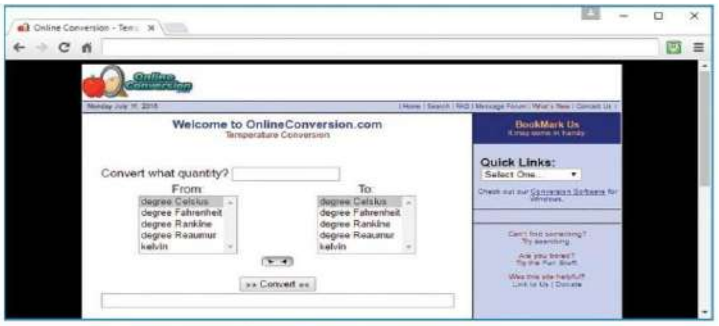


FIGURE 8.4 **Web page that performs temperature conversions**

There are two basic strategies for processing a form's input data. The calculations may occur on the client side (on the browser's computer) or on the server side (on the web server's computer). With server-side processing, the form input values are transmitted across the Internet to the server computer. The server then does the calculations and transmits the answers back to the client computer With client-side processing, there's no need to go back and forth across the Internet with user input and generated results. After the web page downloads, the client computer does all the work. Therefore, client-side processing tends to be faster. So normally, you should use client-side processing for relatively simple web pages. On the other hand, there are several reasons why server-side processing is sometimes preferable:

- When the calculations require a lot of programming code. If client-side processing were used, all the calculation code would have to be downloaded to the client, and that would slow things down.
- When the calculations require the use of large amounts of data, which usually means using a database. The rationale is basically the same as for the case where there's lots of programming code. With large amounts of data, you don't want to have to download it across the Internet to the browser because that would slow things down.
- Proprietary code is code that gives the programmer (or, more often, the programmer's company) a competitive advantage. You should keep proprietary code on the server side, where it's more difficult for a competitor or hacker to access it.
- When the inputs and/or calculation results need to be shared by other users. In order for the data to be shared, it needs to be transmitted to the server so it can be later transmitted to other users.
- When user information needs to be processed securely behind the scenes on the server. For example, credit card numbers and passwords should be processed on the server side.

form Element

Here's a template for the form element's syntax:

```
<form>
   label
   text-box, list-box, check-box, etc.
   label
   text-box, list-box, check-box, etc.
   ...
   submit-button
</form>
```

Note how there's a submit button control at the bottom and other controls above it. That's probably the most common layout because it encourages the user to first provide input for the controls at the top before clicking the button at the bottom. If it's more appropriate to have your submit button at the top or in the middle, then you should put your submit button at the top or in the middle. One other thing to note in the template is the labels. The labels are text prompts that tell the user what to enter in the subsequent controls.

The following code implements a form with two text controls and a submit button:

```
<form>
  First Name:
```

```
  <input type="text" id="first" size="15"><br>
  Last Name:
  <input type="text" id="last" size="15"><br><br>
  <input type="button" value="Generate Email"
    onclick="generateEmail(this.form);">
</form>
```

Notice how this code matches the template provided earlier. The first two controls are text controls that hold first name and last name user entries. The bottom control is a button. When the button is clicked, its onclick event handler calls the generateEmail function that combines the entered first and last names to form an email address.

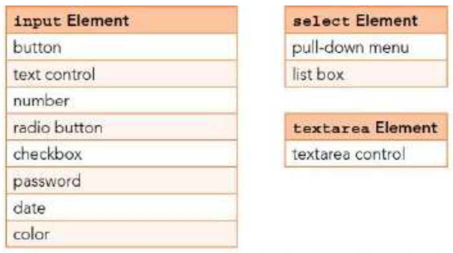
Although it's legal to use input elements—like text controls and buttons—without surrounding them with a form element, you'll usually want to use a form. Here are some reasons for doing so:

- Forms can lead to faster JavaScript processing of the input elements. Understanding why that's the case will make sense after we explain the JavaScript code in an upcoming web page later in this chapter.
- Forms provide support for being able to check user input to make sure it follows a particular format. That's called input validation, and we'll spend a considerable amount of time on it in the next chapter.
- Forms provide the ability to add a reset button to assign all the form controls to their original values. To implement a reset button, specify reset for the type attribute, like this:

```
<input type="reset" value="Reset">
```

Controls

Figure 8.6 shows some of the more popular controls and the elements used to implement them. As you can see, most of the controls use the input element for their implementation.



input Element
button
text control
number
radio button
checkbox
password
date
color

select Element
pull-down menu
list box

textarea Element
textarea control

FIGURE 8.6 Some of the more popular controls and the elements used to implement them

just know that the number control provides a mechanism for users to enter a number for input, and it has built-in checking to make sure the input is a properly formatted number. The password control allows the user to enter text into a box where, to help with privacy, the entered characters are obscured. Typically, that means the characters display as bullets. The date control allows the user to enter a month-day-year value for a date. Most browsers implement the date control with a drop-down calendar where the user picks a date from it. The pull-down menu control normally displays just one item at a time from the list and displays the rest of the list only after the user clicks the control's down arrow. On the other hand, the list box control displays multiple list items simultaneously without requiring the user to click a down arrow. control that uses the textarea element—the textarea control

Text Control

```
<input type="text" id="text-box-identifier"
  placeholder="user-entry-description"
```

```
size="box-width" maxlength="maximum-typed-characters">
```

The preceding text control template does not include all the attributes for a text control—just the more important ones. We'll describe the attributes shown, plus a few others shortly, but let's first look at an example text control code fragment:

```
<input type="text" id="ssn"
placeholder="#########" size="9" maxlength="9">
```

The text control is for storing a Social Security number, so the id attribute's ssn value is an abbreviation for Social Security number. What's the purpose of the nine #'s for the placeholder attribute? Social Security numbers have nine digits, so the nine #'s implicitly tell the user to enter nine digits with no hyphens.

Attributes

Text Control Attributes								
type	id	placeholder	size	maxlength	value	autofocus	disabled	readonly

- always include type="text" for your text controls.
- The id attribute's value serves as an identifier for the text control, so it can be accessed with JavaScript
- The placeholder attribute provides a word or a short description that helps the user to know what to enter into the text control.
- The size attribute specifies the text control's width (size="5" means approximately 5 characters could display in the box simultaneously. The default size is 20)
- The maxlength attribute specifies the maximum number of characters (unlimited)
- The value attribute specifies an initial value for the text control.
- The autofocus attribute specifies that after the page has loaded, the browser engine positions the cursor in the text control.
- The disabled attribute specifies user cannot copy or edit the text control's value.
- The readonly attribute specifies that the user can highlight the control's value and copy it, but the user cannot edit it.

Accessing a Form's Control Values

As you learned earlier, whenever you pass an argument to a function, you should have an associated parameter in the function's heading. Therefore, to receive the form object passed to the generateEmail function, there's a form parameter in the function's heading

```
function generateEmail(form)
```

Figure 8.9B shows the head container for the Email Address Generator web page. Note the form parameter in the generateEmail function's heading. Within the generateEmail function body, we use the form parameter to retrieve the text control user inputs. Here's the code for retrieving the user input from the first-name text control:

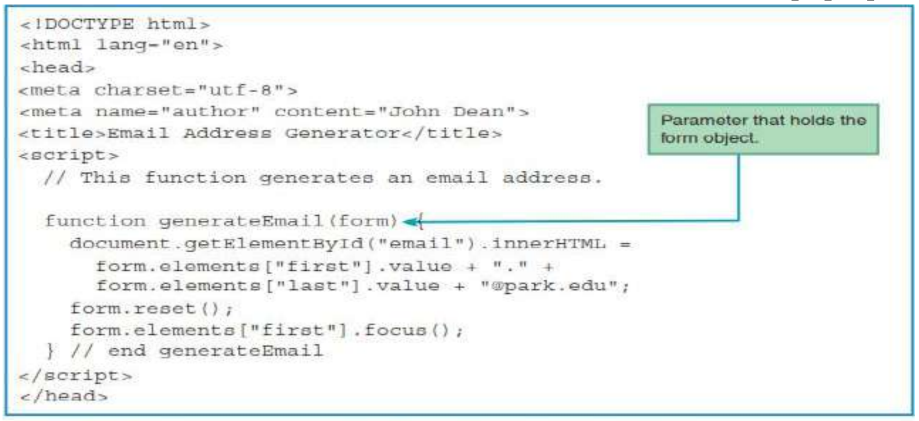
```
form.elements["first"].value
```

To access the controls that are within a form, we use the form object's elements property. The elements property holds a collection of controls, where a collection is a group of items that are of the same type. To access a control within the elements collection, you put quotes around the control's id value and surround the quoted value with []'s.

Here are the corresponding JavaScript properties for a text control element object:

type, placeholder, size, maxLength, value, autofocus, disabled, readOnly

Note that HTML attributes use all lowercase, whereas JavaScript properties use camel case.



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<meta name="author" content="John Dean">
<title>Email Address Generator</title>
<script>
  // This function generates an email address.

  function generateEmail(form) {
    document.getElementById("email").innerHTML =
      form.elements["first"].value + "." +
      form.elements["last"].value + "@park.edu";
    form.reset();
    form.elements["first"].focus();
  } // end generateEmail
</script>
</head>
```

FIGURE 8.9B **head container for Email Address Generator web page**

Control elements' **innerHTML** property

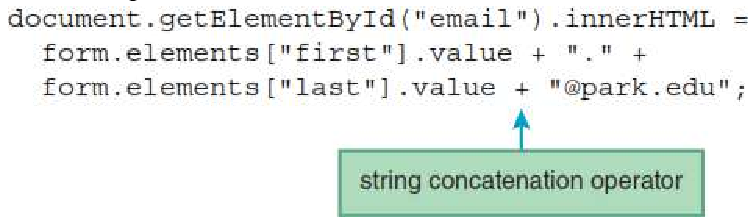
In the Email Address Generator web page's generateEmail function, the goal is to update the following p element by replacing its empty content with a generated email address:

<p id="email"></p>

To do that, we retrieve the p element's object and then use its innerHTML property, like this:

document.getElementById("email").innerHTML

In the generateEmail function, here's the assignment statement that uses innerHTML to update the p element with a generated email address:



```
document.getElementById("email").innerHTML =
  form.elements["first"].value + "." +
  form.elements["last"].value + "@park.edu";
```

To connect a string to something else (e.g., another string, a number), you need to use the concatenation operator, +. The resulting connected value forms a string.

reset and focus Methods

Go back to Figure 8.9B, and you can see that we still haven't talked about the last two lines in the generateEmail function. Here are those lines:

form.reset();
form.elements["first"].focus();

The form object's reset method reassigns the form's controls to their original values. Because the Email Address Generator web page has no value attributes for its text controls, the reset method call assigns empty strings to the text controls, thereby blanking them out. When an element object calls the focus method, the browser puts the focus on the element's control if it's possible to do so. For text control elements, like the first-name text control retrieved in the preceding code, putting the focus on it means the browser positions the cursor in the text control.

Questions:

1. What is a function in Java JavaScript? Explain syntax for a function definition with an example.
2. List Some of the more popular controls and the elements used to implement them.
3. Explain text control attributes.
4. Write a note on reset and focus Methods.
5. Explain the history of JavaScript.
6. Explain the following with an example i) Buttons ii) Functions iii) Variables iv)Identifiers
7. What are the advantages and disadvantages of Client-Side and Versus Server-Side.
8. Write a Java Script program that on clicking a button, displays scrolling text which moves from left to right with a small delay.
9. Create a webpage containing 3 overlapping images using HTML, CSS and JS. Further when the mouse is over any image, it should be on the top and fully displayed.