Understanding Form Validation Exercises

Handout for Computer Technology iv

Form Validation Exercises can be found at
http://qserver.zobel.dlsu.edu.ph/sirpaolo/activities/javascript/validation.html

JavaScript has been applied in many different ways, but probably the most practical and most popular application is its use in form input validation. The advent of Web forms allows users to interact with Web sites through sending data, but of course such technology creates possibilities of server errors because of incorrect data. This hands-on activity explores some basic techniques used in validating form input.

Client-side vs. Server-side

By performing form input validation, we can make sure that the data entering the server is more or less error-free. There are two ways of implementing form input validation: server-side (done on the Web server) and client-side (done on the user's computer).

Using just server-side validation works as a solution, but is not the optimal one. Once the user submits form data, the information has to be transmitted to the server, where a program (and memory) is used to validate the data, and sent back to the user if there are errors. What if the error is simply a field that was left blank by the user? Such a method is inefficient, and this inefficiency is the rationale behind client-side validation. In client-side validation, data is already checked before it gets to the server, thereby preventing most simple errors.

JavaScript is typically used to implement client-side validation, and the hands-on exercise provides a few sample scripts that perform four kinds of validations: textbox input, password, radio button selection, and email address.

Understanding the Code

This section explains how the scripts work and presents certain lines of code in detail.

Input Validator

The Input Validator script makes sure that a textbox is filled up prior to submission to a processing server. The function validateInput() performs the following steps:

1. The script extracts the value of the textbox (input tag with id="textbox" in the body of the document).
2. The **if** conditional statement checks whether the value of the textbox (from #1) is empty. If empty, it displays the paragraph that contains the error message (which is hidden through CSS when the page loads).
3. If the value of the textbox (from #1) is not empty, the script enters the **else** part of the conditional statement. It hides the paragraph containing the error message through CSS (in case it has been displayed) and presents an alert box containing the user input.

This script is simple enough to understand, but here are some additional explanations:

```
    strInput = document.getElementById("textBox").value
```
This line of code extracts the value of the textbox (input tag with id="textbox" in the body of the document) using the JavaScript getElementById() function and assigns it to a variable named strInput.

```
    if (strInput == "")
```

This line of code presents an **if** conditional that checks whether the variable strInput is empty. An empty string is denoted by the value "" (two consecutive quotation marks with no space in between). Note that the conditional will check for a different value if a space or other characters were in between the quotation marks (it will check if the string is equal to a space). If the string is empty, the conditional generates a **TRUE** value and proceeds to the program subsection defined in the curly braces that follow this snippet of code. Otherwise, a **FALSE** value is generated and the script skips the subsection inside the curly braces.

```
    document.getElementById("errorMsg").style.display = "inline"
```

This line of code sets the value of the tag with id="errorMsg" to a value of **inline**. The .display part of the DOM accesses the CSS property Display, which controls how an element is displayed on the Web page. The default value if **inline**, which displays the element as part of the current line in the paragraph. A value of **none** prevents an element from being displayed on the Web page (hides the element).

Password Validator

This script makes sure that a password is typed properly before submission to a Web site. Web sites typically employ this kind of validation when users register to access a Web site's services. In this script, users are asked to type a password twice. This is done not only for comparison purposes, but also to make sure that the password is typed properly. It's very easy to make a mistake in typing a password, because the password appears as a series of dots or asterisks instead of regular characters. Typing a password twice ensures that a password is the correct one being registered.

The Password Validator performs the following tasks:

1. The script extracts the values of the two password textboxes from the document (with ids "pass1" and "pass2").
2. The succeeding **if** conditional checks whether the first password textbox was left empty. If it is empty, the script generates an error message displayed on the Web page.
3. If the password is not empty, the script goes into the **else** part. An **if-else** conditional is nested inside the conditional checking for empty passwords, checking if the typed passwords are equal to one another. If the two passwords are equal, the script goes into another nested **if-else** conditional, this one checking whether the password contains 8-15 characters.
4. If the password meets all the conditions, it is considered valid. Otherwise, the appropriate error message is generated and displayed on the Web page.

Here's an additional explanation for a line of code present in the Password Validator:

```
    if (strInput1.length >=8 && strInput1.length <=15)
```
This conditional checks whether the password contains 8-15 characters. Let's break it down further. The **if** keyword denotes the start of a conditional statement, and the script checks the condition inside the parenthesis.

The first condition to be met is **strInput1.length >=8**. Simply put, the length of the strInput1 variable should be at least 8 characters. The .length part of the DOM allows the script to determine the length of the variable or object. In the case of strings, it determines the number of characters comprising the string variable.

The ampersands (**&&**) indicate a union or joining of two or more conditions, and subsequent conditions must be met in order for the conditional to generate a TRUE value.

The second condition to be met is **strInput1.length <=15**. This time, we are looking for strInput1 to be at most 15 characters. Combining everything, we want strInput1 to be at least 8 characters and at most 15 characters, or simply, containing 8-15 characters.

Selection Validator

The Selection Validator makes sure that a choice is made between two radio buttons. The script is written in such a way that it can accommodate any number of radio buttons (more than 2). It performs the following tasks:

1. It creates a variable called genderChoice and gives it an arbitrary value of -1. This variable is used to indicate whether a choice has been made or not, and at this point, it is assumed that no choice has been made. The value -1 does not really have any special meaning to it. It was set to -1 simply because the succeeding lines in the function can't make the value of genderChoice negative.
2. The succeeding **for** loop cycles or goes through all of the radio buttons in the Web page. Inside that loop, a conditional statement checks whether the current radio button is selected. If the current radio button is selected, the value of genderChoice is changed to the index of the radio button (0 is the first radio button, 1 the second, 2 the next, so on…) and the value of the choice is displayed. This value is contained in the input tag in the body of the document.
3. After exiting the **for** loop, a conditional checks whether the value of genderChoice remains -1. If the value remains negative, it is obvious that none of the choices were selected. The script generates a message that alerts the user to select either the "Male" or "Female" value.

The **form** tag used in this document looks different than the ones used in other scripts, and an additional explanation is presented here:

```
    <form name="myForm" onSubmit="return validateChoice(this)">
```

The **onSubmit** attribute is a JavaScript event that defines any events that will be executed when the user clicks on the Submit button. In other exercises, we've used the **action** attribute to execute our functions. However, the **action** attribute, aside from allowing the execution of JavaScript code, is typically used to indicate the address of the Web server where the data will get sent. We use the **onSubmit** attribute to intercept the data before it gets sent to the Web server. The **return** keyword requires that the function send back a **TRUE** or **FALSE** value, which either allows the page to continue or halt the sending of data to the processing server. Lastly, the keyword **this** refers to the current form being processed, in this case, the one with id="myForm".

Email Validator

The Email Validator makes sure that an email address entered by the user follows the correct format for electronic mail addresses. It performs the following tasks:

1. It creates a variable **validFormatRegExp** and sets it to a value of **/^\w(\.?\w)*@\w(\.?[-\w])*\.[a-z]{2,4}$/I**, which represents a JavaScript regular expression (explained later).
2. The succeeding **if** conditional tests the value of the regular expression against the email address extracted from the textbox in the Web page. If the input follows the correct format for an email address, the user is told that the input is a valid email address. Otherwise, the program enters the **else** part which alerts of an invalid email address.

One of the lines in this script looks a little strange, and merit additional explanations:

```
validFormatRegExp = /^\w(\.?\w)*@\w(\.?[-\w])*\.[a-z]{2,4}$/i
```

The series of otherwise garbage-looking characters in this line is a JavaScript regular expression. The data that users input does not ascribe to just one specific pattern. It can appear in many different ways, and regular expressions are used to evaluate patterns of data. In this case, if the email address follows the correct pattern indicated in the regular expression, it is considered as a valid email address.

Consult these references if you want a more comprehensive explanation than otherwise provided in this handout: Practical JavaScript for the Usable Web by Wilton, et.al.; HTML for the World Wide Web by Elizabeth Castro, 5th edition; JavaScript and AJAX for the World Wide Web by Tom Negrino and Dori Smith, 5th Edition.