Learning Outcome 5: Manage form data with JavaScript

# Learning Step 5.1: Describe how form information is transmitted to a server

Example: <form name="aForm" id="aForm" action="FormSubmit.html" method="get">

<input type="text" id="name" name="name" />

<input type="text" id="age" name="age" />

FormSubmit.html?name=Mike&age=50

**Activity:** See **aform.html** and compare the results of GET and POST

# Learning Step 5.2: Use JavaScript form objects to collect user input

**Activity:** Try playing with **simpleform-start.html** (and **styles/form.css**). See what data gets passed along in the URL. In particular, note what values get submitted for the Size selection box, the Colour radio buttons, and the forChild checkbox. Compare it to the HTML code.

When a form is submitted, a “submit” event is fired. Note that this event is fired on the form, NOT on the submit button.

We can use a submit handler on the form to work with the field values before they are submitted to the server. We can also cancel a form submission (for instance, if your submit handler returns a value of false).

**Activity:** Add a submit handler (the event name is “submit”) to the form so that an alert box is displayed indicating that the form is being submitted, and then cancel the form submission.

The submit handler needs to return false to cancel the submission, or you can use the event object to call the preventDefault() method.

**Accessing a form using the legacy DOM**

We can access the form using:

* document.forms[0]
* document.forms["simpleForm"]
* document.forms.simpleForm
* document.forms.simpleFormID (or ["simpleFormID"]

We can also access the form using getElementById or other selectors.

Forms have an elements collection, one element for each form element. The length of the elements array is the number of input controls in the form (including fieldsets, buttons, etc.).

**Practice:** Use the JavaScript console to find out how many elements we have in our form. Is this what you expect?

My solution:

document.forms[0].elements.length

All input elements have properties. At minimum:

* form: a reference to the containing form
* name: the value of the name attribute of this element
* type: the value of the type attribute of the element (or other types like textarea or select-one)
* value: the value that the website visitor has entered into the form for this element

Depending on the input element’s type, the element may have additional properties (for example, the checked property for radio buttons and checkboxes).

**Practice:** Change your submission handler to loop through all the ~~input~~ elements of the form. Create a string with one line for each element, indicating the number of the element and the values of its name, type, and value properties. Display this string in an alert box (or at the console), and then cancel the form submission.

We can change the value in a text box or select box by assigning a string to the value property of that element.

For other input elements, we’ll need to check different properties:

* defaultValue: elements of type text, textarea, password, and fileUpload (among others) have a defaultValue property – that is the value that is hardcoded into the HTML, the value that is initially displayed when you first bring up the page
* checked: elements of type checkbox and radio have a Boolean value that indicates if the radio button or checkbox is checked.
* defaultChecked: for checkboxes and radio buttons, indicates the default state hardcoded into the HTML

Select boxes have some other interesting properties:

* length: the number of options in the select box
* options: a collection of the options
* selectedIndex: the index of the FIRST selected option, or -1 if no options are selected. If multiple selected options are possible, you need to loop through the options array and look at their “selected” Boolean property.

The elements in the options array have other properties:

* defaultSelected: Boolean indicating if the options is selected by default (hardcoded into the HTML)
* text: a string that indicates what is displayed for that option
* value: a string that indicates the value that would be submitted with the form if that option is selected

**Extended practice:** Change your submit handler to display an alert box/console log with the following information:

* the value and default value of the “name” text input
* the length and selected index of the selected option in the “size” select box
* the text and value of the selected option in the “size” select box, if one is selected
* which of the three “colour” buttons has been checked
* which of the three “colour” buttons is checked by default
* whether the “forChild” checkbox is checked, and whether it is checked by default

# Learning Step 5.3: Construct regular expressions to use with form validation

Ways to do validation:

* No validation (bad)
* Just HTML5 validation – limited
* JavaScript alone for validation – can be complex
* Validation with a plugin (such a jQuery extension)

**Practice:** Change your simple form so that you can check to see if anything has been entered into the “name” text input. If anything has been entered (even just a bunch of spaces), accept the form submission. If nothing has been entered, reject the form submission. For now, have your submission handler always return false or call event.preventDefault() so that we stay on the same page, but display an alert box indicating whether the form is valid.

**Practice:** Before the buttons, add a <span> that will contain an error message. Give that span a class of “errorMessage” and update your form.css file to use a text color of red for that class. Update your submit handler so that in the case of an error, we put an error message in that span AND change the background color of the “name” box so that it is “tomato”; if successful, clear the error message span and remove the background color from the “name” box.

**Other form events:**

The following event handlers could be invoked for a form element (NOT the form itself):

* **onclick** – generated by clicking the mouse on a form element (such as a button, checkbox, etc.)
* **onchange** – generated when the form element has been edited and changed (such as for text boxes, select boxes) – note that this event won’t be fired until we leave the field
* **oninput** – generated when input is made or changed in the form element (such as for text boxes) – note that this event is fired when the field still has the focus
* **onfocus** – generated when the form element receives focus (tabbed into or clicked on)
* **onblur** – generated when the form element loses focus (tabbed out of or clicked away from)

**Regular expressions in JavaScript**

Regular expressions allow us to compare a string against a pattern.

JavaScript has regular expression objects called RegExp. You can create a regular expression using a constructor or a literal. The literal is a sequence of characters with slashes around it: /*regexp*/

String objects have a method called search() that we can use to search for patterns in the string:

*someString*.search(*regexp*)

This returns the index where the pattern occurs in the string, or -1 if it doesn’t match.

Note: there’s also a match() method that we can use if we are looking for multiple occurrences.

The simplest regular expression looks for an exact match.

let str = "Michael";

let index = str.search(**/ch/**);

What will index contain? *2 – it looks for the string “ch” inside str and returns the index where that string starts.*

Other relevant functions:

* *someString*.match( *regexp* ): like search, but returns an array of the matches found, or null if there were no matches found
* *someRegExp*.test( *string* ): returns true if the supplied string matches the regular expression that test() is called on, and false if it doesn’t

**Practice:** Create a regular expression that checks if the string contains the phrase “bob” (anywhere in the string). Return false if it does not.

*Solution:* **/bob/**.test("Bobobo")

We can also use special flags on regular expressions to indicate different things:

* i: ignore case
* g: global search (doesn’t stop after the first match)
* m: multi-line mode (helps when you’re looking for the start and end of the line)

So to search for "Bob", ignoring case:

**/bob/i**.test("BoB")

**More advanced features of regular expressions:**

See <https://regex101.com/> or google “regular expression tester”

You can match one of a set of characters by putting the characters in square brackets:

**[abc]** will match one of a, b, or c

If you want to have a bunch of characters and don’t want to type them all in, you can have a range:

**[a-f]** is the same as **[abcdef]**

You can also say NOT to match specific characters. In this case, you put a ^ at the front of the characters.

**[^a-f]** will match any characters besides a, b, c, d, e, or f

You can say that a character can occur multiple times with a **+**

**a+** means that “a” has to match 1 or more times.

You can say that a character needs match 0 or 1 times with a **?**

**/Rege?/i** will match both the “reg” in regular and the “rege” in RegExp

You can say that the character needs to match 0 or more times with a **\***

**e[a-z]\*s** will match any sequence of characters starting with an e, ending with an s, and having 0 or more letters in the middle

**Practice:** Try creating a regular expression that will only allow letters, spaces, and digits, with at least one character matching.

*Solution:* **/[a-z 0-9]+/i**

You can specify that an expression matches either of two patterns by using the | pipe character, so **(abc|def)** would match either “abc” or “def”.

You can group characters together using parentheses.

You can specify that you want an expression to only match at the start of a string with a ^ *(note: not in square brackets)*: **/^bob/**

You can also specify that you want an expression to only match at the end of a string with a $: **/bob$/**

These are often combined to say that we want an exact match: **/^bob$/**

**Practice:** Create a regular expression that will match Robert, Bob, or Rob but not Bobert. Note that **/[rb]ob(ert)?/i** will not work because it matches Bobert.

*Solution:* **/^(bob|rob(ert)?)$/i**

**Practice:** Write a regular expression that checks to see if a string contains only letters and numbers (from the start to the end)

*Solution:* **/^[a-z0-9]+$/i**

Additional information: Go through the quick start found at <http://www.regular-expressions.info/>

# Learning Step 5.4: Use regular expressions and a JavaScript library to validate form data

**jQuery Form Validation**

jQuery has a large number of “plugins” for a variety of tasks. One of these tasks is form validation. There are many form validation plugins – we will look at one:

<https://jqueryvalidation.org>

The jQuery validation plugin works by using the HTML5 attributes that are assigned to the elements, and also by looking for specific classes assigned to elements. It will create a label with the class “error” that is added immediately after the field with the error; also, a class “error” is added to the field with the error. See **jqueryvalidateform-start.html** and **styles/jform.css** for an example – see how the formatting is set up in the CSS file.

Once the error has been cleared, the label is removed, and the error class is removed from the field.

We can add validation by using HTML5 input elements and attributes, and they will automatically be reformatted using the jQuery validation style.

**Practice:** Make the CST ID field, the Email field, and the Age field required.

*Solution:* After adding jQuery and the jQuery validation plug-in, just call validate() on the form and add the attribute “required” to the relevant form fields.

**Additional validation values** (other than **required**)

These HTML5 input types (among others) are also validated immediately:

* email
* url
* number

You can also add classes to the fields to apply validation:

* required
* date
* url
* email
* number
* digits – like number, but must contain only nonnegative integers
* creditcard – must contain a validly formatted credit card number

**Activity:** Update your form to require the email field to be an email address, age to be a nonnegative integer, and homepage to be a URL.

**Advanced jQuery Validation**

In the easy way, we called $("#testForm").validate() without any arguments.

You can also pass in an object that specifies the rules and error messages. The object that you pass in has two properties: “rules” and “messages”. Each of theses is itself an object.

Within the rules object, you have a bunch of properties – each property has a name which is the same as the ID of the field that you want to be validated, and a value which is yet another object. This value object has a name for each validation rule that you want to apply (required, digits, etc.) and a value of true for those rules…

{  
 rules: {

name: {

required: true,

minlength: 2

},

age: {

required: true,

digits: true

}

},

messages: {

name: {

required: "Please enter a name"

},

age: {

required: "You must specify an age",

digits: "Age must be a nonnegative integer"

}

}

}

Besides validation rules with a value of true, there are some additional rules that we can use:

* minlength – field must have at least this many characters (example – minlength: 2)
* maxlength
* rangelength – specifies both minlength and maxlength in one rule  
  (example: rangelength: [3, 6])
* min – specifies a minimum numeric value
* max
* range – combines min and max (example – range: [13, 17])
* equalTo – this field must have a value equal to another field   
  (example – equalTo: "#password")

As shown above, we can also have custom error messages. The format of the messages object is the same as the rules object, but rather than giving a value, you give an error message that you want displayed for that particular rule.

**Custom Validation Rules**

We can add a validation rule by calling the validator’s addMethod method. The example below adds a rule named “nameChars”, specifies a function called nameTester to actually do the validation for us, and specifies a default error message:

jQuery.validator.addMethod("nameChars", nameTester,

"A name can only contain letters or numbers");

Our validation function usually takes two parameters: a value to be tested, and an input element that we are testing. It returns true if the value is valid and false otherwise.

Validation functions generally follow the same template:

function nameTester( value, elem ) {  
 return this.optional(elem) || *put your own test here*;  
 }

Often, we’ll compare it against a regular expression. The following function uses RegExp’s test() function, which returns true if it matches and false if not, to make sure that the name is made up of letters and numbers:

function nameTester( value, elem ) {  
 return this.optional(elem) || **/^[A-Za-z0-9]+$/.test(value)**;  
 }

Often, we don’t create a named function; we can just use an anonymous function in our call to addMethod:

jQuery.validator.addMethod("nameChars", function( value, elem ) {  
 return this.optional(elem) || **/^[A-Za-z0-9]+$/.test(value)**;  
 }, "A name can only contain letters or numbers");

Once you have your custom rule defined, you add your own rule, like:

$("#testForm").validate( {  
 rules: {

name: {

required: true,

nameChars: true

}, *etc.*

**Practice:** Add the nameChars rule to your form.

**Extended practice:** Add a validation rule to check for a valid CST id in the format "cst###" where ### are valid digits.

**Practice:** Go to **jqueryvalidateform-done.html** and add rule such that the new password must contain at least one lowercase letter, one uppercase letter, and one digit; and the confirm password field must match the new password field.

**Alternatives to custom validation**

If you are just testing against a regular expression, you could use the **pattern** attribute. Unfortunately, the **pattern** attribute doesn’t play well with *just* the jQuery validation plugin.

You can add the **pattern** method to the jQuery validation settings – just add **pattern.js** or **additional-methods.js** to your page, and then you can add the **pattern** to the rules as follows (or just use the **pattern** attribute):

pattern: /*regularExpression*/

**Alternative way of doing form validation: the Constraint Validation API**

Google “constraint validation API” to get to:

<https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint_validation>

<https://developer.mozilla.org/en-US/docs/Web/API/Constraint_validation>

The Constraint Validation API allows you to alter the built-in HTML5 validation rules and messages so that you can customize them.

See **constraint\_validation\_api-start.html**, **constraint\_validation\_api-done.html**, and **scripts/constraint\_validation\_api.js**

# Learning Step 5.5: Alter form field contents based on input from users

You can also change forms based on user input – for instance, if a user selects “Other” from a group of radio buttons, you could have a text field to allow to describe what “other” means appear or become enabled.

Or you could change dropdown list entries based on user input.

Another simple example is adding a message to indicate the number of characters available/used when filling in a text area.

See **last-form-example.html**, **last-form-example-working.html**, and **last-form-example-done.html**.