Week 7 Lab – Checking Form Data with JavaScript and Transferring data between HTML pages

Aims:

- Gain experience in using JavaScript functions
- Use selection and repetition statements to control the execution of a script
- Use JavaScript in-built functions and regular expressions to check entered form data
- Create and apply data consistency rules using JavaScript
- Transfer data between HTML pages using client-side Web Storage

Only Task 1 is marked. We encourage that you complete Task 2 as it is helpful to your assignment part 2.

Task 1: Using JavaScript to check user entered data

While we can check input data using the HTML5 pattern attrbte, JavaScript enables us to do ***more complex data checking*** that involves more than one input field.

JavaScript can be used to create customised error messages.
In this exercise we want to display the error messages in a single alert message.
(As a challenge you might like to try displaying the messages within the html page)

Step 1: Load an example HTML form.

1. Create a local folder called **`lab07`**, download the **`lab07.zip`** file and unzip the file to the folder. Open the file **`register.html`** in your editor.
2. We need to tell the web page **`register.html`** where to submit the information collected from the form when the form is submitted. We will submit with **`post`** to a testing page **`formtest.php`**.
   Add the following **attributes** to the **`<form>`** element

```
method="post" action="https://mercury.swin.edu.au/it000000/formtest.php"
```

3. Check that the form input has been correctly sent to the server by checking that the response from the server has a table with all the **name-value** pairs sent from the form control elements.
4. So we can check the JavaScript data validation is working we will turn off HTML5 data validation by adding the **`novalidate="novalidate"`** attribute to the form.
5. Following the steps in Lab 6, use the **`<script>`** element to create a link to a file called **`validate.js`** from **`register.html`**.
6. ***Validate the HTML.***

Step 2: Create JavaScript template to check the data a user has entered into the form

In this step we will create a JavaScript program that registers a function to respond to submit events of the HTML form.

1. Following the template defined in Lab 6 (T1 step 3), create a JavaScript file called **`validate.js`**. Include an **`init`** function that responds to the browser window loading.

2. At the top of your JavaScript file, insert the directive `"use strict";` to prevent any global variables being declared within functions.

3. Inside the `init` function write the following lines to get a reference to the HTML form and register a function called `validate()` to respond to `onsubmit` events of the HTML form.

```
var regForm = document.getElementById("regform");// get ref to the HTML element
regForm.onsubmit = validate;            //register the event listener
```

4. Just below the "`use strict`"; directive at the top of the file, create the shell of the `validate()` function as shown below. If this function returns false then the form data will not be submitted to the server

```
/*get variables from form and check rules*/
function validate(){
     // initialize local variables
     var errMsg = "";                                  // stores the error message
     var result = true;                                // assumes no errors

     //get variables from form and check rules here
     // if something is wrong set result = false, and concatenate error message
     //...

     if (errMsg != ""){    //only display message box if there is something to show
       alert(errMsg);
     }

     return result;    //if false the information will not be sent to the server
}
```

We need to write some data validation rules here.

Reload the HTML and check there are no errors in the JavaScript (Use Firefox menu --> Web Developer -- > Web console).

Step 3: Use regular expressions to check the format of text input strings

In this step we will use regular expressions within `if-else` statements to check the following rules:

- First name is not empty and contains only alpha characters.
- Last name is not empty and contains only alpha characters or a hyphen.
- The age is an integer greater than 18 but less than 10,000.

Regular expression patterns in JavaScript are the same as those we used in HTML5 but are objects defined between two forward slashes:

```
var myPattern = /regex pattern/;
```

We test strings against regex objects using the `Regex.test(String)` or the `String.match(Regex)` methods.

1. Within the `validate()` function you created in Step 2, after the section where you initialize the variables, write the following statements

We need to get the *value* of the HTML input object

```
     var firstname = document.getElementById("firstname").value;

     if (!firstname.match(/^[a-zA-Z]+$/)){
       errMsg = errMsg + "Your first name must only contain alpha characters\n"
       result = false;
     }
```

2. Last names can contain alpha characters or hyphens. Following the example above, write an **if** statement that will check the last name.

Hint: it often makes the code easier to read if you declare all your local variables at the start of the function rather than scatter them throughout the function.

Step 4: Use global functions to check the data *type* of input strings

For numeric input we can use regular expressions to check that only digits are entered. We can also check the *type* of input is numeric using JavaScript *global functions* like `isFinite()` or `isNaN()`.

1. Using the same approach as above, in the ***`validate()`*** function get the ***value*** the user has typed into the age text box and store it in a local variable called **`age`**. Test the input is numeric as follows:

```
if (isNaN(age)){
        errMsg = errMsg + "Your age must be a number\n"
        result = false;
}
```

2. Let's extend this **`if`** statement with an **`else if`** structure to check the age is between 18 and less than 10,000.

```
else if (age < 18){
        errMsg = errMsg + "Your age must be 18 or older\n";
        result = false;
}
else if (age >= 10000){
        ..........................................................
        ..........................................................
}
```

Write the appropriate code here…

3. Write a check to ensure the number of travellers is between 1 and 100.

Reload the HTML and check there are no errors in the JavaScript. Test with a range of valid and invalid data inputs.

Step 5: Check options have been selected

We need to ensure that the user has selected a species (radio button), one or more trips (checkbox) and a menu preference (select drop-down list). Unless we have a default selection we need to use JavaScript to do this. Let's start with the Meal preferences drop-down list. All we need to do is check if the user has changed the selection to something other than "Please select" that has value "**`none`**".

1. Add the following test to the **`validate()`** function to make sure none is not selected:

Notice we can use the method call within the **if** condition expression. We don't have to explicitly create a var.

```
if(document.getElementById("food").value == "none"){
        errMsg = errMsg + "You must select a food preference";
        result = false;
}
```

2. To check if radio buttons or checkboxes have been selected we use the **`.checked`** property which can be **`true`** or **`false`**. Enter the following code into your **`validate()`** function to ensure at least one trip option is selected from the checkboxes.

Boolean variables (true or false)

Compound condition

```
var is1day = document.getElementById("1day").checked;
var is4day = document.getElementById("4day").checked;
var is10day = document.getElementById("10day").checked;

/*  at least one trip selected */
if (!(is1day || is4day || is10day)) {
        errMsg += "Please select at least one trip.\n";
        result = false;
}
```

same as `errMsg = errMsg + "Please ....";`

3. Use a similar approach to Step 5.2 above to ensure at least one Species radio button is selected. Append an appropriate error message to the **`errMsg`** variable if none is selected.

Test the program to ensure all the data rules are being properly checked.

Step 6: Checking rules for the consistency of input data

More complex conditions that check across data inputs can also be implemented in JavaScript. For example: We want to check the age of applicants but the age range of applicants varies according to species according to the following rules:

a. Humans can live to 120
b. Dwarfs and Hobbits can live to 150
c. Elves are immortal

In this step we will use **function calls, arrays** and the **switch** structure to achieve this.

1. First let's write a function that returns the species of the applicant as a String. Rather than check each **`input`** element individually by its **`id`**, we will use the **`getElementsByTagName`** method to get an **array** of all the input elements within (*are descendants of*) the **`species`** fieldset object. We will then iterate through the array, and if we find an element that has been **checked** we will return its value.

```
/*return the species selected as a string*/
function getSpecies() {
      //initialise variable in case does not get reinitialised properly
      var speciesName = "Unknown";

       //get an array of all input elements inside the fieldset element with id="species"
      var speciesArray = document.getElementById("species").getElementsByTagName("input");

       for(var i = 0; i < speciesArray.length; i++){
       if (speciesArray[i].checked)     // test if radio button is selected
              speciesName = speciesArray[i].value; // assign the value attribute
      }
      return speciesName;
}
```

Note: We could reuse this function to check if a `species` radio button has been clicked – rather than use the approach described in Step 5.3. We just need to check that the function returns "Unknown".

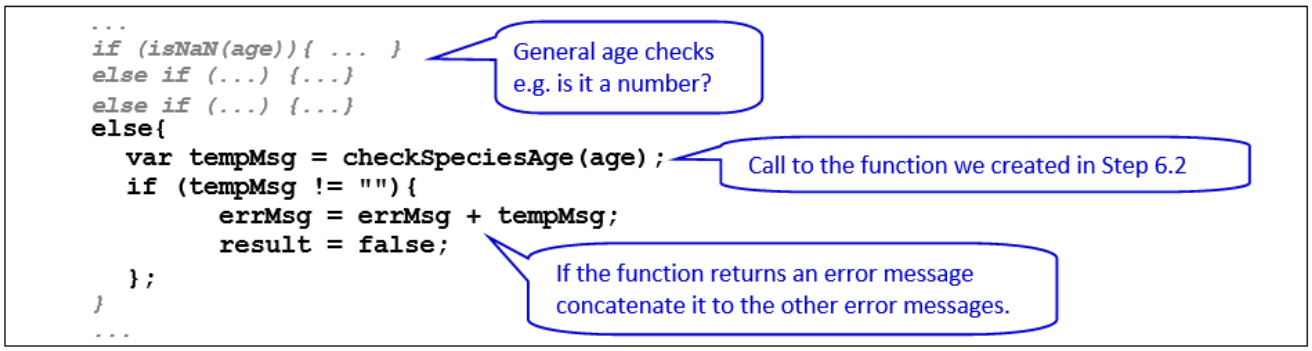
2. We will use this **`getSpecies()`** function to check that the above rules related to age range of the species. This code illustrates the use of the **`switch`** statement. We will encapsulate this functionality into a function called **`checkSpeciesAge()`**.

Note the parameter **`age`** is a variable local to this function.

Call to the function we created in the previous sub-step.

```
/*if rule violated return error message*/
function checkSpeciesAge(age){
      //assume the parameter age has already been checked for general constraints e.g >18
      var errMsg = "";
      var species = getSpecies();
      switch(species){
        case "Human":
              if (age > 120) {
                    errMsg = "You cannot be a Human and over 120.\n";
              }
              break;
        case "Dwarf":        //note no break so next case will be selected if dwarf
        case "Hobbit":
              if (age > 150){
                    errMsg = "You cannot be a " + species + " and over 150.\n";
              }
              break;
        case "Elf": //elves can be any age so no error possible
              break;
        default:
              errMsg = "We don't allow your kind on our tours.\n";
      }
        return errMsg;
}
```

3. We now need to invoke the above **`checkSpeciesAge(age)`** function. We will call it from the **`validate()`** function by adding it at the end of the **`if else`** age check we created in Step 4 above.



Test the program to ensure all the data rules are being properly checked.

4. Using the approach above, create rules that check species against beard length:
   a. Humans may or may not have a beard
   b. Dwarfs over 30 years old always have a beard longer than 12 inches.
   c. Elves and Hobbits never have beards.

   Test the program to ensure all the data rules are being properly checked.

.

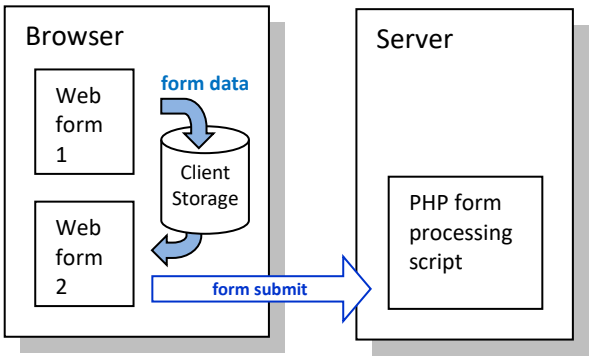
Once you have completed the above tasks, show them to your tutor

Task 2: Transferring data between HTML pages using client Web Storage *(Optional)*

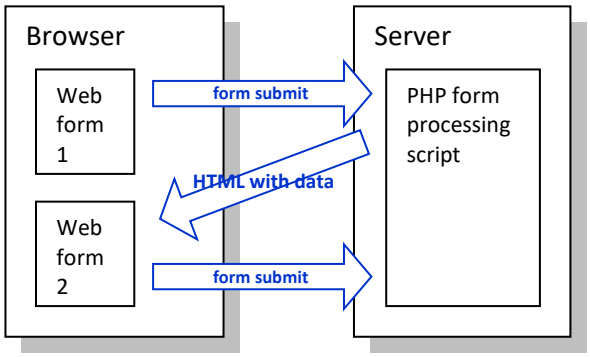
We encourage that you complete Task 2 as it is helpful to your assignment part 2.

Sometimes we want to transfer information entered in a form on one Web page to another Web page. For example, for the Trip Booking form we developed in Task 1, we want to display a confirmation page for the user to accept before the data is submitted to the Web server.

There are a couple of methods for doing this.



Method 1 – Using Client Storage



Method 2 – Server-side processing

Method 1 transfers data the user has entered into their Form 1 via local client storage which is managed by the browser. This data set is then used to initialise the second form when it loads. After the user has checked it (possible provide extra information) this second form is then submitted to the server.

In Method 2, the user data set is submitted by Form 1 to a PHP script on the server. The script is then used to create a new HTML form page that incorporates the data the user sent to the server. This second form is then completed by the user and submitted to the server.

We will use Method 1 in this task. HTML5 introduces new methods to store data locally within the user's browser. In the past this has been done with cookies. However, HTML5 Web Storage is faster and more secure (see http://www.w3.org/TR/webstorage/). The data are not included with every server request, but used only when asked for. It is also possible to store large amounts of data, without affecting the website's performance. If the client does not support Web Storage, we may need to use Cookies to maintain 'state' between stateless Web pages.

In Web Storage data are stored in *key/value* pairs. There are two new objects for storing data on the client:

- **localStorage** - stores data with no expiration date
- **sessionStorage** - stores data for one session (while the browser window /tab is open)

In this Task we will use **sessionStorage**. The way to store a key/value pair is:

```
sessionStorage.key = value;
```

For example, to store the value of an HTML element with `id = username` in JavaScript we could say:

```
sessionStorage.uname = document.getElementById("username").value;
```

To retrieve a value from sessionStorage (or localStorage) and store it in a JavaScript variable we write:

```
value = sessionStorage.key;
```

For example:

```
var username = sessionStorage.uname
```

As explained above, we will send data from **register.html** to the Web page **confirm.html** via **sessionStorage**. The basic procedure is:

1. Set the form on **register.html** to hyperlink to **confirm.html.**
2. After the data on the **register.html** form have been successfully validated, write the value of the input fields or selected options to **sessionStorage**.
3. When the **confirm.html** window loads, read the values from **sessionStorage** and write these to the HTML using JavaScript.
4. The data can be written as text onto the HTML page, or can be set as values for input elements in a form. Data need to be written to form inputs If you want to forward these values onto the server when the user submits the form in **confirm.html**. These input elements can be hidden from the user if required.

Step 1: Modify the form on register.html so data is not directly sent to the server

From the **lab07.zip** file make sure you have the file **confirm.html** in your **lab07** folder.

In **register.html**, change the action attribute of the form so that it hyperlinks to **confirm.html** rather than sends a post message to the test PHP script on the server.

```
<form id="regform" action="confirm.html">
```

This hyperlink will only execute if the **validate()** function returns true.

Step 2: Storing user data

In this step will we write a function that stores the values the user has entered into the form in **sessionStorage**. We will call this function at the end of the **validate()** function we wrote in Task 1.

1. In **validate.js**, create a function called **storeBooking()** as shown below. As we already have variables for `firstname, lastname, age, species,is1day, is4day, is10day` in the **validate**() function, we will pass these to the function as parameters. As the trip was a checkbox with multiple options, we will concatenate the select option and store then in a single string.

```
function storeBooking(firstname, lastname, age, species,is1day, is4day, is10day){
      //get values and assign them to a sessionStorage attribute.
      //we use the same name for the attribute and the element id to avoid confusion
      var trip = "";
      if(is1day) trip = "1day";
      if(is4day) trip += ", 4day";
      if(is10day) trip += ", 10day";
      sessionStorage.trip = trip;
      sessionStorage.firstname = firstname;

      //add other elements here

      alert ("Trip stored: " + sessionStorage.trip); //added for testing
}
```

Concatenate the selections from the Trip checkboxes into a single string

Remove this once tested

2. Note the above concatenation of strings can produce a messy result such as “, 10 day “ if they only select one 10 day trip. Can you think of a better way to do this?

3. Extend the function to store the rest of the user information input in `sessionStorage` – that is **lastname, age, species, trip, food** and **partySize**.
   Note: **food** and **partySize** are not passed to the function as parameters. We will need to get these values from the DOM.

4. Invoke the **storeBooking()** function from the **validate()** function as shown below. We only want to do this if the validation returns **result** returns true.

```
function validate(){

        /* code from Task 1
        ...
        */

        if (result){       //if the form validates ok.
        //As result is a Boolean variable saying (result) is the same as saying (result == true)
              storeBooking(firstname, lastname, age, species,is1day, is4day, is10day)
        }

        /* code from Task 1 */
        return result;     //if false the information will not be sent to the server
}
```

Insert this code before the return statement

5. Load **register.html** and run. You can use a temporary alert box as shown above to test if values are being correctly stored in **sessionStorage**.

Step 3: Create / modify a Web page to accept data

A partially complete HTML file called **confirm.html** has been provided in **lab07.zip.**

1. In the header of **confirm.html**, create a reference to the JavaScript file **confirm.js**

Note that **confirm.html** has a number of paragraphs containing **** elements. We will write the read-only data to be confirmed into these span elements. Also note there are a number of **hidden** input elements. Hidden input can store data in a form on the Web page without being seen by the user. We will use these fields to transfer data to the server when the form is submitted. For example:

```
<input type="hidden" name="firstname" id="firstname" />
```

2. Complete the form by creating hidden input elements for the following data in sessionStorage:
   lastname, age, species, trip, food, partySize and cost.

The cost is not stored but will be calculated from the cost of the *trips booked* x *number of participants*.

Step 4: Write the data to the Web page from sessionStorage

A partially complete JavaScript file called **confirm.js** has been provided in **lab07.zip.**

Now the values from the register form are in **sessionStorage,** we can write them to **confirm.html** when the window loads. To do this we will call a function called **getBooking()** from the **init()** function.

1. A partial complete **getBooking()** function is provided. Complete this function by writing the data to the web pages from sessionStorage:
   lastname, age, species, trip, food, partySize and cost.

2. Invoke this **getBooking()** from the **init()** function.

Step 5: Create a Cancel Button

We need to give the user the ability to cancel the booking from the confirmation page.

1. Create the following function in **`confirm.js`**. When invoked, this will redirect the browser to the specified Web page.

```
function cancelBooking(){
        window.location = "register.html";
}
```

2. Create a listener in the **`init()`** function that will respond to an **`onclick`** event on the button with **`id="cancelButton"`** in the **`confirm.html`**.

Step 6: Prefill the form with stored data in sessionStorage

If the user has previously visited the site with their browser we may want the browser to automatically fill in their personal details (let's assume no one else is using the same browser session – probably a bad assumption!). To do this when the page loads we will check if data exist in **sessionStorage**, and if so we will load it into the form.
This is just the reverse process of storing the data.

1. Create a function called **`prefill_form()`** in **`validate.js`** as below:

```
/* check if session data on user exists and if so prefill the form*/
function prefill_form(){
     if(sessionStorage.firstname != undefined){   //if storage for username is not empty
          document.getElementById("firstname ").value = sessionStorage.firstname;
          /* Write rest of values to the form here
          ....
          */

          switch(localStorage.species){
               case "Human":
                    document.getElementById("human").checked = true;
                    break;
               case "Dwarf":
                    document.getElementById("dwarf").checked = true;
                    break;
               case "Hobbit":
                    document.getElementById("hobbit").checked = true;
                    break;
               case "Elf":
                    document.getElementById("elf").checked = true;
                    break;
          }
     }
}
```

Writing to checkboxes takes a bit more work!

2. Call the **`prefill_form()`** function from within the **`init()`** function which is triggered when the window.onload event fires.
3. Run and test that the output is as expected.
4. 'Reset' the page and then reload it. Do the values preload from sessionStorage?

Test your Web site program to ensure all the data entered into is being correctly being displayed and sent to the server when the submit button is clicked .