|  |  |  |
| --- | --- | --- |
| LAB 16 (chapter 9) JavaScript 2: DOM and Events  |  | | --- | | What You Will Learn   * Creating event listeners to react to events * Tools and tricks to help you develop JavaScript * Working with form validation |  |  | | --- | | Approximate Time  The exercises in this lab should take approximately 45 minutes to complete. | |
| Web Application Development  COP3834  Professor Navarro |
| Textbook by Pearson  http://www.funwebdev.com  Adapted from Pearson, modified 10/16/21 |

## THE DOCUMENT OBJECT MODEL (DOM)

|  |
| --- |
| Preparing Directories |
| 1 | Download zip folder.. |
| 2 | Uncompress to your desktop. |

The document.write() function as a way to manipulate markup using JavaScript. While fine from a learning JavaScript perspective, it is generally not how one typically uses JavaScript. In this lab, you will learn how to use the Document Object Model (DOM) with JavaScript as a “better” way of manipulating content.

## Event Handling

One of the most common uses of JavaScript is to use it to respond to different user-initiated actions. These are generally referred to as **events** and the JavaScript that responds to these events are called **event handlers**. As the textbook indicates, there are several ways of writing event handlers in JavaScript. In this lab, we stick with the preferred event listener approach.

|  |
| --- |
| EXERCISE 9.4 — Simple Event Handling |
| 1 | Open lab09-walkthrough04.html and test in browser. |
| 2 | Open lab09-walkthrough04.js, add the following, and then test.  /\* code goes here \*/  function simpleHandler(event) {  alert("button was clicked");  }  var btn = document.getElementById("testButton");  btn.addEventListener("click", simpleHandler);  This defines an event handler function and assigns it to the click event of the button. It is quite common to instead use an anonymous function for the event handler, as shown in the next steps. |
| 3 | Add the following code and test.  var img = document.getElementById("mainImage");  /\* changes the style of the image when it is moused over \*/  img.addEventListener("mouseover", function (event) {  img.className = "makeItGray";  });  This changes the class of the image when the mouse moves over the image. We now need to remove the class styling when the mouse leaves the image. |
| 4 | Add the following code and test.  /\* remove the styling when mouse leaves image \*/  img.addEventListener("mouseout", function (event) {  img.className = "makeItNormal";  }); |
| 5 | Change the simpleHandler() function as follows and test.  function simpleHandler(event) {  var content = document.getElementById("content");  if (btn.innerHTML == "Hide") {  content.style.display = "none";  btn.innerHTML = "Show";  }  else {  content.style.display = "block";  btn.innerHTML = "Hide";  }  }  This makes the button toggle the visibility of the content paragraph. |
| 6 | Instead of simply hiding/showing the paragraph, we will make use of CSS3 transitions as well as the JavaScript delay function. Change your code to the following and test.  function simpleHandler(event) {  var content = document.getElementById("content");  if (btn.innerHTML == "Hide") {  btn.innerHTML = "Show";  content.className = "makeItDisappear";  // change the display mode after a 1000 millisecond delay  setTimeout(function(){  content.style.display = "none";  },1000);  }  else {  btn.innerHTML = "Hide";  content.style.display = "block";  // change the class after a 500 millisecond delay  setTimeout(function(){  content.className = "makeItNormal";  },500);  }  } |

|  |
| --- |
| EXERCISE 9.5 — Responding to Load Events |
| 1 | Open lab09-walkthrough05.html and test in browser. |
| 2 | Open lab09-walkthrough05.js and add the following.  var divToGet = document.getElementById("div1");  alert(divToGet.innerHTML); |
| 3 | Display the JavaScript console in your browser and then display Open lab09-walkthrough05.js.  If we save and refresh our page we might expect the alert to say "This is div 1" since that is what is contained within the div with id=div1. However, nothing is alerted, and we get an error (the message shown is google chrome). |
| 4 | The reason for the above error is that the script is executed **before** the DOM is fully loaded. To make the script execute **after** the DOM is loaded we must make use of the onload event. Try this yourself by moving the code inside of a listener for the onload event as follows (and then test):  **window.addEventListener("load", function(){**  var divToGet = document.getElementById("div1");  alert(divToGet.innerHTML);  });  Tada |

|  |
| --- |
| EXERCISE 9.6 — Debugging Events |
| 1 | Open lab09-walkthrough05.html and display the JavaScript console. |
| 2 | In Chrome, to access the debugger, you have to click on the Sources tab within the Console and then open the JavaScript file that you wish to debug.  You will now add a breakpoint to your script by clicking to the left of the line that contains the call to alert(). |
| 3 | Try refreshing the page. The line with the breakpoint will be highlighted as shown in Figure 9.2. You can now examine the state of local and global variables. |

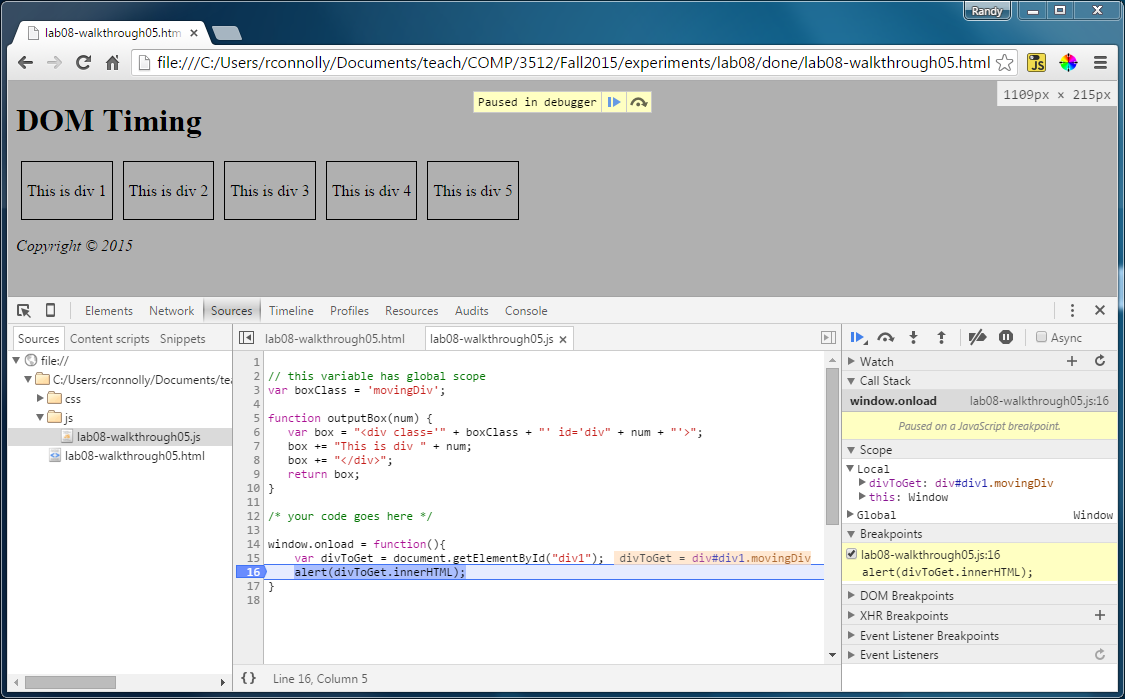


Figure 9.2 – Chrome Debugging

|  |  |
| --- | --- |
| 4 | In FireFox, to access the debugger, you have to click on the Debugger tab within the Developer tools and then open the JavaScript file that you wish to debug.  You can add a breakpoint to your script by **clicking to the left** of the line that contains the call to alert(). |
| 5 | Try refreshing the page. The line with the breakpoint will be highlighted as shown in Figure 9.3. You can now examine the state of local and global variables. |

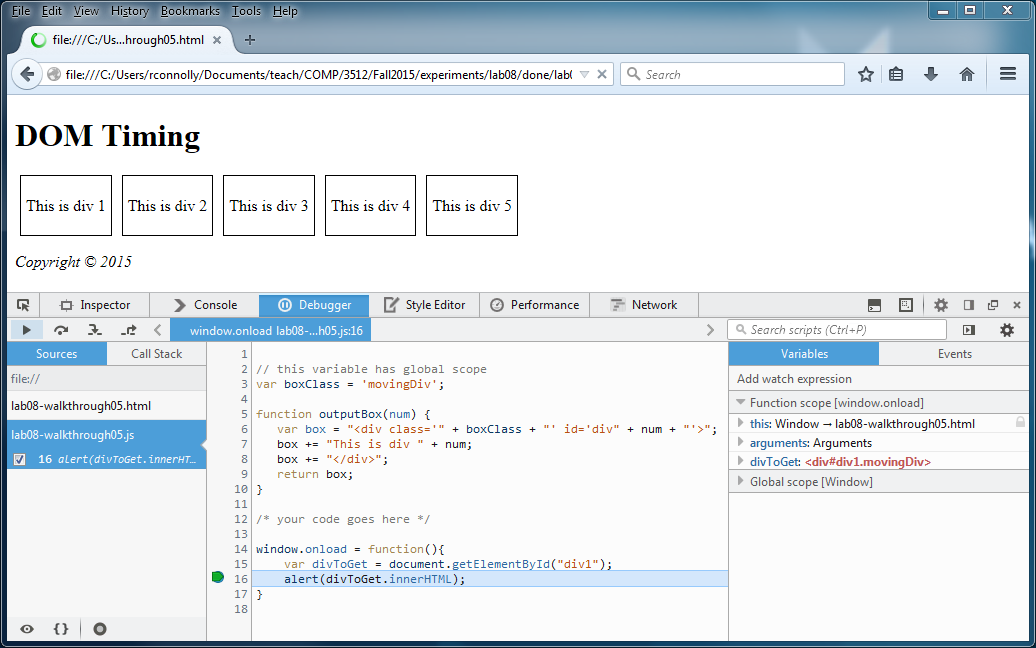
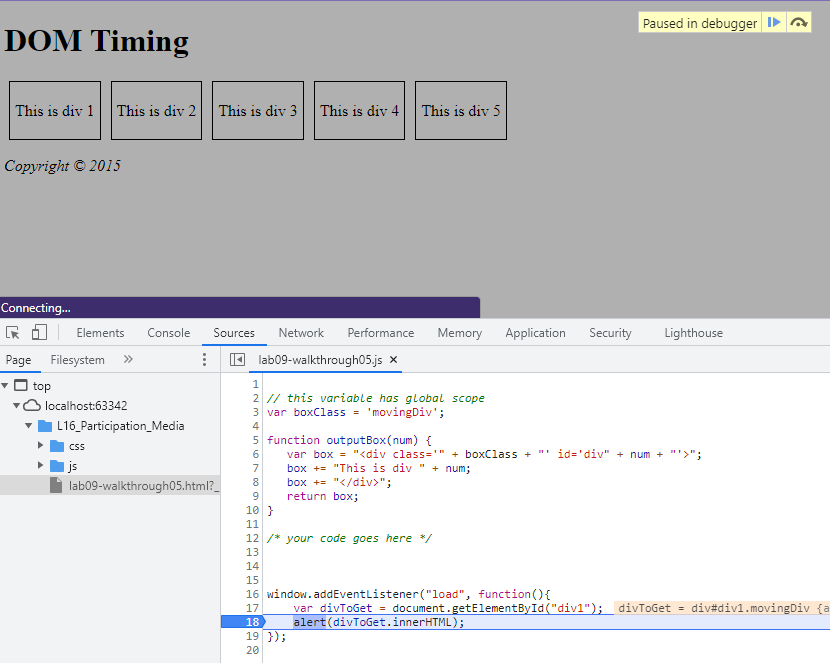


Figure 9.3 – FireFox Debugging

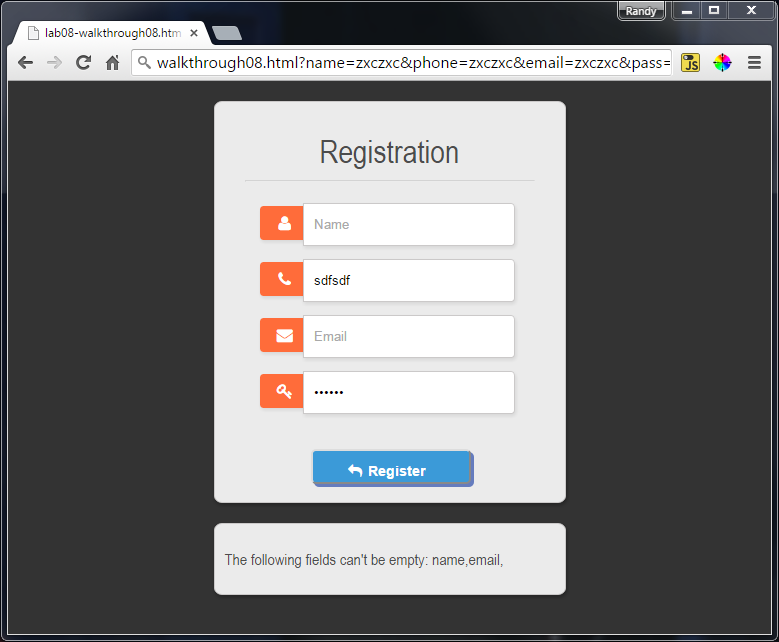


Chrome debugging

|  |  |
| --- | --- |
| 6 | Once you have set a breakpoint, you can resume execution of the script or continue execution line-by-line using the execution buttons: |
| 7 | Edit lab09-walkthrough05.js and edit the code as follows:  window.addEventListener("load", function(){  var divs = document.querySelectorAll(".movingDiv");  for (i=0; i<divs.length; i++)  {  divs[i].addEventListener("mouseover", function (e) {  alert("triggered by " + e.target.id);  });  }  }); |
| 8 | Test in browser by moving your mouse over the different <div> boxes. |
| 9 | Use the debugger and set a watchpoint before the alert() call. Examine the contents of the e variable and e.target property. |

|  |
| --- |
| EXERCISE 9.7 — Working With Forms |
| 1 | Open lab09-walkthrough07.html and test in browser. |
| 2 | Add the following function to lab09-walkthrough07.js:  function setBackground(e) {  **if (e.type == "focus") {**  **e.target.style.backgroundColor = "#FFE393";**  **}**  **else if (e.type == "blur") {**  **e.target.style.backgroundColor = "white";**  **}**  **}**  This function is going to get called every time the focus or blur events are triggered in one of our form’s input elements. |
| 3 | Add the following code:  window.addEventListener("load", function(){  var cssSelector = "input[type=text],input[type=password]";  var fields = document.querySelectorAll(cssSelector);  for (i=0; i<fields.length; i++)  {  fields[i].addEventListener("focus", setBackground);  fields[i].addEventListener("blur", setBackground);  }  });  This assigns an anonymous function to the load event of the browser. The function assigns the setBackground() to the blur and focus events of the relevant <input> elements. |
| 4 | Test in browser. Tab between the different elements. |

|  |
| --- |
| EXERCISE 9.8 — Form Validation |
| 1 | Examine lab09-walkthrough08.html and then add the following code to lab09-walkthrough08.js.  /\* responsible for setting up event listeners on page \*/  function init() {  document.getElementById("sampleForm").addEventListener("submit",  checkForEmptyFields);  }  /\* initialize handlers once page is ready \*/  window.addEventListener("load", init);  /\* ensures form fields are not empty \*/  function checkForEmptyFields(e) {  var cssSelector = "input[type=text],input[type=password]";  var fields = document.querySelectorAll(cssSelector);  // loop thru the input elements looking for empty values  var fieldList = [];  for (i=0; i<fields.length; i++) {  if (fields[i].value == null || fields[i].value == "") {  // since a field is empty prevent the form submission  e.preventDefault();  fieldList.push(fields[i]);  }  }  // now set up the error message  var msg = "The following fields can't be empty: ";  if (fieldList.length > 0) {  for (i=0; i<fieldList.length; i++) {  msg += fieldList[i].id + ",";  }  alert(msg);  }  } |
| 2 | Test in browser. Experiment by filling in different fields. |
| 3 | Modify the checkforEmptyFields() function by adding the following:  function checkForEmptyFields(e) {  // hide the error message element  var errorArea = document.getElementById("errors");  errorArea.className = "hidden";    var cssSelector = "input[type=text],input[type=password]";  var fields = document.querySelectorAll(cssSelector);  // loop thru the input elements looking for empty values  var fieldList = [];  for (i=0; i<fields.length; i++) {  if (fields[i].value == null || fields[i].value == "") {  // since a field is empty prevent the form submission  e.preventDefault();  fieldList.push(fields[i]);  }  }  // now set up the error message  var msg = "The following fields can't be empty: ";  if (fieldList.length > 0) {  for (i=0; i<fieldList.length; i++) {  msg += fieldList[i].id + ",";  }  errorArea.innerHTML = "<p>" + msg + "</p>";  errorArea.className = "visible";  }  }  Instead of displaying the error message inside an alert box this places it within a <div> element. The result should look similar to that shown in Figure 9.4. |



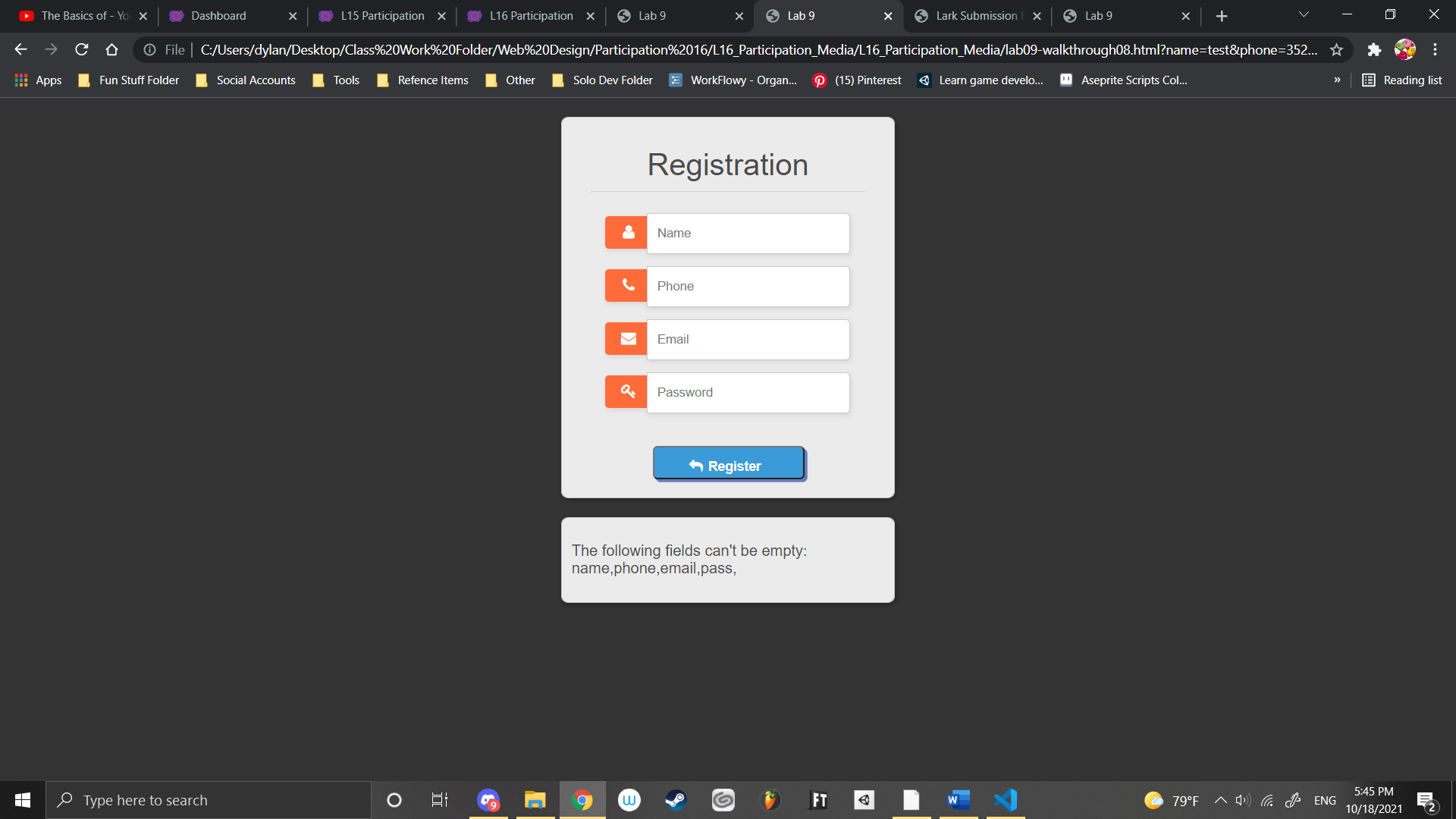


Figure 9.4 – Finished Exercise 9.8

2. Use HTML5 to add phone number validation. Inside the lab09-walkthrough08-html file find the form element called

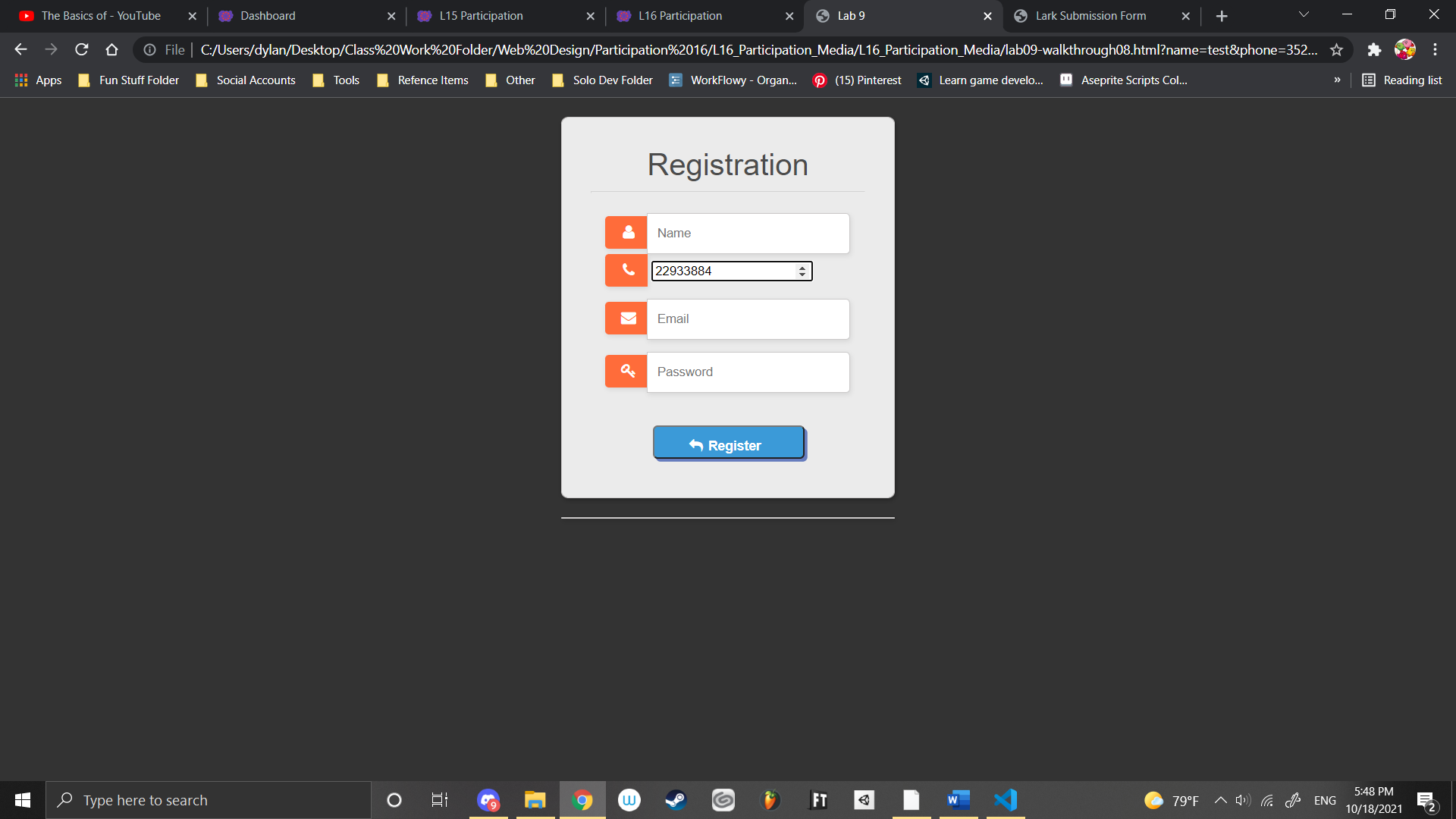
<input type="text" name="phone"

Change it to

<input type="number" name="phone" pattern=”[0-9]\*”

The pattern attribute will now only allow numeric characters of 0-9 in any amount. Test the phone field with non-numeric characters.

Can’t type them



3. Add email validation with test for regular expression match between / and /. Add the follow code after the function init()

/\*validate email \*/

function checkEmail() {

var email = document.getElementById('email');

var filter = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)\*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;

if (!filter.test(email.value)) {

alert('Please provide a valid email address');

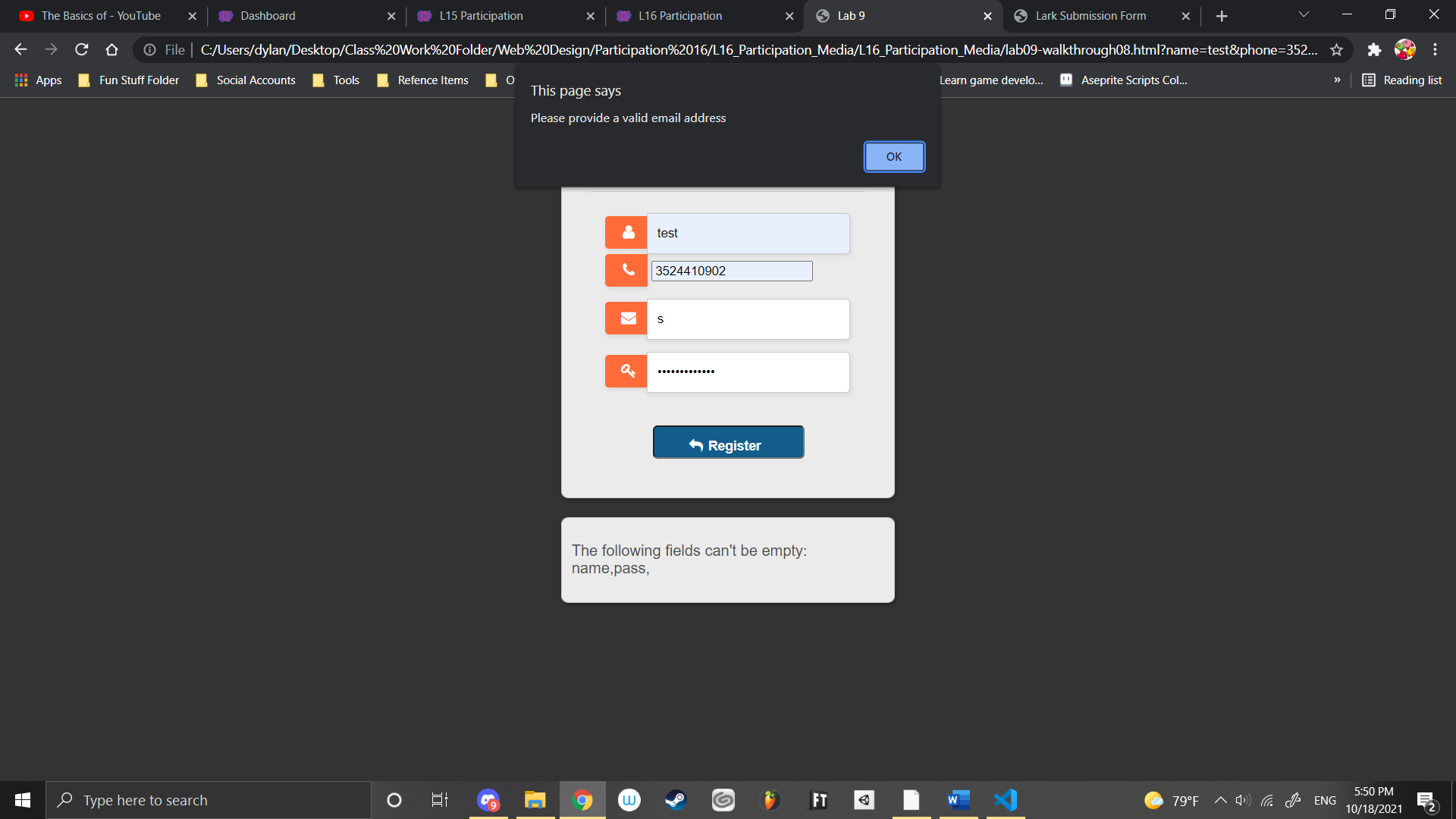
email.focus;

return false;

}

}

Then, add checkEmail(); to the function checkForEmptyFields(e)



|  |
| --- |
| EXERCISE 9.9 — Validation Errors |

1. Open lark-form.html in a browser or IDE.
2. The form html is completed but it doesn’t have any validation yet. Open js/lark-form.js to add the validation code and make corrections.
3. Add listeners for classes with highlightable. This will represent form elements that are assigned to change (toggle) color if they have focus.

In the bracket for window.addEventListener("load", function() { , add the following listeners:

var hilightableInputs = ***document***.querySelectorAll("hilightable");  
***console***.log(hilightableInputs);  
for (var i=0; i < hilightableInputs.length; i++) {  
 hilightableInputs[i].addEventListener("focus", function(e) {  
 e.target.classList.toggle("highlight");   
 });  
 hilightableInputs[i].addEventListener("blur", function(e) {  
 e.target.classList.toggle("highlight");  
 });  
}

1. After adding the listeners, you don’t see input fields toggling color but there are no errors in the console. However, the console output should be showing an array of elements that have the style class “highlightable”. Think about how classes are formatted on the style sheet and correct the error.
2. Add listeners for classes with required... This will represent form elements that have the “.required” class as a style. The eventlistener is monitoring a change event that differs in operation depending on the form input type. Read more at <https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/change_event>

Add this in window.addEventListener("load", function() { and after the “add listeners for classes with highlightable”.

var requiredInputs = ***document***.querySelectorAll(".required");  
***console***.log(requiredInputs );  
for (var i=0; i < requiredInputs.length;i++) {  
 requiredInputs[i].addEventListener("change", function(e) {  
 makeClean(e.target);  
 });  
}

1. On submitting the form, empty checks are performed on required inputs. Add this code in the window.addEventListener("load", function() { after after “Add listeners for classes with required…”

var mainForm = ***document***.getElementById("mainForm");  
mainForm.addEventListener("submit", function(e) {  
  
 var requiredInputs = ***document***.querySelectorAll(".required");  
 for (var i=0; i < requiredInputs.length; i++){  
 if( isBlank(requiredInputs[1]) ){  
 e.preventDefault();  
 requiredInputs[1].classList.add("error");;  
 }  
 else {  
 makeClean(requiredInputs[i]);  
 }  
 }  
});

This block should add an error class style to all of the “requiredInputs”. However, if you click submit, it only adds an error style to the 2nd form input that is using .required class. Input Description has the red error, but not the other inputs that are also required. Icon

Description automatically generated

Find and correct 2 errors in this block so that all required inputs will display in red if empty.

**Submission: Upload corrected files in zipped directory to Canvas.**