**blackCSIS210 - Data Structures**

Web App. Dev.

### Laboratory 4

**Lab 1**

# Names \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 

# General Lab Procedures

* You should create a directory (folder) in your home account called csis390. At the beginning of each lab, create a new sub-directory called labX, where X is the lab number.
* Files used in the lab can be found on the course canvas webpages.
* Turn in this lab sheet stapled to print outs of the code you produce as needed in each assigned section from the laboratory manual. These sheets should be in order. One lab submission is sufficient for each group.
* You can find documentation for the HTML 5 specification, the CSS, and the Javascript references respectively at

<https://html.spec.whatwg.org/>

<https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>

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

**Lab Objectives**

* Practice using username/password verification
* Practice creating and verifying forms using Javascript

## Lab 4

#### Part 1 – Username and Password verification

In this part of the lab, you will create an HTML webpage that allows a user to create a username followed by a password, based upon certain username and password specifications.

* Create an HTML page with three text form inputs, one labeled **Enter a Username**, the second one labeled **Enter a Password**, and the third one labeled **Re-Enter Password**. Beneath these three inputs will be a submit button with the text **Submit**.
* On submit, verify that the username is non-empty with a maximum of 10 characters using HTML verification.

* Instead of the text type, what type should the password forms be?
* Add javascript to verify that the passwords in the two password forms match exactly, and that a legal password has been enter.
  + A legal password must have at least 8 characters.
  + It must have at least one uppercase letter.
  + It must have at least one lowercase letter.
  + It must have at least one digit.
* If your password does not meet a respective condition, print an alert that states
  + Passwords do not match.
  + Password is too short.
  + Password does not contain an uppercase letter.
  + Password does not contain a lowercase letter.
  + Password does not contain a digit.
* Make sure your page is valid, using a validation service.
* Once completed, demonstrate your webpage for your instructor and have him initial here. If you do not finish during the lab period, then demonstrate your webpage at the beginning of the next lab period.

Instructor’s Initials \_\_\_\_\_\_\_\_

#### Part 2 – A Better Way to Create Forms

In this part of the lab, you will practice creating forms using javascript in a more generic manner.

* Download from the Canvas main course page the files **inputs.html, style.css** and **functions.js**.
* Open inputs.html in Notepad++
* Notice we are trying to append three children to newForm, but newForm has not been defined.
* Define a variable called newForm, use the document.querySelector() method to select the form element in the body of the web page, and set newForm to point to that element.
* Open functions.js in Notepad++.
* Notice there are two functions for making text nodes and labels, but there are no functions for making inputs and selects.
* Use the makeLabel function as a model and define a function called makeInput that takes the following arguments/parameters defined in order: type, id, value and label.
* The makeInput function should do the following:
  + Create a new input element.
  + Set the type attribute to the passed type parameter.
  + If the id is not undefined and not blank, set the id and name attribute to this value.
  + Note that === and !== are strict comparison operators where the objects being compared must have the same value and type. You should use the strict operators when checking to see if a variable in undefined.
  + If the value is not undefined and not blank, set the value attribute to this value
  + If the label is not undefined and not blank, do the following:
    - Use the makeLabel function to create a new label
    - Create a new div element
    - Append the new label and new input to the new div
    - Return the new div
  + If the label is undefined or blank, return the new input element
* Consider that this function has the ability to create all the different types of input elements, including text, password, date, email and even submit and reset buttons. It only sets the id, name and value attributes if they are passed as parameters and not blank. If a label is passed, it creates a new label and nests the label and input element in a surrounding div element.
* Use the makeInput function as a model and define a function called makeSelect that takes the following arguments/parameters defined in order: id, values and label.
* The makeSelect function should do the following:
  + Create a new select element
  + If the id is not undefined and not blank, set the id and name attribute to this value
  + Use a for-loop to iterate over the values array. For each value in the array do the following:
    - Create a new option element
    - Set attribute value of the option element
    - Instead of using the innerHTML field, use the makeTextNode function to create a text node with the array value and then append it to the option element you just created
    - Append the new option element to the select element
  + If the label is not undefined and not blank, do the following:
    - Use the makeLabel function to create a new label
    - Create a new div element
    - Append the new label and new select to the new div
    - Return the new div
  + If the label is undefined or blank, return the new select element
* Consider that select elements often have a list of numeric options, e.g., age or years of experience. We should avoid "hard coding" these kinds of arrays. Instead, write a function called makeArray the returns an array of strings for a numeric range from start to end. For example, the function call makeArray(2,5) would return ["2", "3", "4", "5"].
  + Inside your makeArray function, create an empty array like this: var myArray = [];
  + Then, write a for loop that goes from x = start to end inclusive
  + Inside your loop, you can convert the loop counter x to a string like this: var myString = x.toString()
  + Finally, add the string to the array like this: myArray.push(myString);
* Open inputs.html in Notepad++ and add the function calls to create all the form elements in the solution. You should add the following:
  + A password input with id and name equal to "passwd" and the label "Password: "
  + An email input with id and name equal to "email" and the label "Email: "
  + A date input with id and name equal to "birth" and the label "Date of Birth: "
  + A select element with id and name equal to "exp" and the label "Years of Experience" where the values range from 0 to 50
  + At the end of your loop be sure to return the array, i.e., return myArray;
* Verify that inputs.html validates and that it matches the solution at http://breimer.sienacs.com/courses/csis-390-s17/labs/lab4/solution.html
* Once completed, demonstrate your webpage for your instructor and have him initial here. If you do not finish during the lab period, then demonstrate your webpage at the beginning of the next lab period.

Instructor’s Initials \_\_\_\_\_\_\_\_

#### Part 3 – A Better Way to Validate Forms

In this part of the lab, you will be working with JavaScript in order to validate form input.

* In inputs.html add an event listener to the form so that a function called checkInputs is executed when the "submit" event is fired.
* In functions.js, write the checkInputs function to do the following:
  + Use document.querySelectorAll to get all the input elements that are inside of div elements. This function returns an array of elements called a nodeList. Assign the returned array to a variable called inputsArray
  + Write a for loop to iterate over the length of the inputsArray
  + Inside the loop, check to see if the value of each element is blank, i.e., inputsArray[i].value == ""
  + If the element's value is blank, add the attribute class = "error" to the element's parentNode. Note that this would be the div surrounding the element. Also, be sure to call event.preventDefault() so the form is not submitted. Remember to pass event into the function as a parameter. The event is a global variable the stores the event that was just fired.
  + Note that we have defined a CSS selector in styles.css that decorates div.error with red text and a red border.
  + If the element's value is not blank, remove the attribute class = "error" by calling setAttribute("class", "") on the element's parentNode
  + Note that it is superior to give the element a class name to apply the error styling rather than use JavaScript to do it directly. This allows a designer to change the style of an error without touching any JavaScript code. This embodies what we call separation of style. Typically, we separate the style from the HTML structure. But, in this case we are separating the style from the JavaScript behavior.

#### Part 4 – Custom Validation & More Events

#### In this part of the lab, you will implement validation that displays an error message and disables the "Insert" button. See the solution to understand how the form will function.

* First, we must make sure the username is not “admin” or “root”.
* In inputs.html add the following code:

document.querySelector("#username").addEventListener("change", function(event) {

if (this.value == "root" || this.value == "admin") {

this.parentNode.setAttribute("class", "error")

document.querySelector("#error\_message").innerHTML = "Invalid username";

document.querySelector("#insert\_button").disabled = true;

event.preventDefault();

}

else {

this.parentNode.setAttribute("class", "")

document.querySelector("#error\_message").innerHTML = "";

document.querySelector("#insert\_button").disabled = false;

}

});

* Test the code above to make sure it works.
  + This validation is so specific there is no need to create a function. Instead, we can use the anonymous function to implement functionality without defining a function.
  + The word **this** refers to the selected object or the object being "acted on".
  + Using **this** allows us to change the object inside the anonymous functions without having to select it again or use a variable.
* Using the previous code as a model, make sure the password is 10 or more characters long.
* Now make sure the selected "Age" - "Years of Experience" is greater than 18.
* **Create a zip file of your lab4 folder called lab4.zip and submit the file in Canvas. The zip file should contain three files: inputs.html, style.css and functions.js. In the comment area of Canvas put your partner's name.**

