HOMEWORK 5: Client Side User Agent Query and Forms Validation

Due date: Wednesday, week 8, at 11:59:59 when CSIL closes.
Status: DRAFT – may be changed.

Part A (20 pts) Browser Detection and Report

I. The ***requirements definition*** for part A:

Create a JavaScript <script> element embedded at the bottom of a webpage (you will continue to develop the page in Part B) that tests the "navigator.userAgent" property of the browser you are using and reports which one it is at the bottom of the webpage. Do this by using a RegExp with one of the string functions to test for the string appropriate to a given browser, and then write that into the webpage. See examples at:
http://www.w3schools.com/jsref/prop_nav_useragent.asp (Note underscores where spaces appear to be.)

II. ***Algorithm Construction*** *=> design the computational strategy for browser identification*

Design a conditional decision "tree" that would work on 3 different browsers including Chrome, Safari, and Firefox. Each condition uses a RegExp to search for a string in what is reported by the navigator.userAgent property that matches the browser in use. document.write() the correct one to the webpage. You will need the `document.open();` and `document.close();` commands for your writes.
Extra Credit: have a forth "default" condition that quotes the entire userAgent string.

Part B (80 pts.) Client-side Form Validation

I. The ***requirements definition*** for part B:

Develop a JavaScript function to test that all input fields have valid data before it is submitted to the web server. (Later we will do the same on the server side to validate that the data has arrived intact without any subverting code injected into it by a hacker.)

Building on the web page you made for homework 4, design a web page with the form input elements listed below. The JavaScript code must check that required data, **as specified below**, has been entered according to these specifications:

Required data are:

first name (must not be empty),
last name (must not be empty),
address (must not be empty),
city (must not be empty),
state (must be a string of two letters chosen from a selection list like the one you built with the 50 state abbreviations for hw4)
zip (must be a string of five digits),
email (a string that must contain one '@' and at least one '.')
two radio buttons with the choice between "Credit Card" or "BitCoin," (one must be checked)
Four checkboxes with "I am a member," "Please send the newsletter,"

"I am interested in preshow events," and "I'm not interested" (require at least one alternative, but allow any number from 1 to 4 checked.)

Each form input element must be matched with its own label element. You are to encapsulate either the whole list of inputs in a fieldset element, or divide up the elements by some categories of your own and provide each subgroup with its own fieldset element.

If one or more data fields do not meet the requirements, the form and data must not be sent and the user must be asked to change or supply the data, with a message indicating the problem. This should be done in a user friendly way, i.e., the *focus* should be set to the problem input, a message such as *First name required* could be inserted into the field where possible, and ideally some CSS change to the appearance of the field should be made so that the user knows where to go. Modifying the inputs for the failing content in a single run would be much better than a sequence of alert popups.

Set the `'action'` attribute of the form element to:
action="https://cs101.cs.uchicago.edu/~sterner/show-data.php"

Only when the data has been validated and found correct, can it be sent to the server. Sample validation scripts can be found at multiple sources online, and also:

https://cs101.cs.uchicago.edu/~sterner/javascript/validateFormOnSubmit.html

Note especially the RegExp case. You are encouraged, however, to invent or research your own RegExp designs and validation tests.

Use the URL below to go to entire JavaScript reference:

https://cs101.cs.uchicago.edu/~sterner/javascript/

II. ***Algorithm Construction*** *=> design the computational strategy for your validation program*

We have gone through examples of how to build a validation script in class. The basic idea involved a loop of some sort that "counts" each form input element, and a sequence of conditionals for the individual testing procedures for each kind of form input. Construct your own pseudo-code based on your answers to the following analytic questions. Then develop the pseudo-code into JavaScript.

How is the validation script called? What event(s) trigger the call for validation? What happens when the validation of an element fails? When it succeeds? How does the webpage form action get executed when all the form inputs are valid?

What is the "work" of the loop? What kind of loop is best fitted for that "work"?

What strategy will allow the matching of each iteration of the loop with the appropriate testing routine for the specific form input element to be validated?

What JavaScript statement structure will allow all the form inputs to be tested in sequence such that any failure stops the sequence, and any success goes to the next test?

What conditions for each input form need to be tested in order to assure it is valid? Think of how either users might submit wrongful data, or a hacker might try to inject some inappropriate data and then try to eliminate that.

How does a failure get sent back to the user so that the user is setup to correct the failure?

Among others, you will need these constructs, or ones with equivalent function, for this homework:

(a loop statement)

```
form.elements.length
form.elements[i].value
form.elements[i].value.match(regex)
form.elements[j].checked
form.elements[i].name
form.elements[i].focus();
var someRegEx  = /code inserted here/;
if (condition) {
      block of code to be executed if the condition is true
} else {
     block of code to be executed if the condition is false
}
```