Diary Entry

Link to SCIS website: http://student.athabascau.ca/~matsph/

Work for this Unit:

Pre-coding Activities:

Skimmed the Unit 4 and 5 JavaScript FAQ

Step 1: Choosing the project

Ideas:

1. Sorting and filter tools on the special patterns, brand versions, and other puzzles pages.
   On each page that lists items in a table, Javascript will reference a dataset (maybe in a separate file), sort through them according to the sort selected by the user (either from a drop-down menu or buttons on top of columns), and only show items that are in the category/have the tag that the user has selected (either in a navigation field, or a drop-down menu).
   - This would help Janice and Robert to search for special patterns and puzzles by giving them ways to specify what they are looking for.
2. Drop-down menus for each of the main page buttons with sub-pages/categories.
   The 'Special Patterns' menu button will have drop-down buttons for beginner, advanced, and other methods.
   The 'Solving Records' menu button will have drop-down buttons for Worldwide and U.S. results.
   The 'Other Puzzles' menu button will have drop-down buttons for Cuboids, Shape Shifting, Rubik's Cube Design, and Other.
   - This would help Sean to look through each of the main pages of the site and access the beginner methods directly, and would help Robert to go to the advanced solving methods from the home page, without having to click on the 'Solving Methods' link first.
3. Color/theme change option for website. A settings icon can be clicked to show radio buttons, icon buttons, and drop-down menus that can change settings like background-color, foreground-color, enable/disable images, theme, and font.
   - This would improve the browsing enjoyability of Janice, who personally likes exploring different options and enjoys customizing things.
   - Sean could change the color or theme to his liking, to make browsing the site more attractive to him, so he can more easily take up an interest in cubing.

All of my ideas were approved, and I decided to do option number 1, sorting and filter tools on the special patterns, brand versions, and other puzzles pages.

Step 2: Designing the program

After having my ideas approved, I needed to make a design document and submit it. But in order to know how my JavaScript is to be structured, I thought it would be easier to write it first, make a structure based on the JavaScript I wrote, submit the structure design document, and make any changes to the code according to what the tutor says, and then submit the code.

For the design, I have created a list of the classes, functions, and variables to be used in the program, as well as the requirements of the program, and a flow chart. I have attached the design in the pdf document "Program Design.pdf".

Step 3: Coding

Other unrelated HTML changes: I changed the 3 videos in the index page to embedded YouTube video objects and removed the comment that said "These YouTube videos will be video boxes when javascript is used".

First, in order to have something to sort with, I added extra items in each of the lists within the HTML. I also added a category field in each of the puzzles' descriptions since I was going to use it to filter the puzzles by category.

Then, I looked for videos and tutorials on how to sort with JavaScript. I searched on YouTube "JavaScript sort table", "JavaScript ascending order", "JavaScript filter items", "JavaScript array filter method", and "html sort table JavaScript". None of the videos I found showed how to sort a table like I have seen done on shopping sites like Newegg and Amazon. So I decided to make up a system entirely on my own, making up structures as I think of them.

I decided that in order to sort the list, I need to take the data from the HTML and put it into an array which I can read and sort in the JavaScript. So I set up an array, and made a 'for' loop to iterate each row of the array.

Throughout the programming, I often looked up on stack overflow or W3Schools methods of sorting arrays, pulling information from the DOM, or executing JavaScript code from HTML tags.

I then realized that I didn't know how to find the amount of rows in the table, or how to retrieve them in order to put them in the array; so I used the Java2 reference:

<http://www.java2s.com/Tutorials/Javascript/Buildin_Object/index.htm>

Then I ran a simple script that was supposed to grab every element with tag name "tr", and iterate through them replacing them with a number. I ran the JavaScript file for the first time, and it didn't work, so I thought it was because the JavaScript was loaded in the header, before the body of the DOM had time to load. So I put all the code in a function, and set the "window.onload" to the function. And it worked!

So now came the most difficult part, which was sorting the array. I found out that JavaScript had a 'sort()' function that allowed you to send it a custom function as a parameter, so I decided to use it, as it seemed to be the easiest way to re-order the table. I tried to use a custom function sort, but it didn't seem to work. I replaced all the rows with numbers and tried to sort, and it still didn't work. There were no errors, but it still didn't re-order the rows, even if their innerHTML was just a number.

I figured out the reason for this was that when the array was resorted, it just moved around the *references* to the rows, it didn't actually move the rows. That's when I thought I would need to copy the entire table into a new array in the new order, and then copy the array back onto the DOM.

But before I did this, I changed the code to first get references to all the rows grouped in 2, so that each item in the array referenced 2 rows. I did this by making the array a 2D array, of size [numberOfRows/2][2].

I then created my first custom function sort, an alphabetical sort. I set it up to sort the array by the title in the first cell of the first row, copied this reordered array into a new array, and overwrote the DOM with the new array.

After I did this, the JS code wouldn't run, and I didn't know why, so I ran the debugger in Firefox to check why it didn't work. I found out that I didn't initialize my 2D array properly, so I looked up "JavaScript 2D array", found out I needed to loop through and create a new 1D array in each item in the 1D array. So I did, that, and the table was now sorted alphabetically by title!

Now that I had a way to sort it, I needed a way to have the user choose the way to sort it. So I used an online reference guide (http://www.java2s.com/Tutorials/Javascript/Buildin_Object/index.htm>) to know how to use an HTML drop-down menu item. I found out what I was looking for was called the 'select' tag, and I learned how to execute a JavaScript command when an option is selected. I then created a select box in the HTML of the other puzzles page within the navigation menu, and had it run the sort function when the alphabetical option was chosen.

Then, in order to add other sorting options, I restructured the code to run the sorter, but added a switch statement to change the sorting function according to what the user wanted to sort by. I then added a sort for difficulty and rating.

I then worked on a filter system that would filter out categories according to what the user checks or unchecks. I first set up a backup system in an initialization function to restore any table rows that are deleted when filtering, but then I changed my mind and thought to setting the display to none for each row that was to be filtered out was a better idea.

But then I got into a big problem of not being able to specify a check box by ID within a form specified by ID, which would require chaining 'getElementById's, which is not allowed. So I instead gave each box its own ID, and referenced each using 'document.getElementById()'.

The filter function for the form boxes was very simple, and was largely based off of the sort function I already made. I had a few problems returning the display mode for the rows to the default, since I didn't know that the default display for row elements was 'table-row'.

After getting the form boxes to work, I fixed the select options to have the default option of 'Sort by:', and have it reset to that option after a page refresh by adding 'autocomplete="off"' to the select tag. I also fixed the form boxes so that all are checked by default.

After I was finished with the 'other_puzzles.html' page, I added the JavaScript and sort options to the 'special_patterns.html" page, but only added the alphabetical and difficulty sorts. This was simple and straightforward, except I had to reformat the special patterns page to include a navigation bar like in the other puzzles page, which required a little restructuring.

How I have met the Learning Outcomes:

Write well-structured, easily maintained JavaScript code following accepted good practice, including

- General appearance and form: well-commented, properly laid out, appropriate capitalization.
- Structure: modular, using functions, classes and objects effectively, making effective use of variables.
- Standards-compliant: avoiding proprietary elements where possible and working with the vast majority of web browsers.
- Accessible: usable in some form by people with a wide range of disabilities or, if not, failing gracefully and offering alternatives to achieve similar functionality.

General appearance and form and Structure:

Using my Unit 4 notes on JavaScript code conventions as well as the HTML comments section of my Unit 2 notes, I added comments and structured the code to fit the code conventions and give it good form.

I also used 2 functions, 1 custom function, and 29 variables in the 'sorter.js' program code.

Standards-compliant:

I tested the sorting and filter elements on Firefox, Opera, Internet Explorer, and Chrome browsers. I also ensured that the new HTML code and JavaScript were compliant with the XHTML 1.0 strict standards and the WCAG 2.0 standards.

Accessible:

The website should have an 'A' rating by the WCAG 2.0 standards for the most part, and the new changes to the website have also been tested on an Android device.

Make effective and efficient use of a full range of programming constructs including sequence, selection and iteration.

The following programming constructs were used:

Integer, String, Decimal, Reference, and Array Variables

Functions, built-in methods, and custom functions

Switch statements, for loops, and if/else statements

Effectively use variables, including passed parameters, local and global variables and arrays, to improve the efficiency, re-use, and maintainability of the code.

The sortKind() custom functions have 2 passed parameters, one for each element of an array.

About 29 different variables were used in the program, holding many different types of data, including:

- DOM element references
- Integers
- Custom functions
- HTML
- Boolean values

Use a wide range of programming features and commands to improve the efficiency, re-use and maintainability of the code.

The following built-in JavaScript functions and methods were used:

- document.getElementById() – To retrieve DOM elements
- element methods/properties:
  - selectedIndex – To get selected option of <select> element
  - children – To sort through the children of an element
  - value – Grab the value of a <select> <option>
  - innerHTML – To copy the HTML from inside an element
  - checked – To get the checked/unchecked status of checkboxes
  - length – To get the number of row elements in a document
  - style.display – Change the display property of a row element
- getElementsByTagName() – To select row elements
- slice() – For dissecting a piece of data from an item description
- indexOf() – To search for the location of a piece of data from an item description
- trim() – To make text uniform to be used in a switch statement
- toLowerCase() – To make text uniform to be used in a switch statement
- Array() – To create a set of 1D arrays in another 1D array to make up a 2D array

Write JavaScript code that works in all major browsers (including IE, Mozilla-based browsers such as Firefox, Opera, Konqueror, Safari, Chrome).

I wrote about 300 lines of JavaScript code that I tested in the Firefox, Chrome, and Opera browsers.

Effectively debug JavaScript code, making use of good practice and debugging tools.

I used the built-in Firefox debugger, by going line-by-line through code until it reported an error, which I read and researched to find where my code had a bug. I used the breakpoint feature to stop the execution of the code at the point where I wanted to see the execution in detail. I also kept track of the variables in the function scope in the variables tab on the right side of the debugger.

Other Requirements

Explain in detail how the code improves the experience of the personas you created in Unit 1

Persona 2: Janice Miranda

Scenario 1: Finding special Rubik's Cube patterns

Now that there is a navigation that allows sorting by title and difficulty, Janice can sort alphabetically to find a pattern she knows the name of, and by difficulty if she wants to learn a pattern that is more easy or especially difficult.

Scenario 5: The 5*5*5

The new sorting and filter options allows Janice to choose categories that interest her and filter out ones that don't, and sort the puzzles by what quality she thinks will give her more interesting puzzles.

Scenario 7: Solving away from home

Now that the special patterns allow alphabetical sort, she can search through the puzzles alphabetically instead of having to browse through them one by one.

Persona 3: Robert Connelly

Scenario 6: Buying a new puzzle

Robert can now filter the puzzles according to the categories he likes, and sort them to give puzzles according to difficulty or rating to give him puzzles that are either hard or with a good rating, which probably means that he will find cool.

Notes:

Went well:

The copying of the code from the alphabetical sort function and modifying it for the rating and difficulty sort functions. This part developed the most functionality with the least effort, as I already had all the bugs fixed in the alphabetical sort, all I had to do was make some changes to sort it differently, and it was easy and quick.

Didn't go well:

Getting and copying the elements from the DOM, as the JavaScript methods such as 'getElementId' would sometimes work and sometimes not, depending on which element or variable I was using it on. In some instances I had no idea why it wouldn't work, and so I just used a different method entirely.

What was most difficult and why:

Re-ordering the table according to the 'sort()' function. This was the most difficult because I had to get a list of all the rows, read their title to use it to sort with, and send a custom function as a parameter to the 'sort()' function, all without being able to see any results along the way, as it wouldn't work unless I had all parts working together. I eventually tried something simple like re-ordering numbers, and that eventually let me see what I did wrong, but before that I was taking stabs in the dark for hours.

If done again, would it be done differently and why:

Yes, I probably would have learned what variable types are being sent/received from functions like 'getElementId' so I would know what parameters work with them and what wouldn't. I would also test the built-in functions with something simple first so I know the basics of how it works, before I try it out on the complex problem in the project.

How did previous experience help/hinder completing the tasks:

Previous experience with using C++, Java, and Ruby programming languages helped me understand the JavaScript code, as it had the same structure and logic inherent in all of those languages. Also, using the code by someone else in the 'floater.js' file helped me understand how someone else uses JavaScript, and how it can be used to manipulate the DOM.

Most surprising thing learned:

JavaScript variables can take on the shape of pretty much anything, and they can change from integer, to decimal, to string, to array, to reference, without complaining about using the wrong types. I'm used to C++ always enforcing type of variable such as int, char, string, or references and pointers.

Most useful thing learned:

JavaScript's built-in 'sort()' function, since in many of my programming projects, I will often have to sort data in an array, but since I want to sort by different parameters for different objects, I had to make a custom sort function every time. Now I can just make a simple custom function and feed it to 'sort()'.

Map to Course Outcomes

In the original site mock-up, the 'Other_Puzzles.png' has a categories box that was supposed to have links to each of the categories, which I first intended to be on separate web pages. The JavaScript code I have written allows the user to select a category like the category links were supposed to do, but it does it better with more options and on one web page, and the sort drop-down menu adds to the functionality and control of the puzzle searching.

Self-assessment:

Learning Outcome	Evidence of Meeting the Learning Outcome	Your Own Assessment of the Grade You Believe Would Be Appropriate	Tutor's Justification of Grading (optional)
Apply a structured approach to identifying needs, interests, and functionality of a website.	I have made 7 scenarios that give structured processes of how the anticipated audience will address their needs with the website. I also took into account the constraints users might have, such as different operating systems, slow computers, and browsing the site using mobile devices.	B	
Design dynamic websites that meet specified needs and interests.	The JavaScript code in the 'floater.js' and 'sorter.js' files give the website dynamic content, which meets the needs and interests of the personas as described in 'Other Requirements' in the diary entries of Unit 4 and Unit 5.	A	
Write well-structured, easily maintained, standards-compliant, accessible HTML code.	Sample1.html and Sample2.html are edited to show my ability to write standards-compliant and structured HTML. The errors that used to exist in the code are listed in the learning diary under the section "Other Requirements". The 11 pages I wrote for my website all have links easily clickable on a mobile device, and are all well-structured and standards-compliant.	B	

Write well-structured, easily maintained, standards-compliant CSS code to present HTML pages in different ways.	The CSS I wrote is either in the external CSS file, or in a one-time internal CSS in the 'other_puzzles.html' page. In both, the code is organized with comments and well-structured using proper spacing, character case, property ordering, and naming schemes. It is easily maintained as the external CSS file is less than 130 lines long, and the internal CSS only defines styling for one selector. It is standards compliant with WCAG 2.0 by having no obscure or hard to read text, and proper contrast.	A	
Use JavaScript to add dynamic content to pages.	I have added the JavaScript file 'floater.js' which dynamically changes the positioning of the navigation field depending on where the user's viewport is.	B	
Critique JavaScript code written by others, identifying examples of both good and bad practice.	I modified the code written by LearnWebCode as described in the 'How I have met the Learning Outcomes' section of the diary entry for Unit 4.	A	
Select appropriate HTML, CSS, and JavaScript code from public repositories of open source and free scripts that improves your site and that enhances the experience of site visitors.	I selected JavaScript code from LearnWebCode as described in the 'How I have met the Learning Outcomes' section of the diary entry for Unit 4, which meets the needs and interests of the personas as described in 'Other Requirements' in the diary entry for Unit 4.	C	
Modify existing HTML, CSS, and JavaScript code to extend and alter its functionality, and to correct errors and cases of poor practice.	I edited the templates given in Unit 2 as described in the Unit 2 learning diary under the first requirement under the section 'Other Requirements'. I also modified the code written by LearnWebCode as described in the 'How I have met the Learning Outcomes' section of the diary entry for Unit 4.	C	

Write well-structured, easily maintained JavaScript code following accepted good practice, including	I have evidence of this listed below, as well as in the Unit 5 diary entry under the 'How I have met the Learning Outcomes' section, which both describe how I checked, edited, and wrote the JavaScript code.	B	
• general appearance and form: commented, properly laid out, appropriate capitalization	Using my Unit 4 notes on JavaScript code conventions as well as the HTML comments section of my Unit 2 notes, I added comments and formatted the code according to standards. I also used CamelCase notation, with lowercase first letters of variables.	B	
• structure: modular, using functions and objects effectively	I structured the code according to notes on proper spacing given in The Landing FAQs and the Moodle study guide. I also used 2 functions, 1 custom function, and 29 variables in the 'sorter.js' program code.	B	
• standards-compliant	I tested the sorting and filter elements on Firefox, Opera, Internet Explorer, and Chrome browsers. I also ensured that the new HTML code and JavaScript were compliant with the XHTML 1.0 strict standards and the WCAG 2.0 standards.	B	
• accessible	The website should have an 'A' rating by the WCAG 2.0 standards for the most part, and the new changes to the website have also been tested on an Android device.	B	
Write JavaScript code that works in all major browsers (including IE, Mozilla-based browsers such as Firefox, Opera, Konqueror, Safari, Chrome).	I tested the 'sorter.js' JavaScript code in the Firefox, Chrome, and Opera browsers, as well as on an android tablet.	B	

Effectively debug JavaScript code, making use of good practice and debugging tools.	I used the built-in Firefox debugger, by going line-by-line through code until it reported an error, which I read and researched to find where my code had a bug. I used the breakpoint feature to stop the execution of the code at the point where I wanted to see the execution in detail. I also kept track of the variables in the function scope in the variables tab on the right side of the debugger.	B	
Use JavaScript libraries (e.g., JQuery) to create dynamic pages.		A, B, C, D	
Use JavaScript to access and use web services for dynamic content (AJAX, JSON, etc.).		A, B, C, D	
Overall		A, B, C, D	