Web Apps Assignment 7

The purpose of this assignment is to continue to work with the DOM and callbacks and do more work with arrays.

Part 1: Create remote and local repos

1) Click on the link https://classroom.github.com/a/8BMMAljY and it will take you to github classroom where you will have your own private repo for the assignment.
2) You should have a bare bones README.md and sample code files: timeout, flex-add-elements, bootexample, and flexboxexample .
3) Clone down the repo to a local repo.
4) Open the repo in VS Code.
5) **Screen shot 1 of repo on VS Code for submission**

Part 2: Create basic HTML structure

1) Use VS Code to a new file named treasure.html. You can start with either flex or bootstrap. Checkout the sample code from Part 1.
2) Pick a descriptive title and for the header use "Treasure hunt" followed by your name.
3) Put a <p> or <div> in the body for the instructions "Click on the chests to search for treasure"
4) Add a <p> or <div> "Number of chests checked is zero." Give it an id of "chests".
5) Add a <p> or <div> "Score is unknown." Give it an id of "score".
6) Add a <p> or <div> "Help report". Give it an id of "help"
7) You may style them and move them around to make it look good.
8) Save your work.
9) Open the file with a browser (like Chrome) and verify that you have the title on the tab and your header is displayed.
10) Inspect the code
11) **Screen shot 2 on your browser of rendered page and the inspector**.

Part 3: Create the game field and populate it with images. The goal is to have a 5 by 5 display of images.

1) Find three images that are close to square in size
    a. The first image will be the starting image and should be something that is like a treasure chest
    b. The second image will be treasure
    c. The third image will be something that is not treasure. Could be an empty chest or a sad face.
    d. The final image will be in the middle square and should be an image for help.
2) Create the 5 rows using divs as the containers.
3) In each row add 5 divs to hold our search locations.
4) In each location put in the starting image (aside from the middle square)
5) Verify that you see the 5 by 5 array of images.

6) Commit and Push. You can either use git bash or github desktop to do your commit and push operations.
7) **Screen shot 3 on browser** (Should see instructions and the 5 by 5 game field.)

Part 4: Making the game operational.

1) Give each of the 25 images a unique id.
2) Give each of the 25 images an onclick property that is "check(x)" where x is an index. Each image gets its own index. So the first couple images would be
   onclick="check(0)"
   onclick="check(1)"
   …
   The middle image (index 12) will have its own definition
   onclick = "help()"

3) In a script we define the code to handle clicks.
4) Declare a variable ids which is an array of 25 strings, where each one is one of the unique ids from step 1.
5) Declare a variable chestCount which is zero
6) Declare a variable score which is zero
7) Declare a variable lastCheck which is 12.
8) Declare a variable treasureLocation which is 18.
9) Create the function check(position)
   a. Use the ids array to get the id for the image.
   b. Get the image element associated with that id.
   c. Change the source to the treasure or the not treasure image depending on if the position argument matches the treasureLocation.
   d. Add one to the chestCount.
   e. Add one to the score
   f. Update the text on the element with id chests.
   g. Update the text on the element with id score.
   h. Update lastCheck to the position
10) Create the function help () which has a result that is one of the strings "left", "right", "up", "down" or "sorry, no help available".
    a. We will assume that the values are 0 to 24 and represent the positions in a 5 by 5 grid as we read across left to right, and top to bottom. Compute the string to return as follows: Compute the row and col for the lastCheck by modding by 5 for the col and dividing by 5 for the row. Compute the row and col for the treasure location by modding/dividing by 5.
    b. Add two to the score
    c. Update the text on the element with id scores.
    d. Update the text on the element with id help. The following shows how to determine which string to use.
    e. If the last and treasure are in the same row, we use either "left" or "right" depending on whether the treasure is left or right of the lastCheck.

f. If the last and treasure are in the same col, we use either “up” or “down” depending on whether the treasure is up or down from the lastCheck.
g. In every other case, use “Sorry, no help available”. (This includes the possibility that the lastCheck and Treasure position are the same… They have already won the game.)
h. Example: The help for 2 (last) and 13(treasure) is “sorry, no help available” (not in the same row or col)
i. Example: The help for 7(last) and 13(treasure) is “sorry, no help available (not row, not col)
j. Example: The help for 1(last) and 16(treasure) is “down” (in the same col)
k. Example: The help for 16(last) and 1(treasure) is “up” (in the same col)
l. Example: The help for 10(last) and 13(treasure) is “right” (same row)
m. Example: The help for 10(last) and 10(treasure) is “sorry, no help available” (same row and col)

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

11) Test it out. Each click should change the image and produce feedback via the number of chests and the score.
12) Commit and Push.
13) **Screen shot 4 on browser** (Do an incorrect click, click the help, and then click on the treasure)
14) **Screen shot 5 on VSCode** showing the java script.

Part 5: Use setTimeout to create a reaction game. You will press a button to start a countdown. You will try to press a second button as close to zero time without being late.

1) Use VS Code to make a copy of timeout.html with the name reaction.html.
2) Pick a descriptive title and for the header use your name followed by “ reaction timer”.
3) You will need two elements (P or DIV) to hold text
   a. The first should have the id target and starts with the string “Your time”.
   b. The second should have the id countdown and starts with the string “Count”.
4) You will need two buttons.
   a. The first button will have the text “Start Timer”
   b. The second button will have the text “React Now”
5) We need some code to handle the buttons. We already have a script section so we start working there.
   a. We need a variable myTime to record when you pressed the react button. Initialize it to undefined.
   b. Create a function recordReact(). It will set myTime to Date.now(). This static method will return the current time as the number of milliseconds since the Unix epoch date of January 1, 1970, 00:00:00 UTC. Log the value to the console.
   c. Set the onclick property of the react button to call recordReact().

6) Check your code. Verify that the time is being captured on the console
7) Commit and Push.

Part 6: Add in the code to handle the count down and check against the time the reaction button was pressed.

1) Create a function startTime() and make the onclick property of the Start Timer button invoke the function.
2) In the first part of the method use a loop to invoke the setTimeout function 4 times with values of i = 0, 1, 2, and 3.
3) In the anonymous callback function, get the element with id countdown and set the innerHTML to i.
4) Set the second argument of setTimeout to be (3-i) times 1000. (We want the count to drop to zero after 3000 milliseconds.
5) Check that this works. If you press the Start Timer button. The count down should occur.
6) Now it is time to see how well the user did in attempting to click just before the timer hits zero and not after.
    a. After the for loop, invoke setTimeout one last time.
    b. Set the second argument to 3000.
    c. Set the callback to an anonymous function which checks to see if myTime is undefined. If so, the user was too slow. Set the innerHTML of the element with id target to “Too Slow”.
    d. Otherwise, we can see how well they did. Create a variable buttonTime and set its value to Date.now().
    e. Compute the difference.
    f. Set the innerHTML of the element with id target to “Time was “ plus the elapsed time.
7) Verify that the app works.
8) **Screen shot 6 on browser** (Show after a successful attempt.)
9) **Screen shot 7 on VSCode** showing the java script.
10) Commit and Push.

Part 7: Construct a “cycler” of items. Items will be added and removed from the end.

1) Use VS Code to make a copy of flex-add-elements.html with the name cycler.html.
2) Pick a descriptive title and for the header use your name followed by “ builds a cycler”.
3) You will need an input element of type text
4) You already have a button with the name “Add One”. Change it to “Add value”.
5) You already have a div flex-container that will hold the divs.
6) We need to make a few changes to the behavior of adding at the end of the flex container.
    a. For each of the starting div inside the flex container, add a class property of “cycler”.
    b. After you create the new div element. Get the value from the input element.

    c. Set the innerHTML of the new element to the value from the input.
    d. Set the className property of the new element to “cycler”.
    e. Append child to the container is already there, so try it out.
7) Every time you press the button you should get a new element with text from the input.
8) Commit and Push.

Part 8: Implement Get and Spin.

1) Add two buttons with text “Get” and “Spin”.
2) Define a **function get()** and associate it with the onclick for the get button.
3) We are going to remove the last element in the cycler. First use getElementsByClassname(“cycler”) to get an array like thing that has all the elements in the document with a class of “cycler”. Assign it to a variable cyclerItems
4) Get the length of cyclerItems. If the length is greater than 0, we will remove the last item from the flex container using cyclerItems[].remove(), with the index of the last item in the [].
5) Define a **function spin()** and associate it with the spin button.
6) We are going to remove the last item and put it at the front. Get the elements by class name “cycler” and assign to cyclerItems. If the length is 0 or 1, do nothing.
7) Otherwise, use the same kind of code as for get() to remove the last item but save the removed item in a variable removedItem.
8) Get the container div and then
    a. Use insertBefore on the container with arguments
    b. The first argument is the element removedItem
    c. The second argument is container.firstChild.
9) Now you can get and spin.
10) Test it out!
11) Commit and Push.
12) **Screen shot 8 on browser**.
13) **Screen shot 9 on VSCode** showing the java script.

Part 9: Host on Github pages

1) Go to setting on github and select the main branch.
2) Copy the URL.
3) Edit README and add three lines “Hosted at “ followed by a link to the URL from the previous step with the files hunt.html, reaction.html, and queue.html. (Commit and Push if you did it on the local repo.)
4) Verify that if you click on the links in the README, that you get your pages.

Bonus: Use the Math object to generate random values in the following ways. In treasure, initialize the location to a random value from 0 to 24. (Redraw the value is 12) Add in a new game button and method that resets the number of chests and score to zero and randomly selects a new location for the treasure. In react, Randomly choose a starting from value for the countdown of 2 to 10. In cycler, add a new button “Swap” and function that will randomly select two elements and swap their

Provide screen shots VSCode showing the use of random highlighted.