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/r4iG5kOC 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 search.html. You can start with either flex or bootstrap. Checkout the sample code from the previous part.
2) Pick a descriptive title and for the header use “Animal hunt” followed by your name.
3) Put a <p> or in the body for the instructions “Click on the squares to search for wild life”
4) Add a <p> or “Number of tries is zero.” Give it an id of “tries”.
5) Add a <p> or “Distance is unknown.” Give it an id of “distance”.
6) You may style them and move them around to make it look good.
7) Save your work.
8) Open the file with a browser (like Chrome) and verify that you have the title on the tab and your header is displayed.
9) Inspect the code
10) **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 4 by 6 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 a plant, tree, bush… anything vegetation that could conceal an animal.
    b. The second image will be an animal image. Pick something dramatic.
    c. The third image will be something that is not an animal, like an empty hole or a pile of dirt.
2) Create the 4 rows using divs as the containers.
3) In each row add 6 divs to hold our animal locations.
4) In each location put in the starting image.
5) Verify that you see the 4 by 6 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 24 images a unique id.
2) Give each of the 24 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)”
   …
3) In a script we define the code to handle clicks.
4) Declare a variable ids which is an array of 24 strings, where each one is one of the unique ids from step 1.
5) Declare a variable triesCount which is zero
6) Declare a variable animalLocation which is 13.
7) 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 animal or the not animal image depending on if the position argument matches the animalLocation.
   d. Add one to the tries.
   e. Update the text on the tries id element.
   f. Call the distance function with the click position and animal location and use the return value to update the text on the distance id element.
8) Create the function distance (pos1, pos2)
   a. We will assume that the values are 0 to 23 and represent the positions in a 4 by 6 grid as we read across left to right, and top to bottom. Compute the distance as the maximum of the difference in x and the difference in y.
   b. Example: The distance from 2 to 13 is 2 (max of 2 vertical, 1 horizontal.)
   c. The distance should never be negative and the order of arguments should not make a difference in the result. E.g. the distance from 2 to 11 is the same as the distance from 11 to 2. In both cases, it is 3.

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

9) Test it out. Each click should change the image and produce feedback via the number of tries and the distance.
10) Commit and Push.
11) **Screen shot 4 on browser** (Do two incorrect clicks and then click on the animal)
12) **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 "stack" of items. Items will be pushed and popped.

1) Use VS Code to make a copy of flex-add-elements.html with the name stack.html.
2) Pick a descriptive title and for the header use your name followed by " builds a stack".
3) You will need an input element of type text
4) You already have a button with the name "Add One". Change it to "Push 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 to the end of the flex container.
    a. For each of the div inside the flex container, add a class property of "stack".
    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 "stack".
    e. Append child 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 Pop and Unique.

1) Add two buttons with text "Pop" and "Zap values."
2) Define a function popItem() and associate it with the onclick for the pop button.
3) We are going to remove the last element added. First use getElementsByClassname("stack") to get an array like thing that has all the elements in the document with a class of "stack". Assign it to a variable stackItems
4) Get the length of stackItems. If the length is greater than 0, we will remove the last item from the flex container using stackItems[].remove(), with the index of the last item in the [].
5) Define a function zap() and associate it with the zap values button.
6) Get the elements by class name "stack".
7) Get the value from the input element.
8) Use a for-of loop on the stackItems. If the item has the same innerHTML as the value of the input, remove the item and log it to the console. (We are getting rid of all elements with that text.)
9) Now you can pop and zap.
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 search.html, reaction.html, and stack.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 animal, initialize the animal location to a random value from 0 to 23. Add in a new game method that resets the number of trues to zero and randomly selects a new location for the animal. In react, Randomly choose a starting from value for the countdown of 2 to 10. In stack, add a new button “Unique” and function that will remove duplicates from the stack keeping the value that is highest. E.G. If the stack contains A B E A C B C A D where D is the top, the resulting stack is E B C A D.

Provide screen shots VSCode showing the use of random highlighted.