CSE 154: Web Programming Spring 2018

Homework Assignment 3: ASCIImation

Due Date: Wed. April 18th: 11pm

Overview

This assignment tests your understanding of JavaScript and its interaction with HTML user interfaces. You will be creating a page that allows you to view ASCII art animations, or "ASCIImations." ASCII art is pictures that consist of text characters. ASCII art has a long history as a way to draw pictures for text-only monitors or printers. For example, an ASCII image of three getting your ducks in a row might look like this:

```
   _      _      _
__(.)< __(.)> __(.)=
\___)  \___)  \___)
```

Learning Objectives

- Continue to practice all of the learning objectives from Homeworks 1 and 2, including:
  - Carefully reading a specification.
  - Choosing semantically appropriate HTML tags to set the structure of a page and appropriate CSS rules to style a page.
  - Reducing redundancy in your CSS while producing expected output.
  - Producing quality readable and maintainable code.
- Practice using a new programming language (JavaScript) in the context of adding behavior to a webpage, comparing it with previous language(s) you have used before outside of web programming applications.
- Retrieve information from a user through HTML form elements using the Document Object Model (DOM) within your JS.
- Modify your web page using JS and DOM objects.
- Use the JS timer functionality to add an animation to your web page.
- Separate the behavior of your web page from the content and presentation by using unobtrusive modular JS code.

Your site will consist of three files. The first is `ascii.html` which will contain a user interface (UI) for creating/viewing ASCIImations (note that you will not be given any starter `.html` or `.txt` file for this assignment). Your page will be styled by linking to a style sheet named `ascii.css`. After creating your page, you will make the UI interactive by writing JavaScript code in `ascii.js` so that clicking the UI controls will cause the appropriate behavior.

You will also create an ASCIImation of your own, stored in a file named `custom-animation.js`. Your ASCIImation must show non-trivial effort, must have multiple frames of animation, and must be entirely your own work. Be creative! We will put students' ASCIImations on the course web site for others to see.

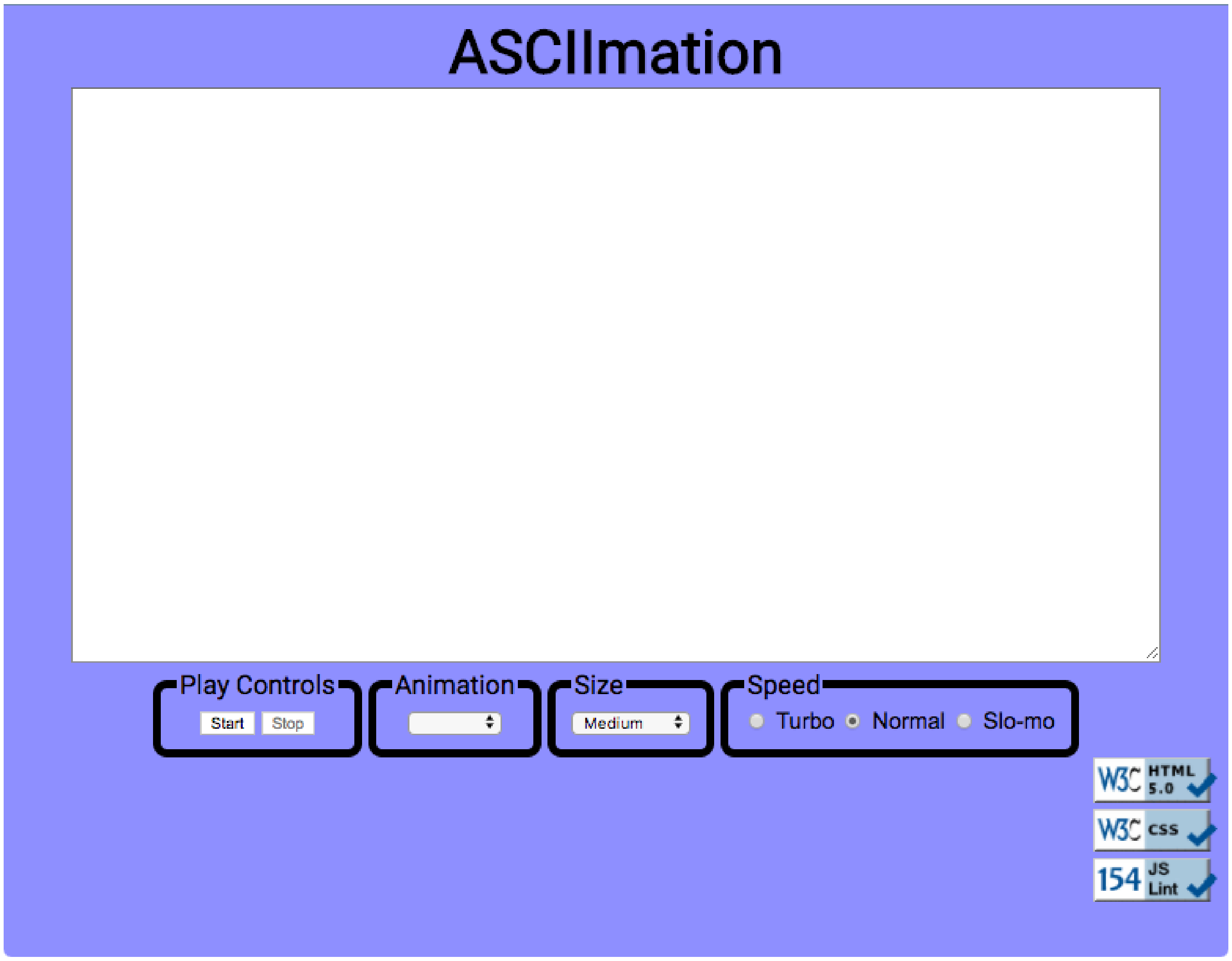

Figure 1: ASCII Animation Page: Expected output (rendered on Mac OSX)

Files to Submit

In total, you will submit the following files:

1. `ascii.html`: The file containing the HTML for your webpage.
2. `ascii.css`: The stylesheet for `ascii.html`.
3. `ascii.js`: The JavaScript code for `ascii.html`.
4. `custom-animation.js`: Your custom ASCII animation as JavaScript code (so it can be used on the page)

Images Provided

There are only three images in needed for this assignment, all located at the bottom of the page (see the External Requirements section for more details on placement and functionality). The source images you'll need are:

- https://webster.cs.washington.edu/images/w3c-html-blue.png
- https://webster.cs.washington.edu/images/w3c-css-blue.png
- https://webster.cs.washington.edu/images/w3c-js-blue.png

Provided `animations.js`

Your `ascii.html` page must link to the provided `animations.js` file using an absolute path:

- `https://webster.cs.washington.edu/js/asciimation/animations.js`

You will find more information on this file in the **Animation Controls** section of this document.

External Requirements

Your webpage *should match the overall appearance of the provided screenshots* and it *must match the appearance specified in this document*. We do not expect you to produce a pixel-perfect page that exactly matches the expected output image. However, your page should follow the specific style guidelines specified in this document and match the look, layout, and behavior shown here as closely as possible.

Appearance Details

- The page should have a title of **ASCIImation**.
- The overall page has a background color of `#9999FF`.
- The font of the page heading and the text labeling the four control options ('Play Controls', 'Animation', 'Size', and 'Speed') should be 'Roboto' (imported from Google Fonts), falling back to the default monospace font on the user's machine.
- The page's heading text (bolded) should have a font size of 32pt and should be centered horizontally. There is no margin between the heading content area and other neighboring content on the page.
- Any text in the animation area should have the default monospace font on the user's machine and should be bold. Initially, its font size should be 12pt.
- This text box has 80 columns and 20 rows and is centered horizontally on the page (**Hint:** you should specify the column and row properties in the corresponding HTML for the text box. This is ok because not the fundamental these are properties defining the HTML element, not the fundamental styling). Its width is 90% of the page area and its height is 400px.
- Below the text box is a set of controls grouped into several field sets, each with a 5px black border around it (with 10px border radius) and a label on top. You should make sure that the tops of the field set line up by setting their vertical alignment.
- The text labeling the four control options should have a font size of 14pt. Any text inside of these four control option regions should have a font size of 12pt and a preferred font family of Arial, falling back to the default sans-serif font on the user's machine if they do not have the Arial font available.
- The text box and control field sets are centered horizontally. The legends of the field sets should be text-aligned left.
- The choices available in the Animation drop down are: Blank, Exercise, Juggler, Bike, Dive, Ducks, Pony, Custom.
- The choices available in the Size drop down list box are Tiny, Small, Medium, Large, Extra Large, XXL. These correspond to the sizes 7pt, 10pt, 12pt, 16pt, 24pt, 32pt respectively. Initially Medium is selected.

- Below the controls is a right-aligned section with images that are links to the W3C validators and our own JSLint tool. Each image appears on its own line. Each image should be 70px wide. These images should remain 5px from the bottom right corner no matter how the page is resized. The three images are found as listed in the Images Provided section above and should link to the following:

Image:	**Links to:**
`w3c-html-blue.png`	`https://validator.w3.org/#validate_by_input`
`w3c-css-blue.png`	`http://jigsaw.w3.org/css-validator/#validate_by_input`
`w3c-js-blue.png`	`https://courses.cs.washington.edu/courses/cse154/17au/jslint/jslint.html`

Behavior Details

The behavior of this web page can also be viewed in the video located http://recordit.co/YHFHi1lYFi, but don't rely only on what you see in the video. Details on how the web site should function are listed below.

Play Controls:

Start Button:

- When the page is idle, all frames of the animation are visible. Frames are separated by 5 equals signs and a line break (\n) character.
- When clicked, animation begins. When animation starts, whatever text is currently in the text box is broken apart to produce frames of animation. This might be a pre-set animation, or text that the user has typed manually.
- During animation, one frame is visible at any moment, starting with the first frame. The speed of the animation is shown is set by the Speed control.
- When the animation reaches the last frame, it loops back around and repeats indefinitely (for full credit, you must implement your animation using a JavaScript timer with the `setInterval` function.)

Stop Button:

- When this button is clicked, any animation in progress halts. When the animation is stopped, the full text that was in the text area immediately before animation began is redisplayed.

Animation Controls:

This control group is a drop-down list of ASCII animations the user can choose from. When one of the animations is chosen (onchange event), the main text area updates to display all text of the chosen animation. Initially the Blank animation is selected and no text is showing in the text area.

If you've successfully linked to the provided `animations.js` file (see first page for absolute path), you will have access to the default ASCIImations. `animations.js` declares the starting ASCIImations as global string variables named `EXERCISE`, `JUGGLER`, `BIKE`, `DIVE`, `DUCKS`, and `PONY`. You shouldn't edit this file, but your `ascii.js` can refer to these variables. For example, if you have a `textarea` on your page with an id of 'my-textarea', you can set the `textarea` value to be the ducks animation with the following statement:

```
document.getElementById(‘‘my-textarea’’).value = DUCKS;
```

The provided `animations.js` file also defines a global associative array named `ANIMATIONS` that maps from indexes (keys) that are strings equal to the names of the animations, such as "Bike" or "Ducks", to values that are long strings representing the entire animation text for that image. Using this array well can help you avoid redundancy.

Here is a short example that uses the `ANIMATIONS` array:

```
let whichOne = "Ducks";
document.getElementById("mytextarea").value = ANIMATIONS[whichOne];
```

The user may type new text in the field after choosing a pre-set animation. The animation shown when Play is pressed should reflect these changes. (i.e., Don't capture the text to animate until the user presses the Start button.) You may assume that the user will not try to type into the text area while animation is in progress. You may also assume that the user will not use the selection box to change to a new animation while animation is occurring; assume that the user will stop any existing animation before changing to a new one.

Custom Animation:
The Custom choice in the Animation box should show an animation that you have created. To create an animation that can work correctly in this assignment, you should use our String Maker tool which converts your animation to a string you can put into `custom-animation.js`. The text of your animation **should consist entirely of plain ASCII characters, not special characters outside of the ASCII character range**. Your converted result should define a CUSTOM variable with the 'let' keyword at the top. You can find the String Maker here:

- https://courses.cs.washington.edu/courses/cse154/18sp/18sp-data/homework/hw03/stringmaker.html

Note: You will need to link to `animations.js` *before* the link to `custom-animation.js` since `animations.js` defines the `ASCIIMATIONS` array, and `custom-animation.js` assigns `ASCIIMATIONS[CUSTOM]` (from String Maker). Link to the three JS files in your `head` in the following order:

1. `https://webster.cs.washington.edu/js/asciimation/animations.js`
2. `custom-animation.js`
3. `ascii.js`

Font Size Controls

A drop-down list of font sizes. When one of the font sizes is chosen, it immediately sets the font size in the main text area. The font sizes listed in the drop-down list, and the corresponding font size to set, are:

- Tiny (7pt), Small (10pt), Medium (12pt), Large (16pt), Extra Large (24pt), XXL (32pt)

Initially Medium is selected and the text is 12pt in size. If the animation is playing and one of these buttons is clicked, the font size changes immediately.

Note that when you write the code for changing the font sizes, it is easy to introduce redundancy. By setting a value attribute on each of the options in the drop-down list, you can avoid a long series of if/else statements.

Speed Controls

Contains three radio buttons labeled "Turbo", "Normal", and "Slo-Mo". When one of the buttons (or the text next to it) is clicked, causing the box to become checked, it changes the speed of animation. Turbo uses a 50ms delay, Normal is 250ms, and Slo-Mo is 1500ms. Initially the Normal button is checked and the delay is 250ms.

If the animation is playing and the buttons are clicked, the change should take effect immediately (the user shouldn't have to stop and restart the animation to see the change). Checking the box shouldn't cause the animation to start if it wasn't already started. It also shouldn't reset what frame is showing; it should just change the delay immediately

Control Enabling/Disabling

Modify your GUI to disable elements that the user shouldn't be able to click when they are not supposed to. Initially, and whenever animation is not in progress, the Stop button should be disabled. When an animation is in progress, the Start button and the select box of animations should be disabled. The Size box and the Speed radio buttons should always be enabled.

Enable or disable a control with its disabled property. For example, to disable a control with id of `customerList`:

```
document.getElementById("customerlist").disabled = true;
```

Development Strategy and Hints

1. Write the basic HTML content including the proper UI controls. (Don't use the form tag.)
2. Write your CSS code to achieve the proper layout.
3. Write a small amount of "starter" JS code and make sure that it runs. (For example, make it so that when the Start button is clicked, an alert box appears.)
4. Implement code to change the animation text and font sizes. Make it so that when an option is chosen in the selection box, the proper text string appears in the text area. Get the font size options working.
5. Implement a minimal Start behavior so that when Start is clicked, a single frame of animation is shown. Clicking Start multiple times would show successive frames of animation.
6. Use a JavaScript timer to implement the proper animation based on your previous code.

We strongly recommend that you use the Firefox Development Tools or Chrome DevTools on this assignment, or use the similar tool built into other browsers. Both show syntax errors in your JavaScript code. You can use either as a debugger, set breakpoints, type expressions on the Console, and watch variables' values. This is essential for efficient JavaScript programming.

Our JSLint tool can help you find common JavaScript bugs. Since this is your first JavaScript program, you will probably encounter tricky bugs. If so, paste your code into JSLint to look for possible errors or warnings. For full credit, your JavaScript code must pass the provided JSLint tool with no errors reported. ("Warnings" are okay.)

For full credit, your .js file must be written in JavaScript "strict" mode by putting this exact line of code at the top:

```
"use strict";
```

Internal Requirements

- Minimize the use of global variables. Do not ever store DOM element objects, such as those returned by the `document.getElementById` function, as global variables. As a reference, our own solution has three global variables, mostly related to timing, the set of frames to draw, which frame is currently displayed, and so on.
- For full credit, your page must use valid HTML5, CSS3, and JS and successfully pass the W3C HTML5 and W3C CSS3 validators and our JSLint (https://courses.cs.washington.edu/courses/cse154/17au/jslint/jslint.html) with no errors.

- Do not express style information in the HTML, such as inline styles or presentational HTML tags such as `b` or `font`. Your HTML page should link to your style sheet through your style tag.
- Properly separate your JavaScript from your HTML, using unobtrusive JavaScript so that no JavaScript code, onclick handlers, etc. are embedded into the HTML code. Your HTML page should link to your JS file in a script tag.
- Use organizational tags such as `header`, `footer`, `aside`, `article`, `div`, `section`, and `nav` for dividing your page into sections and give them semantic meaning.
- Prefer organizational tags over ids and classes when appropriate.
- Use `h1-h6` tags as appropriate.
- Express all stylistic information on the page using CSS defined in `ascii.css`.
- Part of your grade comes from expressing your CSS concisely and without unnecessary or redundant styles. For example, the content block shares many (if not all) styles in common, so you should not specify those styles twice in your CSS file.
- Use context selectors to avoid needing to apply classes and ids to large numbers of elements.
- Format your HTML, CSS, and JS to be as readable as possible, similarly to the examples from class:
  - Place a comment header in each file with your name, section, a brief description of the assignment, and the file's contents.
  - Your JavaScript should have more descriptive comments using JSDoc, including a header on each function (including anonymous) and complex sections of code describing the relevant code, the function's behavior, etc.
  - Properly use whitespace and indent your HTML, CSS, and JS code as shown in class.
  - To keep line lengths manageable, do not place more than one block element on the same line or begin a block element past the 100th character on a line.

Academic Integrity

As with any CS homework assignment, you may not place your solution to a publicly-accessible web site, neither during nor after the school quarter is over. Doing so is considered a violation of our course academic integrity policy. As a reminder: The University of Washington has an entire page on Academic Misconduct on their Community Standards and Student Conduct Page. Please acquaint yourself with the University of Washington's resources on academic honesty, and in particular how academic misconduct will be reported (which has been changed for 2017).