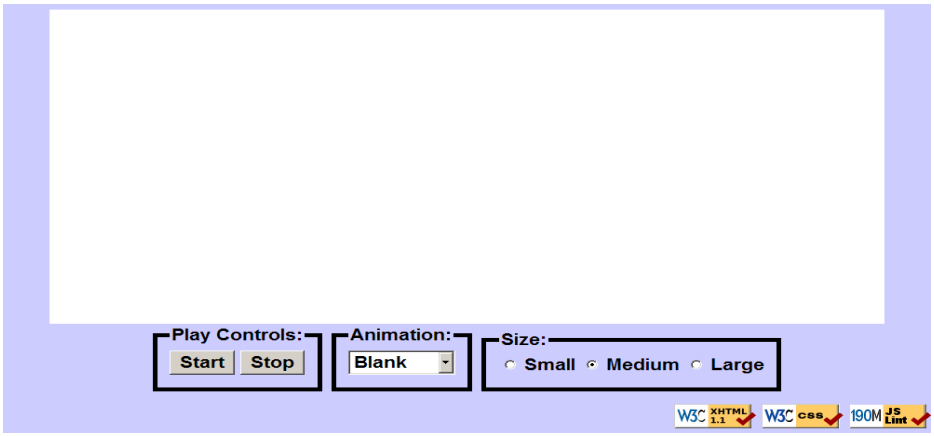
HW2: ASCIImation

Special thanks to Dave Reed of Creighton for the original idea of this nifty assignment.

This assignment tests your understanding of JavaScript and its interaction with HTML user interfaces. You must match the appearance and behavior of the following web page:



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. We will draw animated ASCII art, or "ASCIImation." Groups of nerds are working to recreate the entire movies e. g. *Star Wars* as ASCIImation.

The first task for is to create a page `ascii.html`with a user interface (UI) for creating/viewing ASCIImations. The skeleton `ascii.html` links to a skeleton CSS style sheet `ascii.css`. On the course web site are skeletons of both of these files to start with. After creating your page, you must make the UI interactive using JavaScript in `ascii.js` so that clicking the UI controls causes appropriate behavior.

The final part of your task is to create an ASCIImation of your own, stored in a file named `asciimation.txt`. Your ASCIImation must show non-trivial effort, must have multiple frames of animation, and must be entirely your own work. Be creative! In total you will turn in the following files:

- `ascii.html`, your web page
- `ascii.css`, the style sheet for your web page
- `ascii.js`, the JavaScript code for your web page
- `asciimation.txt`, your ASCII animation

Appearance Details:

The page's title is "ASCIImation". The page should have a Favorites icon ("favicon") of `favicon.gif`.

Under the page's heading is a text box with 80 columns and 20 rows, centered horizontally. Its width is 90% of the page size and its height is 400px. It has no border and uses a 12pt monospace font initially. Even though the CSS width/height properties will determine the text area's true size, you must include `rows` and `cols` attributes in your `textarea` HTML element for the page to validate and display properly.

Below the text entry box is a set of controls grouped into field sets with 5px black borders and labels. Their behavior is described in the Behavior section of this document. To make the field sets display in the same line rather than each in its own block.

The bottom of the page has a right-aligned section of links to the W3C validators and our JSLint tool.

Behavior Details:

The following are the groups of controls at the bottom of the page and each control's behavior:

Play Controls:

Start: When clicked (`onclick`), animation begins. When the the page is idle, all frames of the animation are visible. Frames are separated by 5 equals signs and a line break: `"=====\n"`

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 animation changes frames once every 200ms. When the animation reaches the last frame, it loops back around and repeats indefinitely.

You may assume that the user will not click Start again while the animation is already running.

Stop: When clicked, halts any animation in progress. When animation is stopped, the text that was in the box before animation began is returned to the box. The user should be able to click Stop multiple times in a row.

Font Size:

Contains three radio buttons. When one of these buttons (or the text next to it) is clicked, it immediately sets the font size in the main text area to small (7pt), medium (12pt), or large (24pt).

Initially the medium button is checked and the text is 12pt in size. Only one font size button can be selected at a time. If the animation is playing and one of these buttons is clicked, the font size changes immediately.

Animation:

A drop-down list of ASCII animations. When one of the animations is chosen (`onchange`), the main text area updates to display all text of the chosen animation. The choices available are: **Blank, Exercise, Juggler, Bike, Dive, Custom**

Initially the Blank animation is selected and no text is showing in the text entry box.

The `ascii.html` page links to a provided file `animations.js` that declares the ASCIImations `String` variables named `exercise`, `juggler`, `bike`, and `dive`. **You shouldn't edit this file**, but your `ascii.js` file can refer to these variables. For example, if you have a `textarea` on your page with an `id` of `mytextarea`:

```
$("mytextarea").value = juggler;
```

Note that the user may decide to type new text into the field after choosing a pre-set animation, and the animation shown when Play is pressed should reflect these changes. (In other words, don't capture and split the text to animate until the moment the user presses the Start button.)

You may assume that the user will not try to type into the text area while animation is occurring. You may also assume that the user will not change to a new animation while animation is occurring; assume that the user will stop any existing animation before changing to a new one.

The Custom choice should show your own custom ASCIImation that you have created. A utility will be available on the course web site to convert your `asciimation.txt` file into a long JavaScript string, suitable for inclusion at the bottom of your `ascii.js` code. Please don't place any comments or other header information at the top of your `asciimation.txt` file; it should contain only your custom animation, so that if someone did a Select All, Copy, and Paste into your program, it would run as is.

NOTE: Please understand that you are turning in your custom animation in two ways: once in your `asciimation.txt` file (as plain text), and once in your ascii.js as an encoded String variable so that it can be selected from the GUI.

Extra Features:

In addition to the previous requirements, you must also complete **at least one of the following additional features**. If you want to complete more than one, that is fine, but only one is required.

1. *Turbo speed*: Add an additional fieldset to the GUI, just to the right of the font size section, with a similar border and title of "Speed:". The field set contains a single checkbox labeled "Turbo". When the box is checked, it sets the speed of animation to use a 50ms delay instead of 200ms.

   Initially the box is unchecked and the delay is 200ms. If the animation is playing and the box is checked/unchecked, the speed changes immediately.

2. *Control Enabling*: Your GUI should disable any elements that the user shouldn't be able to click at a given moment. Initially and whenever animation is not in progress, the Stop button should be disabled. When animation is in progress, the Start button and selection box of animations should be disabled.

   The size radio buttons (and Turbo checkbox, if you implement it) should always remain enabled regardless of whether the animation is currently playing. The drop-down list should be enabled when the page is idle but should be disabled while the animation is playing.

   Set a control to be enabled or disabled by changing its boolean `disabled` property. For example, to disable a control with `id` of `customerlist`, you would write:

```
$("customerlist").disabled = true;
```

3. *Unobtrusive JavaScript*: Modify your XHTML file to contain no JavaScript code whatsoever, as will be discussed in lecture. If you do this extra feature, the only permitted JavaScript in your XHTML file should be the link to your external `.js` file in the page's header. In other words, no `onclick` or other event handlers should be embedded in the XHTML body code. Instead, attach all event handlers using `window.onload` and the Document Object Model (DOM) methods as described in lecture.

Near the top of your JS or HTML file, put a comment saying which extra feature(s) you have completed. If you implement more than one, also comment which one you want us to grade (the others will be ignored). Regardless of how many additions you implement, the main behavior and appearance should still work as specified. If you have a different idea for a creative addition to this program, please ask us and we may approve it. A screenshot of the extra feature output will be provided on the course web site.

Implementation and Grading:

We suggest writing this program using the following development strategy:

- Edit the XHTML file to add the proper UI controls.
- Write your CSS code to achieve the proper layout.
- 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.)
- Implement the code to change animations and font sizes. That is, make it so that when an option is chosen in the selection box, the proper text string appears in the main text area. Also make it so that when the font size radio buttons are clicked, the text area's font size changes appropriately.

- Implement a minimal Start behavior so that when Start is clicked, a single frame of animation is shown. Clicking Start multiple times will show successive frames of animation.
- Use a JavaScript timer to implement the proper animation based on your previous code.

Since this is your first exposure to JavaScript programming, you will probably encounter frustrating bugs. You may find that your code does nothing at all when you load the page, or you may see an unexpected value. We strongly recommend that if you are having JavaScript troubles, you paste your code into our JSLint tool and see if it contains any errors or warnings. JSLint can help you find many common JavaScript bugs.

For reference, our `ascii.js` is around 70 lines long including comments (excluding our custom ASCII art), though you do not need to match this.

For full credit, your page must successfully pass the W3C XHTML 1.1 validator. Choose appropriate XHTML tags to match the structure of the content on the page. Do not express stylistic information in the XHTML page itself, such as inline styles, inline CSS/JS code in the HTML page's header, or presentational XHTML tags such as `b` or `font`.

Express all stylistic information on the page in CSS using your style sheet file. For full credit, your style sheet must successfully pass the W3C CSS validator. You should not use HTML or CSS constructs that have not been discussed in lecture, slides, or textbook chapters during the first three weeks of the course.

Format your HTML, CSS, and JS so that they are as readable as possible, similarly to the examples shown in class. Also place a comment in each HTML/CSS/JS file containing your name and section. Implement the behavior of each onscreen control using JavaScript event handlers defined in your script file.

For full credit, your JavaScript code should pass the provided JSLint tool with no errors reported (when the Recommended options are checked). You should also follow reasonable stylistic guidelines. In particular, you should avoid redundant code, minimize the number of global variables, utilize parameters and return values properly, correctly use indentation and spacing, and place a comment header on top of the file and atop every function explaining that function's behavior.