CSE 154: Web Programming — Spring 2017

Homework Assignment 3: Speedreader

Due Date: Tuesday, April 18th

This assignment tests your understanding of JavaScript and its interaction with HTML user interfaces.

Overview

Everyone wishes they could read faster. One method for doing so is called Rapid Serial Visual Presentation (RSVP).

RSVP is based on three observations:

1. Using your finger or some other pointing device to train your eyes and focus while reading increases speed.
2. Eliminating subvocalization, internally speaking words while reading them, can dramatically increase your reading speed.
3. Speed reading skills rely on the readers discipline to develop good reading habits, and it is easy for a reader to learn "the wrong way" and thus never see the purported benefits of speed reading.

Computer programs are a great help in this area as they force readers to accurately do one and two while avoiding three. RSVP programs do this by presenting words to the reader in quick succession. Therefore, the reader is only able to focus on a single word at a time. And furthermore, the words appear at such a speed that the reader is unable to subvocalize like normal.

Your first task is to create a page `speedreader.html` with a user interface (UI) for speed reading. No skeleton files are provided. Your page should link to a style sheet you'll write named `speedreader.css` for styling the page. After creating your page, you must make the UI interactive by writing JavaScript code in `speedreader.js` so that clicking the UI controls causes appropriate behavior. Your HTML page should link to your JS file in a script tag. In total you will turn in the following files:

- `speedreader.html`, your web page
- `speedreader.css`, the style sheet for your web page
- `speedreader.js`, the JavaScript code for your web page

Our screenshots were taken on Windows in Firefox, which may differ from your system.

Appearance Details

The page should have a title of `SpeedReader` centered horizontally. The overall page has a background color of `#EAF6F6`. The preferred font for all text on the page is Garamond in size 12pt. If that is not available it should use the default serif font available on the system. Under the page's heading is a div with a width of 80% and a height of 100px, centered horizontally. It uses a 36pt bold font initially, has the system default monospace font and has a line height of 100px. It is surrounded by a solid 2px wide border in `#8EBEBE` with 5px curved corners. Its background color is `#FFFFFF`. Text inside it should be centered.

Below the div is a set of controls grouped into several field sets with title text on top of each. 10px of margin separates these controls from the speedreading div above, and the text input below. These titles should be bold, aligned left, have a bottom border of solid line 1px thick in `#8EBEBE`. They should be 100% wide and have 2px of space between the title text and where the title texts border would be on all sides.

The first three field sets should appear in one line, the last on the line below. The field sets are centered horizontally. To get the field sets to appear in a row horizontally, there are a couple ways that you could do this.

One way is to make the controls flex-items in flex-boxes, and use `justify-content` to align them. Another way would be use change the display of the controls to `inline`. Either way is fine with us. If you want to use the `inline` solution: see textbook Chapter 4's section about Element Visibility and the display property. You should make sure that the tops of the field sets line up by setting their vertical alignment. The last field set, which should appear on its own line, contains a text area box. This text area box should have 10 rows and 80 columns. It should have a solid gray border 1px wide. It should have a background of `#FFFFFF`.

Some of the controls contain buttons. These buttons should have a `#FFFFFF` background and a solid gray border 1px wide. They should be 70px wide. When they are disabled their background color should be lightgray. Below the controls is a left-aligned section with images that are links to the W3C validators and our own JSLint tool. Each image should be 70px wide. These images should remain 5px from the bottom left corner no matter how the page is resized. The three images are found at the following locations, where you should prepend `https://webster.cs.washington.edu/` to each image and file:

Image:	**Links to:**
`images/w3c-html-blue.png`	`validate-html.php`
`images/w3c-css-blue.png`	`validate-css.php`
`images/w3c-js-blue.png`	`jslint/?referer`

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, word display animation begins. Each frame of animation is a single word. Separate words on white space (spaces, tabs and new lines).
You can do this by using the following method call: `var result = yourString.split(/[ \t\n]+/);`

Replace `yourString` with whatever you would like to split on white space. If a word ends with a piece of punctuation (comma, period, exclamation point, question mark, semicolon or colon) the punctuation should be removed and that word should be displayed for twice the normal amount of time (*hint: have a list of frames to display and add this frame twice in a row*). You should only remove the last piece of punctuation if there are multiple in a row.

When animation starts, whatever text is currently in the text box is broken apart to produce frames of animation as described above. These are displayed in the div. By default, the word changes once every 171ms. When the animation reaches the last word, it should stop. (For full credit, you must implement your animation using a JavaScript timer with the setInterval function.)

Stop: When clicked, halts any animation in progress. When animation is stopped, the div should become blank.

Font Size: Contains three radio buttons with text "Medium", "Big", and "Bigger". When one of the buttons (or the text next to it) is clicked, causing the box to become checked, it immediately sets the font size in the div. Initially Medium is selected and the text is 36pt in size. If Big is selected the text should be 48pt and if Bigger is selected it should be 60pt. If the animation is playing and one of these buttons is clicked, the font size changes immediately.

Speed: A drop-down list of speeds. When one of the speeds is chosen, it immediately sets the speed in the div. The speed listed in the drop-down list, and the corresponding speed to set, are:

- 50 wpm (1200ms), 300 wpm (200ms), 350 wpm (171ms), 400 wpm (150ms), 450 wpm (133ms), 500 wpm (120ms)

If the animation is playing and a different speed is selected, the change should take effect immediately (the user shouldn't have to stop and restart the animation to see the change). Selecting the timing 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. 350 wpm should initially be selected.

Note that when you write the code for changing the speed, 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.

Control Enabling/Disabling: Modify your GUI to disable elements that the user shouldn't be able to click at a given time. 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 text area should be disabled. The Size radio buttons and the Speed box 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.
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 div. 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.
7. Get rid of punctuation and add longer delays at the ends of sentences and at commas.

We strongly recommend that you install and use the Firebug add-on for Firefox on this assignment, or use the similar tool built into other browsers such as Chrome. 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 serious 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";`

Implementation and Grading

Submit your assignment online from the course web site. All of your HTML, CSS, and JavaScript code should follow the style guide posted on the class web site. Implement your page using HTML5 as taught in class. Your page must pass the W3C HTML5 validator. Choose appropriate tags to match the structure of the page content. Do not express style information in the HTML page itself, such as inline styles or presentational HTML tags such as `b` or `font`. Use unobtrusive JavaScript so that no JavaScript code, `onclick` handlers, etc. are embedded into the HTML code.

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 to be readable, like to the examples in class. Place a comment header atop each HTML/CSS/JS file. Your JavaScript should have more descriptive comments, including a header on each function (including anonymous) and complex sections of code describing the relevant code, the function's behavior, etc. You should follow reasonable style guidelines similar to those of a CSE 14x programming assignment. In particular, avoid redundant code, and use parameters and return values properly.

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 four global variables, mostly related to the set of frames to draw, which frame is currently displayed, the delay between frames, and so on.

Format your code similarly to the examples from class. Properly use whitespace and indentation. In your HTML, do not place more than one block element on a line or begin a block element past the 100th character. In your JavaScript, properly space/indent your code, and do not write any lines of code longer than 100 characters. You should not use any external JavaScript frameworks or libraries such as jQuery to solve this assignment. Do not place a solution to this assignment on a public web site.