1. HTML (5pts): What's wrong with my HMTL??!?!?!?

1 point each correct change on the LINES ONLY

```
1. <html lang="en">
2.   <heading>  <!-- Line 1 -->
3.     <meta charset="utf-8" />
4.     <link rel="stylesheet" href="styles.css" />
5.     <script src="index.js" />   <!-- Line 2 -->
6.     <title>(NOT) To-Do List</title>
7.   </heading>  <!-- Line 1 -->
8.   <body>
9.     <header>
10.        <h1>The world's first (NOT) To-Do List!</h1>
11.        <img src="logo.jpg" />  <!-- Line 3 -->
12.      </header>
13.      <main>
14.        <p>Enter a thing you don't want to do, and then don't do it!</p>
15.        <div>
16.          <input id="item-in" name="item" type="text">
17.          <button>Add item!</button>
18.        <div>  <!-- Line 4 -->
19.        <ul>
20.          <li>Wait until the last day to start the HW</li> <!-- line 5 -->
21.          <ul>
22.            <li>And then miss the lock deadline</li>
23.          </ul>
24.          <li>Forget to validate and lint my code</li>
25.          <!-- More items to not do are added here by JS! -->
26.        </ul>
27.      </div> <!-- line 6 -->
28.    </body>
29.  </html>
```

1. <heading></heading> tag before body should <head></head> (on line 2, 7)
2. <script> tag is not self closing, should be <script></script> (on line 5)
3. <img> tag should have an alt attribute (line 11)
4. Closing tag for block starting at line 16 should be (line 18)
5. Closing on line 20 should be moved to after the on line 23
6. The closing on line 27 should be a </main> to match with the main on line 13
7. (bonus secret error) missing <!DOCTYPE html> on line 1.

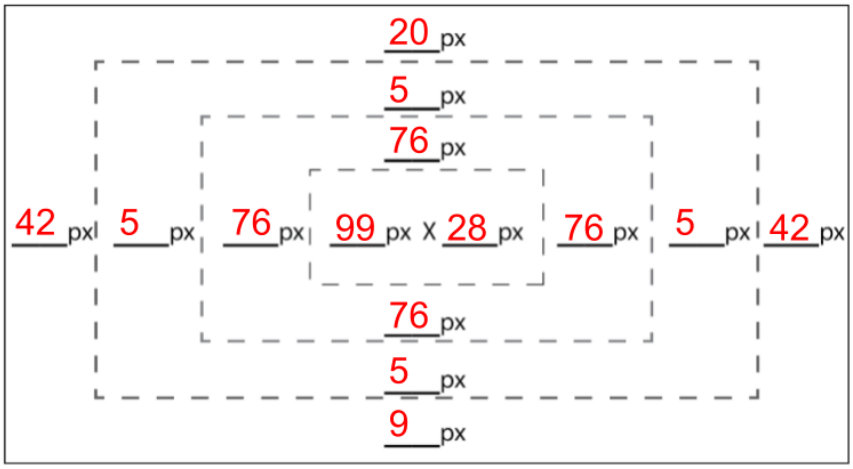
2. CSS Selectors (5 pts): Gotta Select ‘em *!

1 point each

Selector	IDs of selected elements
`h2`	`#d, #i`
`section em`	#e, #n
`.red-text`	`#i, #l, #r`
footer > img	#p
`li.red-text or ol .red-text`	#l

3. Short Answer (10 pts)

3A. Box Model (2 pt):



3B. Flex Box (2 pt):

½ point each answer

```
#box {
  display: flex;
  flex-direction: column-reverse;
  align-items: center;
  justify-content: space-between;
}
```

3C. JSON (3 pts):

½ point each answer

JavaScript Statement	Value
`pony.cooler`	`20.0`
`pony.bestfriends[1]`	`"Fluttershy"`
`pony["is-mane-character"]`	`true`
`pony.birthday.length`	`error`
`pony.show.quotes.length`	`2`
`pony.show.quotes[1].episode`	`"Newbie Dash"`

3D. Event Handling (3 pt):

Scenario 1 and 2, ½ pt, 3 and 4 1 pt each

	Scenario 1: The page is loaded.	**Scenario 2:** Page is loaded, then Button B is clicked.	**Scenario 3:** Page is loaded, then Button B is clicked then Button A is clicked.	**Scenario 4:** Page is loaded, then Button B is clicked then Button A is clicked then Button B is clicked.
Output:	1 4 5	1 4 5	1 4 5 2	1 4 5 2 4 3

3E. DOM manipulation (1 pt):

Both classes "hide" the element; they will not be seen. hidden-a (display: none) removes the element from the DOM; it no longer takes up space, and other elements around it will be rendered as if the element doesn't exist. hidden-b (visibility: hidden) keeps the element in the DOM, but renders it invisible; other elements will render as if it is there.

4. CSS (10 pts): A GOURD-geous Pumpkin Patch

```
/* pumpkin.css */
body, fieldset, #patch {
  /* student to write the two rules (0.5pts)
*/
  margin-left: auto;
  margin-right: auto;
}

/* Student to write the selector (0.5pts) */
body, footer  {
  display: flex;
}

/* student to write the selector (1pt) */
body, button, input {
 text-align: center;
 font-family: "Comic Sans MS", cursive;
}

h1 { font-size: 30pt; }

/* student to write the selector (1pt) */
/* Alternate: */
/* button, input, p */
button, input, #message {
  font-size: 14pt;
}

.pumpkin {
  position: absolute;
  background-size: cover;
  width: 40px;
  height: 30px;

  /* student to write (0.5pts) */
  border-radius: 50%;
}

/* student to write the selector (0.5pts) */
footer > img  {
  height: 20px;
  width: 20px;
}

#patch {
  position: relative;
  width: 500px;
```

```
#message {
  height: 30px;
  /* student to write this rule (0.5pts) */
  margin: 5px;
}
/* Add additional rulesets here */

/* student to write the whole selector/rule
set (1pt) */
body, fieldset {
  width: 80%;
}

/* student to write the whole selector/rule
set (1pt) */
fieldset, #patch {
 border-radius: 5px;
}

/* student to write this whole selector/rule
set (1pt) */
button {
  border: 1px solid black;
}

/* Provided CSS */
body {
  flex-direction: column;
  justify-content: center;
}

footer {
  justify-content: center;
  align-items: center;
}

fieldset { vertical-align: middle; }

.hidden { visibility: hidden; }
.shown { visibility: visible; }

.rotting0 { opacity: 1.00; }
.rotting1 { opacity: 0.75; }
.rotting2 { opacity: 0.5; }
.rotting3 { opacity: 0.25; }

.goofy {
  background-image: url("goofypumpkin.png");
}

.happy {
  background-image: url("happypumpkin.png");
```

height: 500px;
 	
/* student to write (2pts) */
border: 3px solid purple;
padding: 5px;
margin-bottom: 5px;
}
	}
	
	.sad {
	background-image: url("sadpumpkin.png");
	}

5. JavaScript/DOM/Events (12 pts): Planting Pumpkins in the Patch

```
"use strict";

(function() {

  /* Add any module-global variables you need here. */

  let timer = null;

  window.addEventListener("load", init);

  /**
   * Init sets up event listeners and buttons.
   */
  function init() {
    /*
     * Problem 5, Part A)  Write the code that will cause plantPumpkins() to be called
     * whenever the user clicks on the #plant button.
     */
    id("message").classList.add("hidden");
    id("plant").addEventListener("click", plantPumpkins);

    // initTimerButtons is a given function that sets up listeners for
    // other timer-related features.
    initTimerButtons();
  }

  /**
   * Problem 5, Part A) Implement the function that gets the number of pumpkins from
the
   * input field and the calls loadPumpkins. Shows an error message if the quantity is
   * invalid.
   */
  function plantPumpkins() {
     // get the value from the count field
    let count = parseInt(id("count").value);
```

```
  if (count >=  1) {
    loadPumpkins(count);
  }
  else {
    showMessage("Enter a positive integer", 2000);  // short message for exam
  }

}

/**
 * Problem 5, Part B) Write the function that adds the pumpkins into the DOM.
 * See spec details above.
 * @param {number} number The quantity of pumpkins to add.
 */
function loadPumpkins(number) {
  const pumpkinTypes = ["happy", "sad", "goofy"];

  for (let i = 0; i < number; i++) {
    let rand = Math.floor(Math.random() * pumpkinTypes.length);
    let newPumpkin = gen("div");
    newPumpkin.classList.add("pumpkin");
    newPumpkin.classList.add(pumpkinTypes[rand]);

    /* specify that newPumpkin location comes back as JSON object, you can set the
     * top and left in this way. in specification */

    let location = newPumpkinLocation();
    newPumpkin.style.left = location.left;
    newPumpkin.style.top = location.top;


    id("patch").appendChild(newPumpkin);
    newPumpkin.addEventListener("click", removePumpkin);

    /* ignored for testing purposes */
    if (timer === null) {
      id("start").disabled = false;
    }
}
```

```
/**
 * Problem 5, Part C) Write the function to remove a pumpkin element from the DOM
and
 * remove its event listener.
 * See the spec for all details.
 * @param {object} event Object containing information about the triggering event.
 */
function removePumpkin(event) {
  let object = event.currentTarget;
  object.removeEventListener("click", removePumpkin);
  object.parentNode.removeChild(object);
}
```

6. JavaScript/Timers (8 pts): Time to turn into a pumpkin

```
/**
 * Problem 6, Part A) Shows a given message for a certain amount of time
 * and then hides it.
 * @param {string} message The message to show.
 * @param {number} time The duration (ms) the message appears for.
 */
function showMessage(message, time) {
  id("message").classList.remove("hidden");
  id("message").textContent =  message;
  setTimeout(function() {
    id("message").textContent = "";
    id("message").classList.add("hidden");
  }, time);
}

/**
 * Problem 6, Part B) Write the function that calls rotPumpkins every 2 seconds.
 */
function startRotting() {
  id("start").disabled = true;
  id("stop").disabled = false;

  timer = setInterval(rotPumpkins, 2000);

}

/**
 * Problem 6, Part C) Write the function that stops the "rotting" process.
 */
```

```
function stopRotting() {
  id("start").disabled = false;
  id("stop").disabled = true;
  clearInterval(timer);
  timer = null;

}
```

```
/**
   * Problem 6, Part D) Write the function that calls rotAPumpkin, giving it a
specific
   * pumpkin DOM element to "rot". You can assume rotAPumpkin exists and works as
   * expected (the method header for it is given below).
   */
  function rotPumpkins() {
    let pumpkins = qsa(".pumpkin");
    for (let i = 0; i < pumpkins.length; i++) {
        rotAPumpkin(pumpkins[i]);
    }
  }

  /***
   * Below this line are the interfaces and documentation for functions provided to
you.
   ***/

  /**
   * newPumpkinLocation (given) calculates a random location to place a pumpkin based
on
   * the distance from the top-left of the container.
   * @returns {object} JSON object with two fields: top and left.
   */
  function newPumpkinLocation() {
    // ...
  }

  /**
   * initTimerButtons (given) adds event listeners to the start and stop buttons for
   * startRotting and stopRotting respectively.
   */
  function initTimerButtons() {
    // ...
  }

  /**
   * rotAPumpkin (given) applies styles to an element that make it appear as if it's
   * disappearing.
   * @param {DOMElement} el the element to style as if a pumpkin is "rotting away."
   */
  function rotAPumpkin(el) {
    // ...
  }
```

```
})(); // end module
```