CSE 154: Web Programming Spring 2019

Midterm Key

Question	Score	Possible
HTML		5
CSS Selectors		5
Short Answer		8
CSS Styling		10
JS with DOM/Events		12
JS with Timers		10
Total		50

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

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <header> <head>
4     <title>Mowgli's House</title>
5     <meta charset="utf-8" />
6     <link href="style.css" rel="stylesheet" />
7     <script src="main.js" />></script>
8   </header> </head>
9   <body>
10    <h1>Mowgli's House</h1>
11    <p id="text">Hi, I am Mowgli. Welcome to my house!
12      <img src="img/mowgli1.png" alt="Mowgli the dog"/>
13    </p>
14    <p>You can find my friends <a src="www.mowglisfriends.com" >here!</a></p>
15    <p id="text">Things that I like: </p>
16    <ul>
17      <li>Sleep</li>
18      <li>Play with debug duckies</li>
19      <li>Write code on my own!</li>
20    </ul>
21   </body>
22 </html>
```

1. Line 7: <script> tag is not self-closing.
2. Lines 3 and 8: <header> should be <head> (<header> is invalid outside of <body>)
3. Line 12: Missing alt text for image
4. Line 14: src should be href
5. Line 14: Missing closing
6. Line 15: Duplicate #text id, one needs to be eliminated or renamed or both changed into a class.

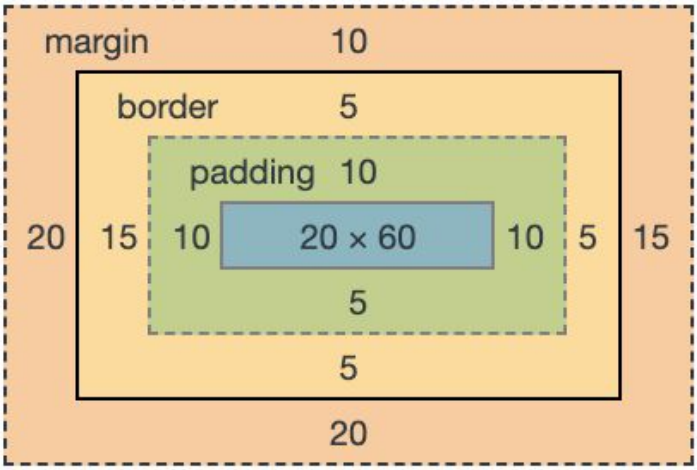
2. CSS Selectors (5 pts): Gotta Select 'em *!

Selector	IDs of selected elements
ol li	#h, #i, #k, #l
section > img	**#n**
ol > li	**#h, #i**
section li li	#k, #l
body section > ol img	**#m**

3. Short Answer (8 pts)

1. Box Model (2 pt): In the Box Model diagram to the right, label the corresponding Box Model properties given the CSS styles for a #content div (the size of the box is denoted as width x height):

```
#content {
  width: 20px;
  height: 60px;
  border: 5px solid black;
  border-left: 15px solid black;
  margin: 20px;
  margin-top: 10px;
  margin-right: 15px;
  padding: 10px;
  padding-bottom: 5px;
}
```



2. Event Handling (1 pt): Suppose your page has a button `#btn` that has a function `doThing` registered to handle the click event when the page is loaded. Now suppose you are adding to some function in the same program where you would like to temporarily prevent user interaction with that button.

Give a brief 1-2 sentence explanation as to why you would want to use `id("btn").disabled = true` **instead of** `id("btn").removeEventListener("click", doThing);` to achieve this result.

2 types of answers accepted:
1. That .disabled shows a visual cue to the user as well as not allowing them to click and that if you do .removeEventListener to disable functionality you have to re-add the event listener and *why* .disabled is preferable in either case.
2. .disabled will disable all event listeners on a button, not just the one you do with .removeEventListener, i.e you have to remember to remove all of them and re-add all of them.

3. JSON (3 pts)

JavaScript Statement	Value
`miniJSON["is-active”]`	`true`
`Row 2: miniJSON["total-minutes"]`	`150.35`
`Row 3: miniJSON["friends-list"][1]`	`"Ericka"`
`miniJSON.friends-list.length`	`error or undefined`

`miniJSON["plan"]["distance"] or`
`miniJSON.plan.distance`
`or`
`miniJSON.plan.runs[0].distance +`
`miniJSON.plan.runs[1].distance`
	`13`

`miniJSON["plan"]["runs"][1]["distance"]`	`7`

4. Page Events (1 pt).

a)
```
drink
bake
simmer
tea
```

b)
```
simmer
bake
drink
tea
```

c)
```
simmer
tea
drink
bake
```

d)
```
tea
drink
simmer
bake
```

5. (1 pt): So far we have used JavaScript to:

1. Respond to UI events (click, dblclick, etc.)
2. Respond to a timed event with setInterval/setTimeout
3. Handle responses returned from a fetch request

Give a brief (1-2 sentence) explanation about what these features of JS share in common.

This was an all or nothing response to see if you have synthesized a mechanism that is being used by JavaScript to allow us (as programmers) to respond to events or actions asynchronously. As such we accepted any of the following explanations

1. All three use a “callback” function (a function named passed into another function as an argument, which is then called later from JavaScript when some event or action happens).
2. All three execute the code asynchronously.
3. Mentions that all three are in response to a behavior or event (something that happens) in JS.

We did not accept the generic response that these are general features of JS (they are also general features of other languages), nor did we accept that all three involve user interaction or that they add something to a web page. While that may be true in some cases, it is not in all cases.

4. CSS (10 pts): CSS (Zen) Tulip Garden

```
/* Start of garden.css - finish the necessary
   styles for each ruleset */
body, button {
  font-family: Helvetica, sans-serif;
  text-align: center;
}

button {
  font-size: 16pt;
  margin-top: 10px;
}

#garden {
  margin-left: auto;
  margin-right: auto;
  width: 400px;
  height: 400px;
  /* Add other styles for #garden below */

  border: 2px solid black;
  background-color: brown;
  padding: 1%;

  display: flex;
  justify-content: space-around;
  padding: 1%;

}

.row {
  background-color: ivory;
  width: 20%;
  display: flex;
  justify-content: space-around;
  align-items: center;
  flex-direction: column;
}

.seed {
  border-radius: 50%;
  background-color: black;
}
```

```
/* Provided CSS (at end of file) */
footer img {
  height: 30px;
  vertical-align: middle;
}

.spot {
  width: 20px;
  height: 20px;
}

.sprout {
  background-image: url("img/grass.png");
}

.purple-tulip, .red-tulip, .yellow-tulip {
  height: 60px;
}

.purple-tulip {
  background-image: url("purple-tulip.png");
}

.red-tulip {
  background-image: url("red-tulip.png");
}

.yellow-tulip {
 background-image: url("yellow-tulip.png");
}
```

5. JS/DOM/Events (12 pts): Planting DOM ~~Trees~~ Tulips!

```
(function(){
  "use strict";

  function grow(item) { /* Details of function not provided  */ }

  window.addEventListener("load”, init);

  /** Begin Problem 5 solution below: */
  function init() {
    let spots = qsa(".spot");
    for (let i = 0; i < spots.length; i++) {
      spots[i].addEventListener("click", function() { grow(spots[i]) });
    }
    id("grow-all").addEventListener("click", growAll);
    let rows = qsa(".row");
    for (let i = 0; i < rows.length; i++) {
      rows[i].addEventListener("dblclick", addSeed);
    }
    id("grow-all").disabled = false;
  }

  function addSeed() {
    let newDiv = document.createElement("div");
    newDiv.classList.add("spot");
    newDiv.classList.add("seed");
    newDiv.addEventListener("click", function() { grow(newDiv) });
    this.appendChild(newDiv);
  }

  function growAll() {
    let allSpots = qsa(".spot");
    for (let i = 0; i < allSpots.length; i++) {
      grow(allSpots[i]);
    }
  }
})();
```

6a. Code Execution (10 pts): Duck, Duck, Goose!

6a. Program A (5 Points):

Console output:	
Line 1	*1 ducks*
Line 2	*2 ducks*
Line 3	*goose*
Line 4	*1 ducks*
Line 5	*2 ducks*
Line 6	*3 ducks*
Line 7	*goose*
Line 8	*1 ducks*

6a. Program B (5 Points):

Console output:	
Line 1	*1 ducks*
Line 2	*2 ducks*
Line 3	*goose*
Line 4	*3 ducks*
Line 5	*4 ducks*
Line 6	*5 ducks*
Line 7	*6 ducks*
Line 8	*goose*

6B. JS/Timers - Code Writing (10pts): The Final Countdown!

Note: The second example output provided in this problem was unclear. The intent was that the second click should pause/reset the timer with the last output being 7, then a third click should start the next countdown shown above. Some interpreted this output to mean that the second click reset the timer to 10 and continued to count down (with no stopping). We accepted both interpretations.

```
(function () {
  let timerId = null;
  let count = 10;

  window.addEventListener("load", init);

  function init() {
    id("btn").addEventListener("click", handleClick);
  }

 /**
   * Solution that matches the specification (not the example 2):
   * click -> 10, 9, 8, 7
   * click -> stop
   * click -> 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, stop
   * **/
  function handleClick() {
    // start countdown
    if (timerId === null) {
      timerId = setInterval(countdown, 1000);
    } else {
      clearInterval(timerId);
      timerId = null;
      count = 10;
    }
  }

  function countdown() {
    console.log(count);
    count--;
    if (count < 0) {
      clearInterval(timerId);
      timerId = null;
      count = 10;
    }
  }

  /**
   * Solution that matches the example:
   * click -> 10, 9, 8, 7
   * click -> 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, stop
   */
  function handleClick2() {
    // start countdown
    if (timerId2 === null) {
      timerId2 = setInterval(() => {
        console.log(count2);
        if (count === 0) {
          clearInterval(timerId);
          timerId = null;
          count = 10;
        } else {
```

```
          count--;
        }
      }, 1000);
    } else { // stop countdown
      count = 10;
    }
  }

})();
```