Note: We strongly recommend printing out practice exams and working through them with only your cheatsheet (provided on the course website) - it's important to be comfortable taking a spec and writing code on paper without the convenience of autocomplete/debugging tools!

Also note that provided exams adapt problems from previous quarter exams, but the number/format of problems may be different (see other provided practice exams for other example problems). This exam in particular is a bit longer than we'd expect for 50 minutes.

Name:
UWNet ID: @uw.edu

TA (or section):

Rules:
- You have 60 minutes to complete this exam.
- You will receive a deduction if you keep working after the instructor calls for papers.
- This is a closed-note exam, but you may use the provided cheatsheet for reference. As noted on the cheatsheet, you may assume $, qs, and qsa are provided in JS as shorthand for document.getElementById, document.querySelector, and document.querySelectorAll, respectively.
- You may not use any electronic or computing devices, including calculators, cell phones, smartwatches, and music players.
- Unless otherwise indicated, your code will be graded on proper behavior/output, not on style.
- Do not abbreviate code, such as writing ditto marks (""’) or dot-dot-dot marks (…). You may not use JavaScript frameworks such as jQuery or Prototype when solving problems.
- If you enter the room, you must turn in an exam and will not be permitted to leave without doing so.
- You must show your Student ID to a TA or instructor for your submitted exam to be accepted.

<table>
<thead>
<tr>
<th>Question</th>
<th>Score</th>
<th>Possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML Validation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSS and the DOM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JS/DOM/UI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JS/Animations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Short Answer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. What’s wrong with my HTML?

Consider the following HTML:

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="index.js"></script>
    <link href="styles.css" rel="stylesheet"/>
  </head>
  <body>
    <h1>Best page ever!</h1> <!-- 1. h1 is not self-closing; cannot have text inside of tag -->
    <div>
      Check out this cool page: <!-- 2. Missing closing </a> -->
      <a href="http://www.pointerpoint.com" <!-- 3., 4. href should be attribute in <a>; <href> is not a tag -->
      </a> <!-- 5. Closing span tag without open tag found -->
      </div>
  </body>
</html>
```

**Solutions:**

1. **h1 is not self-closing; should be `<h1>"Best Page ever!"</h1>`

2. **Missing closing `<a>` - should be `<a href="http://www.pointerpoint.com">http://www.pointerpoint.com</a>`

3. **href should be attribute in `<a>; see 2) for entire fix**

4. **<href> not a tag; see 2) for entire fix**

5. **Missing open `<span>; fix could be to remove </span> (or include open span, though unnecessary)**
2. Cute, Slow, and Sleepy.

In this problem, you will **A.** finish drawing a DOM tree of a provided HTML body and **B.** write CSS with the provided HTML to produce the expected page output below (appearance details are given to supplement the expected output image where needed).

```html
<body>
  <main>
    <header>
      <h1>Baby Sloth!</h1>
    </header>
    <div id="s">
      <div id="l">
        <span id="o"></span>
        <span id="t"></span>
        <span id="h"></span>
      </div>
    </div>
  </main>
</body>
```

**Part A (Drawing the DOM Tree):**

**Solution:**

```
body
  main
    header
    div
      h1
      div
        span
        span
        span
```
Part B: Solution

```css
.main {
    margin: auto auto;
    width: 40%;
}

h1 {
    font-family: Helvetica, Arial, sans-serif;
    text-align: center;
}

div, span {
    border: 1px solid black;
    border-radius: 50%;
}

span {
    background-color: black;
    height: 20px;
    width: 20px;
}

#s {
    background-color: sienna;
    height: 150px;
    margin: auto auto;
    width: 150px;
}

#l {
    align-items: center;
    background-color: peru;
    display: flex;
    height: 110px;
    justify-content: space-between;
    margin-left: 20px;
    margin-top: 20px;
    width: 110px;
}

#t {
    height: 25px;
}
```
3. Define-It!

Solution:

(function() {
    "use strict";

    window.addEventListener("load", function() {
        $("add-entry").addEventListener("click", addEntry);
    });

    function addEntry() {
        let term = $("term").value;
        let definition = $("definition").value;
        if (term && definition) {
            let li = document.createElement("li");
            li.innerText = term + ": " + definition;
            $("entries").appendChild(li);
            $("current-entries").classList.remove("hidden");
            li.addEventListener("dblclick", updateEntries);
            $("term").value = "";
            $("definition").value = "";
        }
    }

    function updateEntries() {
        $("entries").removeChild(this);
        if (!qsa("li").length || $("current-entries").classList.contains("hidden")) {
            $("current-entries").classList.add("hidden");
        }
    }
})();
4. JavaScript DOM and Animations (graph.js)

Solution:

(function() {
    "use strict";

    window.addEventListener("load", function() {
        setInterval(newPoint, 1000);
    });

    function newPoint() {
        let point = document.createElement("div");
        point.className = "point";
        let x = Math.random();
        let y = Math.random();
        // subtract 8 to account for diameter of point
        point.style.left = x * (600 - 8) + "px"; // don't forget the units!
        point.style.top = y * (600 - 8) + "px";
        let red = Math.round(x * 255);
        // inverse blue ratio for distance from
        let blue = Math.round(255 - (y * 255));
        point.style.backgroundColor = "rgb( + red + \, 0 \, + blue + )";
        point.addEventListener("dblclick", removePoint);
        document.getElementById("graph").appendChild(point);
    }

    function removePoint() {
        this.parent.removeChild(this);
    }
})();

5. Short Answers

1. (Flexbox question)

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

2. Why is it important to limit the use of module-global variables in our JavaScript programs?

    **Possible solutions:** To avoid cluttering the module-global namespace, reduce management of information in JS when we can keep variables in local scope (easier to run into bugs when we don’t update module-globals when needed), possible redundancy in the page (e.g. when DOM elements are stored as module-global even though we access to them from the DOM).
3. What is the role of the id of a timer returned by `setTimeout` or `setInterval`? In particular, when do we need it?

**Possible solution:** Returned to keep track of id window uses to manage its timers; need this id to call `clearInterval(timerId)` or `clearTimeout(timerId)`, as well as to test if the id is still null (indicating no animation is in progress)

4. (HTML vs. JS code quality)

**Possible solutions:**

<table>
<thead>
<tr>
<th>Issue A and justification:</th>
<th>There should not be a script tag inside of the HTML; this is poor separation of content (HTML) and behavior (JS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Better alternative:</td>
<td><strong>Move code inside of the script tag into a linked JS file (within a module)</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Issue B and justification:</th>
<th><code>innerText</code> should be used rather than <code>innerHTML</code>; HTML tags should be added with JS using <code>document.createElement(“tagname”)</code>; we're also just adding text, not HTML here.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Better alternative:</td>
<td><strong>Use <code>document.getElementById(“result”).innerText = “you clicked the button!”;</code></strong></td>
</tr>
</tbody>
</table>

5. (JSON Mystery)

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mystery[&quot;i&quot;]</code></td>
<td><code>[&quot;j&quot;, 0, 1]</code></td>
</tr>
<tr>
<td><code>mystery[0]</code></td>
<td><code>undefined</code></td>
</tr>
<tr>
<td><code>mystery.ii.length</code></td>
<td>2</td>
</tr>
<tr>
<td><code>mystery[&quot;i&quot;][0].length</code></td>
<td>1</td>
</tr>
</tbody>
</table>