CSE 154: Web Programming — Summer 2019

Practice Exam 1 Example C | **Key**

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).

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 id, 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.

Question	**Score**	**Possible**
HTML Validation		
CSS and the DOM		
JS/DOM/UI		
JS/Animations		
Short Answer		

1. What's wrong with my HMTL?

Consider the following HTML:

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <script src="index.js"></script>
      <link href="styles.css" rel="stylesheet"/>
   </head>
   <body>
      <h1 "Best page ever!" /> <!-- 1. h1 is not self-closing; cannot have text inside of tag
-->
      <div>
         <a>Check out this cool page: <!-- 2. Missing closing </a> -->
           <href>http://www.pointerpointer.com</href> <!-- 3., 4. href should be attribute in
<a>; <href> is not a tag -->
         </span> <!-- 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 http://www.pointerpoint.com

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

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

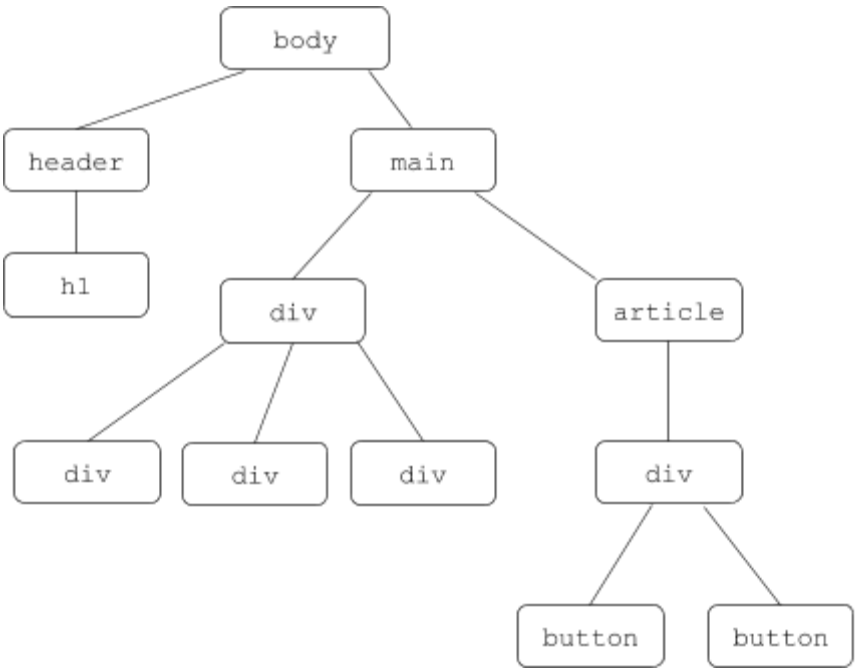
5. Missing open ; fix could be to remove (or include open span, though unnecessary)

3. (CSS) r/css_irl

Part A Solution:

```
body {
  font-family: Helvetica, sans-serif;
}

h1 {
  text-align: center;
}

/* main content layout (to put buttons under lightpost and center page) and
   lightpost layout */
main, main > div {
  display: flex;
  flex-direction: column;
  align-items: center;
}

main > div {
  background-color: black;
  justify-content: space-evenly;
  height: 450px;
  width: 180px;
}

/* circles in lightpost */
div div {
  border-radius: 50%;
  height: 120px;
  width: 120px;
}

button {
  font-weight: bold;
  margin: 10px;
  padding: 10px;
  width: 80px;
}

#red {
  background-color: red;
}

#green {
  background-color: green;
}

#yellow {
  background-color: yellow;
}
```

Part B Solution:



3. (JS/Timers) Street Light Simulator

Note: There are a few possible solutions to this problem, we have provided two common ones below:

```
// Solution with setInterval
(function() {

  /** Part A) */
  const LIGHTS = ["red", "red", "green", "green", "yellow"];
  let timer = null;
  let pos = 0;
  window.addEventListener("load", initialize);

  /** Part B)*/
  function switchColor(color1, color2) {
    id(color1).classList.add("off");
    id(color2).classList.remove("off");
  }

  /** Part C) */
  function initialState() {
    id(LIGHTS[pos % LIGHTS.length]).classList.add("off");
    pos = 0;
    id("red").classList.remove("off");
    clearInterval(timer);
    timer = null;
    id("start").disabled = false;
    this.disabled = true;
  }

  /** Part D) */
  function initialize() {
    id("green").classList.add("off");
    id("yellow").classList.add("off");
    id("start").addEventListener("click", startLight);
    id("reset").addEventListener("click", reset);
  }

  /** Part E) (other functions) */
  function startLight() {
    timer = setInterval(changeLight, 500);
    this.disabled = true;
    id("reset").disabled = false;
  }

  function changeLight() {
    let c1 = LIGHTS[pos % LIGHTS.length];
    let c2 = LIGHTS[(pos + 1) % LIGHTS.length];
    switchColor(c1, c2);
    pos++;
  }
})(); // end module
```

```
// Solution with setTimeout
(function() {

  /** Part A) */
  const LIGHTS = ["red", "green", "yellow"];
  let timer;
  let pos = 0;
  window.addEventListener("load", initialize);

  /** Part B)*/
  function switchColor(color1, color2) {
    id(color1).classList.add("off");
    id(color2).classList.remove("off");
  }

  /** Part C)*/
  function initialState() {
    id(LIGHTS[pos % LIGHTS.length]).classList.add("off");
    pos = 0;
    id(LIGHTS[pos]).classList.remove("off");
    clearInterval(timer);
    timer = null;
    id("start").disabled = false;
    this.disabled = true;
  }

  /** Part D) */
  function initialize() {
    id("start").addEventListener("click", redLight);
    id("reset").addEventListener("click", reset)
    id("green").classList.add("off");
    id("yellow").classList.add("off");
  }

  /** Part E) (other functions) */
  function redLight() {
    timer = setTimeout(function() {
      switchColor("red", "green");
      greenLight();
    }, 1000);
  }
  function greenLight() {
    timer = setTimeout(function() {
      switchColor("green", "yellow");
      yellowLight();
    }, 1000);
  }
  function yellowLight() {
    timer = setTimeout(function() {
      switchColor("yellow", "red");
      redLight();
    }, 500);
  }
 })(); // end module
```

4. Define-It!

Solution:

```
(function() {
  “use strict”;

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

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

    function updateEntries() {
        id("entries").removeChild(this);
        if (!(qsa("li").length || id("current-entries").classList.contains("hidden"))) {
            id("current-entries").classList.add("hidden");
        }
    }
})();
```

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:

Issue A and justification: **There should not be a script tag inside of the HTML; this is poor separation of content (HTML) and behavior (JS)** Better alternative: **Move code inside of the script tag into a linked JS file (within a module)**
Issue B and justification: **innerText should be used rather than innerHTML; HTML tags should be added with JS using document.createElement("tagname"); we're also just adding text, not HTML here.** Better alternative: **use** `document.getElementById("result").innerText = "you clicked the button!";`

5. (JSON Mystery)

Statement	**Value**
`mystery["i"]`	`["j", 0, 1]`
`mystery[0]`	**`undefined`**
`mystery.ii.length`	**`2`**
`mystery["i"][0].length`	**`1`**