

Practice Exam 1: Example A

Note that this practice exam includes problems from various past exams. The actual exam format for Exam 1 of Summer Quarter will not be in exactly the same format (but it will be close).

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.
- 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		
CSS		
Short Answer		
JS/DOM		
JS/Animations		

1. What's Wrong with my HTML?

```
<!DOCTYPE html>
<head>
  <h1>Mowgli's Magical Muffins</h1>
  <link src="mypage.css" rel="stylesheet" />
</head>
<body>
  <p>For Doggies' Best Friends:</p>
  <ul>
    <li>Multi-grain Melody</li>
    <li>Merry-Mint-Chip</li>
  </ul>
  For Doggies:
  <ul>
    <li>The Malt-ese</li>
    <li>Malamint Magic</li>
    <li>Meow Meows</li>
  </ul>
</body>
</!DOCTYPE html>
```

This HTML document won't validate, and would generate errors and warnings in the W3C Validator. However, it is possible to make 5 modifications to the HTML to make it pass validation. Each modification might result in multiple text "changes" to the HTML document, but is considered one modification because it is addressing the same root problem. Indicate the 5 modifications we need in order to make it pass validation. Write directly on the HTML.

Below, briefly describe the changes that you made, and why you had to make each change in order to validate. If it is unclear what changes go with which explanations, feel free to number your changes on the HTML. No need for an essay — 10 words or less should be plenty.

1. _____
2. _____
3. _____
4. _____
5. _____

2. You Selected the Right Class.

Consider the following HTML:

```
<html>
  <heading>
    <title>CSE 154 Course Web Page</title>
  </heading>
  <body>
    <header id="title-1">
      <h1 id="title-2"><em id="em-1">All the CSE 154 Course Stuffff Ever</em></h1>
    </header>
    <p id="subtitle-1">Topics:</p>
    <ul id="list-1">
      <li id="topic-1">What is the Internet</li>
      <li id="topic-2">How to do the Internet</li>
      <li id="topic-3">How to make the Internet</li>
      <li id="topic-4">
        Make cool projects:
        <ol id="list-2">
          <li id="hw-1">Make Pies</li>
          <li id="hw-2">Watch Lion King</li>
          <li id="hw-3">Read <em id="em-2">rly rly rly</em> fast</li>
          <li id="hw-4">Push squares around</li>
          <li id="hw-5">Catch 'em all!</li>
        </ol>
      </li>
    </ul>
    <div id="div-1">
      Our course mascot!</img>
    </div>
  </body>
</html>
```

Write the ID's of the elements selected by each of the given selectors:

1. p

2. ol li

3. li em

4. ul > li

5. li li

3. Short Answers

1. What is the difference between inline elements and block elements?

2. Why do we always want to include an alt attribute on img tags?

3. What's the difference between margin, borders, and padding? (You may provide a labeled diagram)

4. Why is it important to specify multiple font styles for the same element in your CSS? (e.g., `font-family: Helvetica, Arial, sans-serif;`)

5. Why is it important to use the module pattern in JavaScript?

6. What is the difference between `setInterval` and `setTimeout`?

7. Consider the following JSON object:

```
let miniJSON = {  
  "foo" : ["b", 1, 2],  
  "bar" : 0,  
  "FOO" : "Foo?"  
};
```

For each of the following statements, write the value that would be returned (include "" around any string values; if any expression would result in an error, write error. If any expression would return the value undefined or null, specify this as your answer.

a. `miniJSON.foo` : _____

b. `miniJSON["FOO"]` : _____

c. `miniJSON["FOO"][1]` : _____

d. `miniJSON[foo]` : _____

e. `miniJSON["foo"].length` : _____

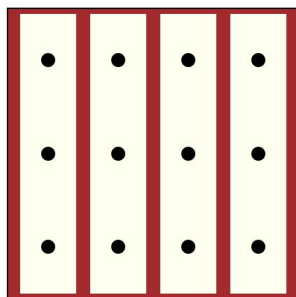
4. JS/DOM/Events (12 pts): Planting DOM Trees Tulips!

It's Spring time and we're going to plant some tulips in our "CSE154 Zen Garden"! On this page, we provide the reference HTML and screenshots. On the following pages, you will find instructions for the implementation of your JS.

```
<!-- start of HTML -->
<main>
  <div id="garden">
    <div class="row">
      <div class="spot seed"></div>
      <div class="spot seed"></div>
      <div class="spot seed"></div>
    </div>
    <div class="row" >
      <div class="spot seed"></div>
      <div class="spot seed"></div>
      <div class="spot seed"></div>
    </div>
    <div class="row">
      <div class="spot seed"></div>
      <div class="spot seed"></div>
      <div class="spot seed"></div>
    </div>
    <div class="row">
      <div class="spot seed"></div>
      <div class="spot seed"></div>
      <div class="spot seed"></div>
    </div>
    <div class="row">
      <div class="spot seed"></div>
      <div class="spot seed"></div>
      <div class="spot seed"></div>
    </div>
  </div>
  <button id="grow-all" disabled>Grow Everything</button>
</main>
<!-- Rest of HTML -->
```

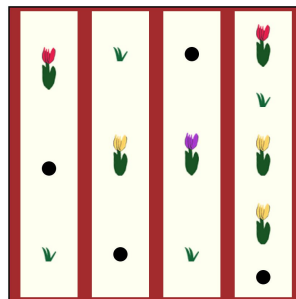
Example Screenshots:

CSS 154 Zen (Tulip) Garden



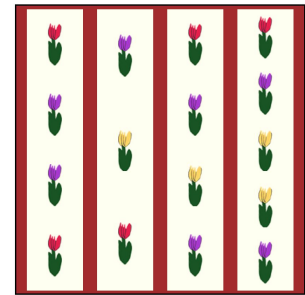
Grow Everything
All images shared under

CSS 154 Zen (Tulip) Garden



Grow Everything
All images shared under

CSS 154 Zen (Tulip) Garden



Grow Everything
All images shared under

Figure 1: Starting the simulation (with JS from Problem 5 attached)

Figure 2: The garden showing three states: seed, sprout, and ___-tulip, and a few new spots added.

Figure 3: The garden is fully grown (all tulips), with more added.




Overview of HTML and provided classes

The Garden

The HTML has a `#garden` containing 4 `.row` divs of "plantable" soil. While these are called rows (a common term for gardens), these are displayed as vertical columns.

The Rows

Each `.row` starts with 3 `.spot` divs (initially in the `.seed` stage), where a "spot" represents the spot allocated for a growing tulip plant having exactly one of the following classes depending on its stage (each `.spot` should always have exactly one of these 5 classes):

Stage/appearance	Class(es)	Description
1 (seed) 	<code>.seed</code>	The initial stage of a tulip plant (also known as a "tulip bulb").
2 (sprout) 	<code>.sprout</code>	The stage after seed, representing a "sprouted" seed. A <code>.spot</code> has this class exactly when a seed is grown.
3 (tulip) 	<code>.red-tulip</code> (shown on left) <code>.purple-tulip</code> , <code>.yellow-tulip</code>	The final stage of a tulip plant, with exactly one of three possible colors. Once a <code>.spot</code> has one of the tulip classes, it will always stay in this stage.

You will be implementing JS to add functionality to "grow" the garden as the user interacts with different elements.

Behavior Details:

- When a `.spot` in the `#garden` is clicked, it should grow to the next stage (until the final tulip stage) **using the provided grow helper method**. A `.spot` grows depending on its stage, as described in the table in the previous Problem 4/5 Overview page. As a reminder:
 - A `.seed` grows to `.sprout`
 - A `.sprout` grows to a random colored tulip (`.red-tulip`, `.purple-tulip`, or `.yellow-tulip`)
 - A `.spot` with one of the 3 tulip classes will not grow further
- Each `.row` should respond to a `dblclick` event by adding a new `.spot` at the bottom of the row (again, oriented as a column) starting in the "seed" stage. These newly-added `.spots` should also be able to grow when clicked as specified.
- If the `#grow-all` button is clicked, every spot in the garden will grow to the next stage (again, tulips won't grow further). This button is initially disabled in the HTML and should be enabled when the when the page is done loading.

You are provided one function, `grow`, to help you with the full implementation of this problem. On the next page, finish the implementation of the program to complete the specified behavior.

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

  /**
   * Helper function that will update the classList for a .spot div to grow to the
   * next 'stage'. A .spot having one of the three .___-tulip classes remains unchanged.
   * @param {Object} item - the .spot div to change from .seed to .sprout, or .sprout
   * to a randomly-colored tulip (.___-tulip).
   */
  function grow(item) {
    /* Details of function not provided */
  }

  window.addEventListener("load", init);

  // You must implement the rest of this program on the next page.
```

// You must implement the rest of the JS program for Problem 4 on this page:

})();

5 JS/Timers (10 pts)

You may choose to complete either 4a (code execution) or 4b (code writing) for this problem. If you do complete both, we will count the higher of the two towards your midterm grade.

5a. Code Execution (10 pts): Duck, Duck, Goose!

In this problem, there are three variants of a function (goose, gooseA, gooseB) which are used in a program using delays/intervals. For each function variant, there is a table with the full program having that function variant and 8 lines of console output you are to fill in. The first program, having the "goose" function, is completed for you to reference as an example of the expected answer format.

For Program A and Program B, similarly fill each row of the Console Output answer area with the first 8 statements logged to the console.

4a. Example Program (0 pts): (Given as reference for expected output):

JS Program	Console Output:	
<pre>(function () { let t1 = null; let t2 = null; let duckCount = 0; window.addEventListener("load", init); function duck() { duckCount += 1; console.log(duckCount + " ducks"); } function init() { /* BEGIN CODE SPECIFIC TO EXAMPLE */ t1 = setInterval(duck, 300); t2 = setTimeout(goose, 800); } function goose() { console.log("goose"); duckCount = 0; clearInterval(t1); t1 = null; t2 = setTimeout(goose, 1000); } })();</pre>	Line 1	<u>1 ducks</u>
	Line 2	<u>2 ducks</u>
	Line 3	<u>goose</u>
	Line 4	<u>goose</u>
	Line 5	<u>goose</u>
	Line 6	<u>goose</u>
	Line 7	<u>goose</u>
	Line 8	<u>goose</u>

5a. Program A (5 Points):

JS Code	Console output:	
<pre> (function () { let t1 = null; let t2 = null; let duckCount = 0; window.addEventListener("load", init); function duck() { duckCount += 1; console.log(duckCount + " ducks"); } function init() { /* BEGIN CODE SPECIFIC TO PROGRAM A */ t1 = setInterval(duck, 300); t2 = setTimeout(gooseA, 800); } function gooseA() { console.log("goose"); duckCount = 0; t1 = null; clearInterval(t1); t2 = setTimeout(gooseA, 800); } })(); </pre>	Line 1	
	Line 2	
	Line 3	
	Line 4	
	Line 5	
	Line 6	
	Line 7	
	Line 8	
Room for Scratch Work		

5a. Program B (5 Points):

JS Code	Console output:	
<pre>(function () { let t1 = null; let t2 = null; let duckCount = 0; window.addEventListener("load", init); function duck() { duckCount += 1; console.log(duckCount + " ducks"); } function init() { /* BEGIN CODE SPECIFIC TO PROGRAM B */ t1 = setInterval(duck, 300); t2 = setTimeout(gooseB, 1000); } function gooseB() { console.log("goose"); clearInterval(t1); t1 = setInterval(duck, 100 * duckCount); t2 = setTimeout(gooseB, 800); } })();</pre>	Line 1	
	Line 2	
	Line 3	
	Line 4	
	Line 5	
	Line 6	
	Line 7	
	Line 8	
	Room for Scratch Work	

5B. JS/Timers - Code Writing (10pts): The Final Countdown!

Suppose we have a button `#btn` on a page linked to the JS program described below. In this problem, a complete (uninterrupted) countdown uses `console.log` to print from 10 to 0 (inclusive), with each number in the countdown printed on a new line at 1 second intervals (i.e. a complete, uninterrupted countdown will log 11 statements over 11 seconds).

Finish writing the program such that whenever `#btn` is clicked:

- If no countdown is currently active, a new countdown is started. Otherwise the active countdown is stopped. Any future countdown starts back at 10 (when the `#btn` is clicked again).
- If a countdown completes (reaches 0 as its last output), no further output should be made until the `#btn` is clicked again.

In Example 2, when `#btn` clicked the first time (countdown starts) and then a second time (4 seconds after the first click), the current countdown is stopped. When the button is then clicked a third time, a new countdown is started back at 10. Below are two examples of expected sequences (console output and events), where `<... click>` is just a placeholder to indicate a click event on the button and `// ...` indicates nothing should happen until the next click.

Example 1	Example 2	Your Solution
<pre><first click> 10 9 8 7 6 5 4 3 2 1 0 // ...</pre>	<pre><first click> 10 9 8 7 <second click> // ... <third click> 10 9 8 7 6 5 4 3 2 1 0 // ...</pre>	<pre>(function () { // TODO: Add any module-global(s) here window.addEventListener("load", init); function init() { id("btn").addEventListener("click", handleClick); } // TODO: Finish this function (you may add a helper // function below if you'd like) function handleClick() { } })();</pre>