CSE 154: Web Programming — Autumn 2018

Practice Final Exam 2 (based on 18sp final)

Name:

UWNet ID: @uw.edu

TA (or section):

Rules:

- You have 110 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**
CSS		10
Short Answer		20
SQL		10
JS/DOM/Timers		20
JS/AJAX		20
PHP Web Service		20
Extra Credit		1

1. CSS (Cute, Slow, and Sleepy)

In this problem, you will 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).

```
<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>
```

Appearance Details:

- The `main` element is 40% of the page's width. The text and sloth image are centered on the page.
- The `h1` text has a font family of Helvetica, falling back to Arial if Helvetica is not available on the system, falling back to the system's sans-serif default font if Arial is also not available.
- The background color of the outermost circle is `sienna` and the background color of the inner circle containing the "eyes" and "nose" (which each have a black background) of the sloth is `peru`.
- The two div circles have a 1px-width solid black border and a border radius of 50%.
- The outermost circle has a width and height of 150px. Its inner circle has a width and height of 110px.
- The `#t` span is positioned in the center of the face and has a height of 25px - the `#o` and `#h` spans both have a height of 20px and touch the very left and right borders of the parent `#l` div, respectively. All span elements have a width of 20px and a border radius of 50%.

Write your CSS below:

2. Short Answer

a) JavaScript Events

For the following JS program, label the `// __` following each `console.log` statement with 1, 2, 3, or 4, corresponding to the relative order in which that statement will print (where 1 indicates the first statement printed).

```
console.log("Foo");          // ____
(function() {

  console.log("Bar");        // ____
  window.onload =
  pageLoad; foo();

  function pageLoad() {
   console.log("Baz");       // ____

  }

  function foo() {
    console.log("Mumble");   // ____
  }
})();
```

b) JSON Mystery

Consider the following JSON definition:

```
let mystery = {
  "i" : ["j", 0, 1],
  "ii" : "ii",
  "I" : "i"
};
```

Write the JavaScript value that would be returned for each of the following statements. Include "" around any string values, and make sure to consider possibly undefined or null values.

Statement	**Return Value**
`mystery["i"]`	
`mystery[0]`	
`mystery.ii.length`	
`mystery["i"][0].length`	

c) PHP Database Connections

Consider the following PHP code, where `$db` is a defined PDO object:

```
$str = "INSERT INTO users (username, date, email, password)
        VALUES ('$username', NOW(), '$email', '$password');";
$rows = $db->exec($str);
```

Briefly explain why this method of using the PDO object is insecure, as well as what change(s) you would need to make it secure given what we've covered in lecture (you may cross out/modify the provided code to indicate the changes).

d) Validation Methods

What is one advantage of validating user input on the client (HTML5 or JS) vs. on the server (PHP)?

What is one advantage of validating user input on the server as opposed to on the client?

e) Technology Trade-offs: indexDB vs. Dexie

Briefly explain the relationship between indexDB and Dexie and why you might want to use one over the other.

f) Web Service Trade-offs: Plain Text vs. JSON

From the client perspective, what is an advantage of working with a JSON response over one in plain text format?

g) Storage Technologies

In class we learned about a number of storage technologies including cookies, sessions, localStorage, sessionStorage, and indexDB. Of all of these technologies, circle the most appropriate choice for the following use cases. Each technology will be a ”best choice” for one use case. You will get both 2pts for this question if you correctly answer 4 of the 5 use cases.

i) Your development team wants to keep track of emojis that are used on the client only.

localStorage	sessionStorage	indexDB	cookies	sessions

ii) You want your site to temporarily retain large pieces of information that are being downloaded from a website, but most of your users primarily use mobile phones to access the site.

localStorage	sessionStorage	indexDB	cookies	sessions

iii) Storing the status a user has successfully logged into a website, but ensuring the log in status is deleted when they close the browser tab.

localStorage	sessionStorage	indexDB	cookies	sessions

iv) You want to use JavaScript to store a value in the browser that is accessible from the server.

localStorage	sessionStorage	indexDB	cookies	sessions

v) Securely storing a user has logged in so they can surf through a number of links in a site without having to log in each time the page changes.

localStorage	sessionStorage	indexDB	cookies	sessions

h) Regex

Circle all of the following strings which match the regex: **`/^\[-?\d(,\s*-?\d)+\]$/`**

- `[1, 2, 3]`
- `[]`
- `[154]`
- `[\d]`
- `0`
- `[-1]`
- `][`
- `[?]`

Circle all of the following strings which match the regex: **`/^<img src=".+.(gif|jpg|png)" alt=".+"\s?\/?>$/i`**

- `^<img src=.+.(gif|jpg|png) alt=.+>$`
- `<img src="foo.gif" alt="A foo!">`
- `img src="foo.gif" alt="A foo!"`
- `<img src="puppy.png" alt="A puppy" />`
- `<img src="a.jpg" alt="A letter a" />`
- `<img src="FOO.GIF" alt="A foo!">`

3. SQL Around the World (10 pts)

Recall the following tables in the world database:

world:

code	name	continent	independence_year	population	gnp	head_of_state	...
AFG	Afghanistan	Asia	1919	22720000	5976.0	Mohammad Omar	...
NLD	Netherlands	Europe	1581	15864000	371362.0	Beatrix	...
...							

countries

Other columns: region, surface_area, life_expectancy, gnp_old, local_name, government_form, capital, code2

country_code	language	official	percentage
AFG	Pashto	T	52.4
NLD	Dutch	T	95.6
...			

languages

id	name	country_code	district	population
3793	New York	USA	New York	8008278
1	Los Angeles	USA	California	3694820
...				

cities

a.) Shared Official Languages

Write a SQL query that returns the **languages** that are official in at least two different country codes. Order your results by language name in ascending order and do not include duplicate languages in your result.

Expected Results:

```
+-------------+
| language    |
+-------------+
| Aimará      |
| Arabic      |
| Chinese     |
| ...         |
+-------------+
23 total rows returned. (2 ms)
```

Write your SQL query here:

b.) Cities and Languages in Countries with ”Island” Names

Write a SQL query to select the names of all cities in a country with the word ”Island” in the country’s name, as well as official languages spoken in that country. Your result should include the city **name**, the **continent** name, and the official **language** names as columns. Note that some countries have more than one official language. Order your results by city name (ascending).

Expected results:

```
+--------------------+---------------+-------------+
| name               | continent     | language    |
+--------------------+---------------+-------------+
| Avarua             | Oceania       | Maori       |
| Bantam             | Oceania       | English     |
| Charlotte Amalie   | North America | English     |
| Cockburn Town      | North America | English     |
| Dalap-Uliga-Darrit | Oceania       | English     |
| Dalap-Uliga-Darrit | Oceania       | Marshallese |
| …                  | …             | …           |
+--------------------+---------------+-------------+
16 rows returned. (3 ms)
```

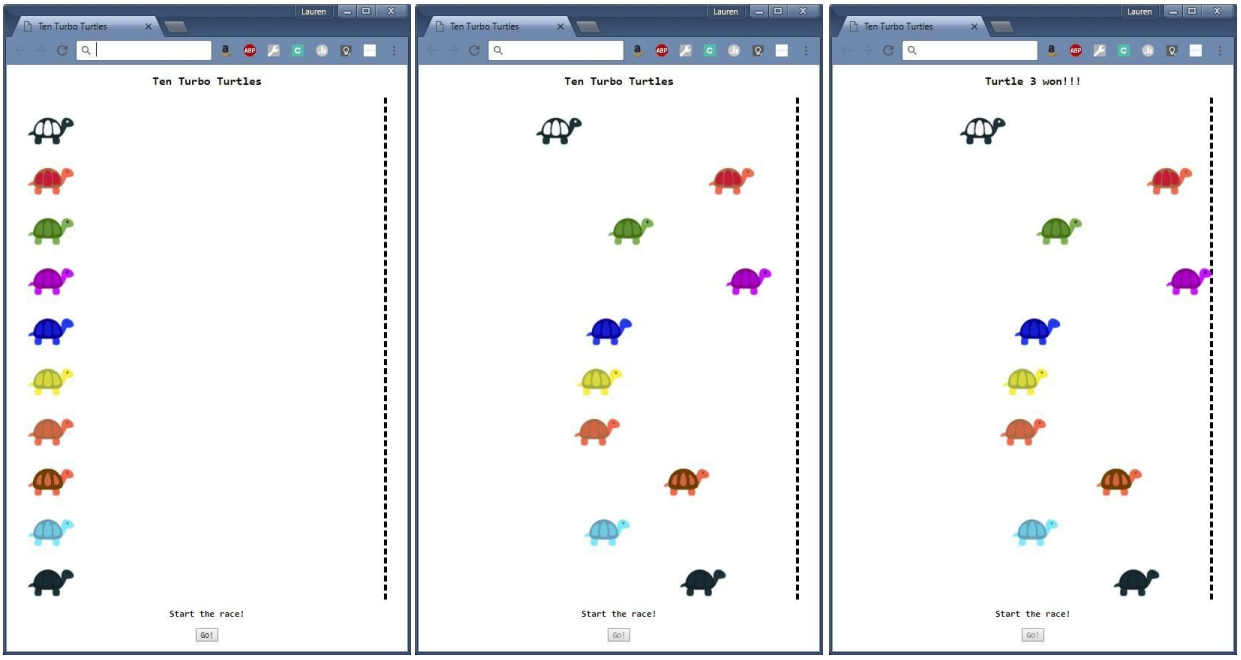
Write your SQL query here:

4. Turbo Turtles: The Championship Finale

They're back! And they're hankering for more action. The two turbo turtles have brought their friends to the race track to see who will crowned the winner.

You are provided the skeleton HTML (on the next page) and will write **parts** of the JavaScript program to add interactivity and animation to this site. You may assume your JavaScript is running in a file successfully linked in from the HTML.

The following are examples of what the page looks like before, during, and after a race:



View Before Race Starts Middle of race End of Race)

Requirement Details

- The racetrack must be populated with all ten racers.
- Each racer image must be placed in a div that has the class racer. The id for the div is the word "turtle" plus its 0 based position in the starting line up (i.e. `"turtle0" ... "turtle9"`). The images for each racer is the racer's id with the ".png" extension. In other words, if we wrote `turtle0`'s div in HTML, it would look like this:

```
<div class="racer" id="turtle0">
  <img src="turtle0.png" alt="turtle0 image">
</div>
```

- The race starts when a user clicks on the Go! button. After the race starts that button will be disabled and will never be re-enabled in this program.
- When the race starts, all racers should move towards the dashed finish line from left to right. There is a provided function moveRacer(racer) which takes the DOM element of the div of the racer which will do that, check if any of the racers have won, and declare a winner. moveRacer(racer) will call a function that you will write called stopAll() which will end the animation.

- Each racer's speed is decided randomly as an interval value between 1 and 250ms (inclusive). You are provided a function `getRandomValue(min, max)` that takes two parameters (a min and a max) and which will return you a random number from min to max (inclusively). Once the race has started, each turtle maintains their speed for the duration of the race.
- A full credit solution will not include 10 or more module variables.

You will be doing five things in this question:

(a) Declaring the necessary module global(s) to complete the problem.

(b) Writing a function to create a racer's div called `createNewRacer` from the racer's id, image file name and alt text.

(c) Writing the body of the initialize function which is called when the window is loaded.

(d) Write a function startRace to start the race.

(e) Write a function stopAll to stop the animation of all racers.

The following is the `<body>` section of an HTML page for your Ten Turbo Turtles:

```
<body>
  <section id="game">
    <header>
      <h1 id="title">Ten Turbo Turtles</h1>
    </header>
    <div id="racetrack">
    </div>
  </section>
  <section id="start-race">
    <header>
      <h3>Start the race!</h3>
    </header>
    <button id="go">Go!</button>
  </section>
</body>
```

The framework of the JavaScript program is as follows. You will write your code on the pages after.

```
(function() {
  /** Other Module global(s) will be listed here */
  /****** Student will fill this in as part (a) ******/

   /* Override of the window onload function */
   window.addEventListener("load", initialize);

  /**
   * Function to initialize the web page with 10 turtles and a hare
   */
   function initialize() {
      /****** Student will complete this function for part (c) ******/
   }
```

```
// Framework for Problem 4 continued
/**
 * Function to create a new racer with the given id and class racer. The div
 * contains the racer's image, with the alternate text also set.
 * @param id - The id of the div being created.
 * @param image - the file name that contains the image of the racer.
 * @param alt - the alt text for the image.
 * @return the new div with correct class, id that contains the image.
 */
 function createNewRacer(id, image, alt) {
  /****** Student will complete this function for part (b)
                            ******/
 }

 /**
  * Function to start the race
  */
 function startRace() {
   /****** Student will complete this function for part (d) ******/
 }

 /**
 *   Function that will ensure all racers will stop.
 */
 function stopAll() {
   /****** Student will complete this function for part (e) ******/
 }

////////////////////// Helper Functions ////////////////////////////
/**
 * This function returns a random value between min and max inclusive.
   Assumes
 * min is less than max.
 * @param min - the minimum value the random number can take.
 * @param max - the maximum value the random number can take.
 * @return - the random value between min and max inclusive.
 **/
 function getRandomValue(min, max) {
  /* code not displayed here */
 }

/**
 * This function move a racer across the screen. If the racer hits the
 * finish line it will stop the race (call stopAll) and display the results.
 * @param racer - The DOM element racer to move.
 */
function moveRacer(racer) {
  /* code not displayed here */
}
```

```
})(); // See next page for functions you are asked to implement
```

Write your parts of the Javascript code for the Ten Turbo Turtles here. You may assume that the aliases `$(id)`, `qs(sel)` and `qsa(sel)` are defined for you and included as appropriate.

a)

Write the additional module global(s) you will need for this program here:

b)

Write the code to create a turtle racer here:

```
/**
 * Function to create a new racer with the given id and class racer. The div
 * contains the racer's image, with the alternate text also set.
 * @param id - The id of the div being created.
 * @param image - the file name that contains the image of the racer.
 * @param alt - the alt text for the image.
 * @return the new div with correct class and id that contains the image.
 */
function createNewRacer(id, image, alt) {









}
```

c)

Assume the code in Part b) functions correctly. Write the code for the initialize function here:

```
/**
 * Function to initialize the web page with 10 turtles
 */
function initialize() {









}
```

d)

Write your code for the `startRace` function here:

```
/**
 * Function to start the race
 */
function startRace() {









}
```

e)

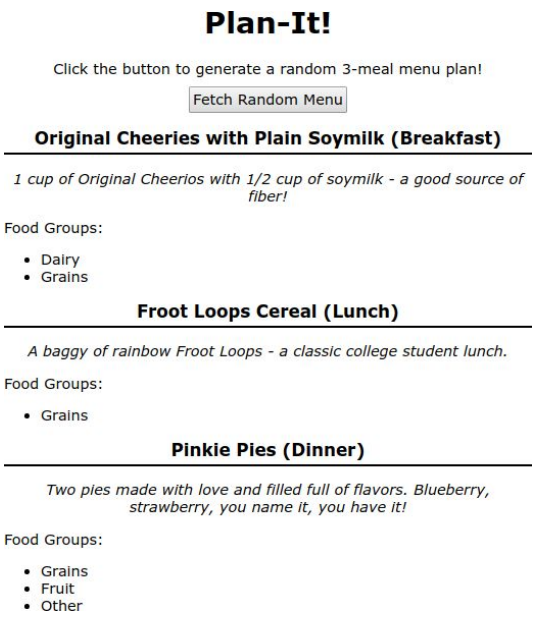
Write your code for the `stopAll` function here:

```
/**
 * Function that will ensure all racers will stop.
 */
function stopAll() {









}
```

5. Plan-It! Fetching You Meals a Day at a Time (20 pts)

In this question, you will write JavaScript to implement a small Meal Planner web page called Plan-It!. For simplicity, we will consider a ”full day meal plan” as a breakfast, lunch, and dinner (the 3 standard meal types). Below are two screenshots of the Plan-It! page:

Initial View



After Fetching Random Full-Day Menu

Implementation Requirements

The web service your JS will fetch data from is called planit.php (the same web service you will implement in Problem 6). Here, you will fetch using **only one** of the two request types you implement in Problem 6.

Clicking the `#day-btn` should make a request to planit.php with `mode=day` to fetch a random full day meal plan (one option for each of breakfast, lunch, and dinner). The response JSON format will be in the following format (example):

```
{ "breakfast" : {
    "name" : "Blueberry Oatmeal",
    "description" : "One cup of hot oatmeal with 1/4 cup of fresh blueberries.",
    "food-groups" : ["Grains", "Fruit"]
  },
  "lunch" : {
    "name" : "Froot Loops",
    "description" : "A baggy of rainbow Froot Loops - a classic college student lunch.",
    "food-groups" : ["Grains"]
  },
  "dinner" : {
   "name" : "Spaghetti with Tomato Sauce",
    "description" : "Two pies made with love and filled full of flavors. Blueberry,
                     strawberry, you name it, you have it!",
    "food-groups" : ["Grains", "Fruit", "Other"]
  }
}
```

Upon success, remove the .hidden class from the #day-results (you will not ever need to add it back) and use the response JSON to populate each of the #breakfast, #lunch, and #dinner aside elements on the page such that:

- The span.name is populated with the corresponding meal name.
- The p.description is populated with the corresponding meal description
- The ul.food-groups is populated with a list of the food groups for that item (one or more food groups may be in the ”food-groups” array in the JSON).

A client should be able to click the #day-btn multiple times to get a new randomly-generated 3-meal menu (previous menu results should be replaced as a result).

Note: You do not need to know any CSS used by the page other than the .hidden class which should be added/removed as previously specified.

Provided HTML:

```
<body>
  <h1>Plan-It!</h1>
  <p>
    Click the button to generate a random 3-meal menu plan!
    <button id="day-btn">Fetch Random Menu</button>
  </p>

  <section id="day-results" class="hidden">
    <article id="breakfast">
      <h2><span class="name"></span> (Breakfast)</h2>
      <p class="description"></p>
      <p>Food Groups:</p>
      <ul class="food-groups"></ul>
    </article>
    <article id="lunch">
      <h2><span class="name"></span> (Lunch)</h2>
      <p class="description"></p>
      <p>Food Groups:</p>
      <ul class="food-groups"></ul>
    </article>
    <article id="dinner">
      <h2><span class="name"></span> (Dinner)</h2>
      <p class="description"></p>
      <p>Food Groups:</p>
      <ul class="food-groups"></ul>
    </article>
  </section>
</body>
```

Write your JS solution below. You may assume that the `checkStatus` function and the aliases `$(id)`, `qs(el)`, and `qsa(sel)` are defined for you and are included as appropriate.

```
(function() {

})();
```

6. *Serving* Up Some Meal Ideas (20 pts)

In this question, you will *implement* the PHP web service (planit.php) you were asked to use in the previous problem (JavaScript/AJAX).

Directory Structure and File Format Details

Your web service will be located in the same level as three directories corresponding to each of the standard meals in a day (breakfast, lunch, and dinner). Inside of each of these three directories are txt files for different food options common for that meal.

Meal Type Directory Structure:

```
breakfast/
  <foodoption>.txt
  <foodoption>.txt
  ...
              lunch/
  <foodoption>.txt
  <foodoption>.txt
  ...
             dinner/
  <foodoption>.txt
  <foodoption>.txt
  ...
```

Example Directory Contents:

```
  breakfast/ banana.txt
  blueberry-oatmeal.txt
  ...
lunch/
  froot-loops.txt spinach-salad.txt
  ...
dinner/
  brown-rice-with-veggies.txt
  pinkie-pies.txt
  …
```

Inside of each directory (which you may assume is non-empty), each txt file has three lines:

.txt File Content Format:

```
name
description
food groups
```

Example File Contents: (blueberry-oatmeal.txt)

```
Blueberry Oatmeal
One cup of hot oatmeal with 1/4
cup of fresh blueberries.
Grains Fruit
```

The name is the full name of the food option, the description is a short description, and the food groups lists any major food groups associated with the food item.

Web Service Implementation

Your web service should support the following two GET requests (all query parameters are case-insensitive):

`planit.php?mode=meal&type=<mealtype>`

This request should return a plain text response of all food options available on in the corresponding <mealtype> directory. Each food option should be output on a new line in the format of name: description, where name corresponds to the first line of the .txt file for that food item and description is the second line. The order of lines in the output not matter.

For example, a request to `planit.php?mode=meal&type=breakfast` might output:

```
Banana: A good source of potassium on the go.
Blueberry Oatmeal: One cup of hot oatmeal with 1/4 cup of fresh blueberries.
Cheerios Cereal with Plain Soymilk: One cup of original Cheerios with 1/2 cup
of soymilk.
```

Note that this response corresponds to three breakfast meal options in the breakfast directory, but your request should work for any number of txt files that may be found in the directory.

`planit.php?mode=day`

This request should return a JSON response containing a random option from each meal type on a new line. One example response may look like the following (identical to the example we gave in Problem 5):

```
{ "breakfast" : {
    "name" : "Blueberry Oatmeal",
    "description" : "One cup of hot oatmeal with 1/4 cup of fresh blueberries.",
    "food-groups" : ["Grains", "Fruit"]
  },
  "lunch" : {
    "name" : "Froot Loops",
    "description" : "A baggy of rainbow Froot Loops - a classic college student lunch.",
    "food-groups" : ["Grains"]
  },
  "dinner" : {
   "name" : "Spaghetti with Tomato Sauce",
    "description" : "Two pies made with love and filled full of flavors. Blueberry,
                     strawberry, you name it, you have it!",
    "food-groups" : ["Grains", "Fruit", "Other"]
  }
}
```

Error Handling

Your web service should handle the following errors with 400 error codes (in order of highest priority to lowest priority). If any error occurs, only output the respective error message in plain text.

- If missing the mode parameter or the mode parameter is passed with a value other than meal or day, output an error with a message ”Please pass a mode of meal or day”.
- If instead mode=meal is passed but no type parameter is passed, output a message ”Missing type parameter for meal request”

You do not need to handle the case when `mode=meal` is passed with a type parameter having a value other than breakfast, lunch, or dinner.

You will write your solution to `planit.php` in two parts:

Part A:

Write a function that returns an associative array based on the passed `$meal` where you may assume `$meal` is ”breakfast”, ”lunch”, or ”dinner”. The returned array should contain information found from a random .txt file in the `$meal` directory. For example, if `$meal` has the value ”breakfast”, the function might return the following array corresponding to the contents in the breakfast/blueberry-oatmeal.txt file (part of the example response given on the previous page):

```
{
  "name" : "Blueberry Oatmeal",
  "description" : "One cup of hot oatmeal with 1/4 cup of fresh blueberries.",
  "food-groups" : ["Grains", "Fruit"]
}
```

Implement your function here:

```
function get_meal_data($meal) {









}
```

Part B:

Assume your function from Part A works. In this part, you will write the rest of the planit.php web service, handling both request types specified on the previous page. Hint: You will find it helpful to use the `get_meal_data` function for the `mode=fullday` to get each of the breakfast, lunch, and dinner options returned as the response.

```
<?php
# Write your code here
```

```
# continue your part B code here
```

```
function get_meal_data($meal) {
  # assume this function works as specified in Part A
}

?>
```

X. Extra Credit

For this question, you can get 1 point of extra credit (demonstrating at least 1 minute's worth of work and being appropriate in content). You can draw or write your answers in text.

If you could give your TA a surprise 1 week summer vacation anywhere in the real (or imaginary) world, what would it be?