CSE 154: Web Programming Spring 2019

Practice Final Exam 2

Name:

UWNet ID: @uw.edu

TA (or section):

Rules:

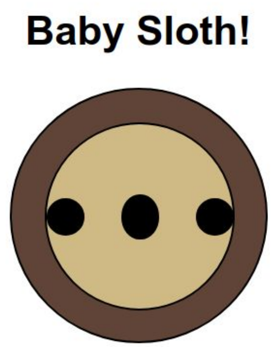
- You have 110 minutes to complete this exam.
- You will receive a deduction if you keep working after the instructor calls for papers.
- You may not use any electronic or computing devices, including calculators, cell phones, smartwatches, and music players.
- Unless otherwise indicated, your code will 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
JS/DOM/Timers		15
JS/AJAX		20
PHP Web Service		20
PHP and SQL with PDO		15
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]`
- `][`
- `[?]`

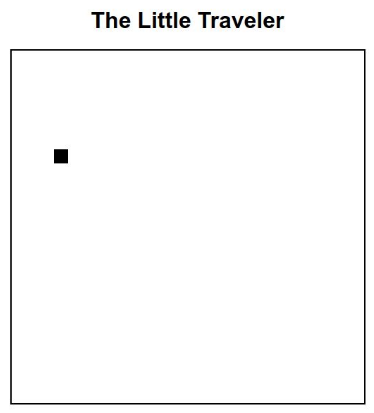
Circle all of the following strings which match the regex: **`/^<img src=″.+.(gif|jpg|png)″ alt=″.+″\s?\/?>$/i`**

- `^<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. The Little Traveler (JS)

Given the HTML and CSS on the following page, write the JavaScript code that adds a `.little-box` div to the top left corner of the `#box` div, and moves the little box inside of the `#box` 20px up, down, left, or right randomly every 100ms. The little box may only move to a position that is inside of the boundaries of the parent `#box`. Note that the `#box` parent has a width and height of 500px, and the `.little-box` div has a width and height of 20px (without any border):

To the right is a screenshot of the little box during an animation. Write your JavaScript solution on the next page following the provided HTML and CSS:

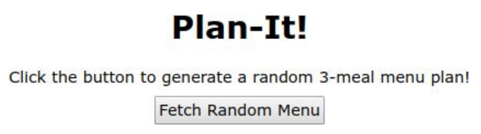


```html
<!DOCTYPE html> <!-- HTML for Problem 3 -->
<html lang="en">
    <head>
        <link href="traveler.css" rel="stylesheet" />
        <script src="traveler.js"></script>
    </head>
    <body>
        <h1>The Little Traveler</h1>
        <div id="box"></div>
    </body>
</html>
```

```css
/* CSS for Problem 3 */
h1 {
  font-family: Helvetica, Arial, sans-serif;
  text-align: center;
}
.little-box {
  background-color: #000;
  height: 20px;
  position: absolute;
  width: 20px;
}
#box {
  border: 2px solid black;
  height: 500px;
  margin: auto auto;
  position: relative;
  width: 500px;
}
```

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

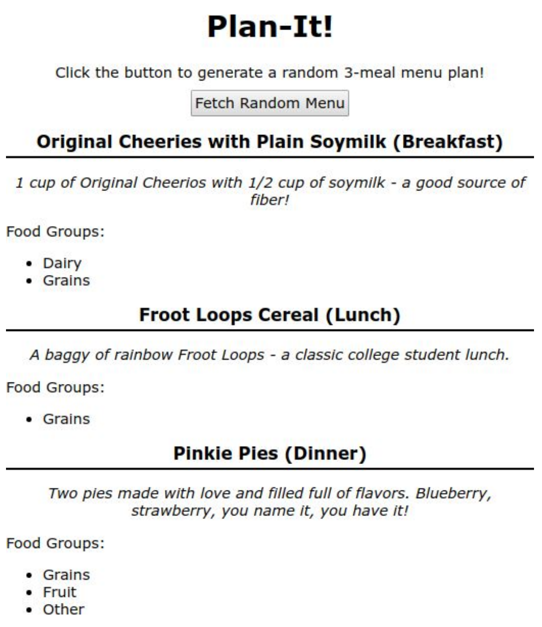
In this question, you will write JavaScript to implement a small Meal Planner web page called Plan-It!. For simplicity, we will consider a ”full day meal plan” as a breakfast, lunch, and dinner (the 3 standard meal types). Below are two screenshots of the Plan-It! page:



Plan-It!

Click the button to generate a random 3-meal menu plan!

Initial View



Plan-It!

Click the button to generate a random 3-meal menu plan!

Original Cheeries with Plain Soymilk (Breakfast)

1 cup of Original Cheerios with 1/2 cup of soymilk - a good source of fiber!

Food Groups:

- Dairy
- Grains

Froot Loops Cereal (Lunch)

A baggy of rainbow Froot Loops - a classic college student lunch.

Food Groups:

- Grains

Pinkie Pies (Dinner)

Two pies made with love and filled full of flavors. Blueberry, strawberry, you name it, you have it!

Food Groups:

- Grains
- Fruit
- Other

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() {

})();
```

5. *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
}

?>
```

6. PHP and SQL Connection with PDO

Recall the `games` table related to the one from lecture and section (but with fewer columns). Some example rows are below:

id	name	platform	release_year	genre	publisher
1	Pokemon Red/Blue	GB	1996	Role-Playing	Nintendo
2	Spyro Reignited Trilogy	PS4	2018	Platform	Activision
3	Universal Paperclips	PC	2017	World Domination	Frank Lantz
…	…	…	…	…	…

Using this table, you will write queries and then use the appropriate PDO functions in PHP.

Part A:
i. Write a SQL query that selects the **id**, **name**, and **platform** of all games where the **platform** starts with "PS" (eg. games with the platform "PS", "PS2", "PS4", "PSP", etc. should all be selected).

Expected results (1000 rows total):

id	name	platform
2	Spyro Reignited Trilogy	PS4
5	Crash Bandicoot	PS
…	…	…

Write your SQL Query here:

For the next two parts, you may assume **id** and **release_year** are INTEGERS, and the rest of the columns are VARCHAR types. You may also assume that the current size of the table is 3000, and the **id** column uses AUTO_INCREMENT.

ii. Write a SQL statement that inserts a new row into the table so that the row would like the following:

id	name	platform	release_year	genre	publisher
3001	Baba is You	PC	2018	Puzzle	Hempuli Oy

Write your SQL Statement here:

iii. Write a SQL statement that updates the row from part ii to have a **release_year** of 2019 instead of 2018.
Write your SQL Statement here:

Part B:
In this part, you will complete the below PHP function `delete_games_with_genre` which should remove all rows from the games table having the given genre.

i. Circle the *most* appropriate option to define the $sql variable in `delete_games_with_genre`:

a. `"DELETE FROM games WHERE genre = {$genre};"`

b. `"DELETE * FROM games WHERE genre = {$genre};"`

c. `"DELETE * FROM games WHERE genre = :genre;"`

d. `"DELETE FROM games WHERE genre = :genre;"`

ii. Fill in the code under the TODO comment to execute the query from part i and define a variable `$count` to be used in the `output_success` function. Use PDO's prepare and execute functions with the `$genre` parameter to prevent SQL injection and do not use `query` or `exec`. You may also find `rowCount()` useful (remember that this returns the number of rows affected in an executed PDO statement).

```
function delete_games_with_genre($genre) {
  // you may assume that get_PDO returns the appropriate PDO object
  $db = get_PDO();
  try {
      $sql = // assume answer from part i is defined here
       // TODO: Use $db with the $sql string to securely execute the query,
       // and define $count so that the success message below includes the
       // number of rows removed from the table.










      // You may assume that output_success correctly sets any needed headers
      // and outputs the message passed to it.
      output_success("Successfully deleted {$count} games from the table!");
  }
  catch (PDOException $ex) {
      // You may assume that the below function sets the correct
      // headers, outputs the message, and stops the script.
      handle_db_error("Can not connect to the database. Please try again later.");
  }
}
```

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?