CSE 154: Web Programming — Spring 2018

Final Exam Solutions

Name: _____

UWNet ID: _____@uw.edu

TA (or section): _____

Rules:

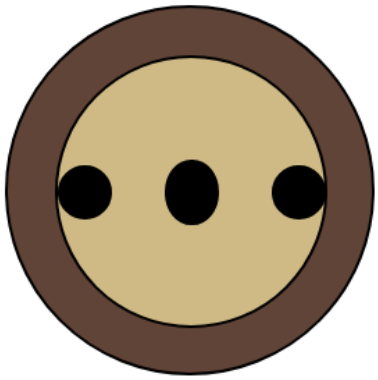
- You have 110 minutes to complete this exam.
- Do not open this booklet until time has begun. If we see you opening this booklet before time is started we will deduct from your grade.
- You will receive a deduction if you keep working after the instructor calls for papers.
- This test is an open note test.
- 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.
- The functions you may assume are defined are $ (for `document.getElementById`), qs (for `document.querySelector`), qsa (for `document.querySelectorAll`), and `checkStatus`.
- You must show your Student ID to a TA or instructor while sitting in your assigned seat in order to have received this exam.
- Once you enter the room, you must turn all parts of the exam, excluding the cheat sheet, with your name on it and will not be permitted to leave without doing so.
- Only solutions written on these pages will be graded for this final. You may choose to write on the question book portion of the exam, but none of that work will be graded.
- There is an extra piece of paper stapled to the back of this exam. If you continue problem, please make sure to note which problem you are continuing back there.
- It is your responsibility to ensure your exam is stapled back together and/or your name is on every page if you choose to pull the staple out of this booklet.

Question	Score	Possible
1. CSS		12
2. Short Answer		10
3. Regex		8
4. SQL		10
5. JavaScript		20
6. JavaScript/Ajax		20
7. PHP Web Service		20
8. Extra Credit		
Total		100

1. Cute, Slow, and Sleepy (12 pts)

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).

Baby Sloth!



Provided HTML `body`:

```
<body>
  <section>
    <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>
  </section>
</body>
```

Appearance Details:

- The `section` 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 here:

Solution:

```
section {
  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;
  display: block;
  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;
}
```

2. Short Answer (10 pts)

a) JavaScript Events

(1pt) 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).

Solution:

In order: 1, 2, 4, 3

b) JSON Mystery

(2 pts) 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.

Solution:

```
["j", 0, 1]
undefined
2
1
```

c) PHP Database Connections

(1pt) 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).

Solution:

If the user tries to insert something that has a ' in the username, email or password, they could potentially do something (at best) to break the query or at worst to inject SQL and do damage to the database.
Better would be do do the following:

```
$str = "INSERT INTO users (username, date, email, password)
        VALUES (:username, NOW(), :email, :password);";
$stmt = $db->prepare($str);
$rows = $stmt->execute(['username'=>$username, 'email'=>$email,
'password'=>$password]);
```

d) Validation Methods

(1pt) What is one advantage of validating user input on the client (HTML5 or JS) vs. on the server (PHP)?

Solution:

Three possible answers

- It is faster to validate on the client than on the server
- It can lead to a better UI/UX experience
- It can assist (but not completely solve) the problem of malicious user input by preprocessing the input before sending it to the server.

(1pt) What is one advantage of validating user input on the server as opposed to on the client?

Solution:

Validating on the server is more secure than on the client.

e) Technology Trade-offs: `indexDB` vs. Dexie

(1pt) Briefly explain the relationship between `indexDB` and Dexie and why you might want to use one over the other.

Solution:

- Dexie is a wrapper around indexDB that makes it easier to program/use.
- indexDB is more general and broadly supported by more browsers than Dexie.

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

(1pt) From the client perspective, what is an advantage of working with a JSON response over one in plain text format?

Solution:

JSON has indices - easier to parse vs plain text.

g) Storage Technologies

(2pt) 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

Solution:

i) localStorage
ii) indexDB
iii) sessionStorage
iv) cookies
v) sessions

3. Regex (8 pts)

a) CSE [1, 5, 4]

Write a regular expression that matches a non-empty array of integers (including negatives) such that any two integers are separated by a single `,` (comma) character which is followed by any number of spaces.

Solution:

Example solution (test with http://rubular.com/r/bYGjqIzkUF)

```
/^\[-?\d(,\s*-?\d)+\]$/
```

b) `img` tags

Write a regular expression that matches an HTML `img` tag with a `src` attribute ending in `.gif`, `.jpg`, or `.png` extension and having a non-empty `alt` attribute value preceding the `src` attribute (separated by a single space). Your expression should consider only double quotes (") for the attribute values and match whether or not there is a space after the `alt` value or a `/` before the ending `>`. There may only be numbers, letters, `.` or `/` in the `src` attribute value (the `alt` text may include character type). Ignore any letter-casing.

Solution:

Example solution (test with http://rubular.com/r/Tjgqvc51py)

```
/^<img src=".+.(gif|jpg|png)" alt=".+"\s?\/?>$/i
```

4. 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 **language**s 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    |
+-------------+
| Afrikaans   |
| Aimará      |
| Arabic      |
| Belorussian |
| ...         |
+-------------+
52 total rows returned. (2 ms)
```

Write your SQL query here:

Solution:

Test here: https://webster.cs.washington.edu/cse154/query/

```
SELECT DISTINCT l.language
FROM languages l, languages l2
WHERE l.language = l2.language
AND l.official = "T" AND l2.official = "T"
AND l.country_code <> l2.country_code
ORDER BY l.language;
```

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:

Solution:

Test here: https://webster.cs.washington.edu/cse154/query/

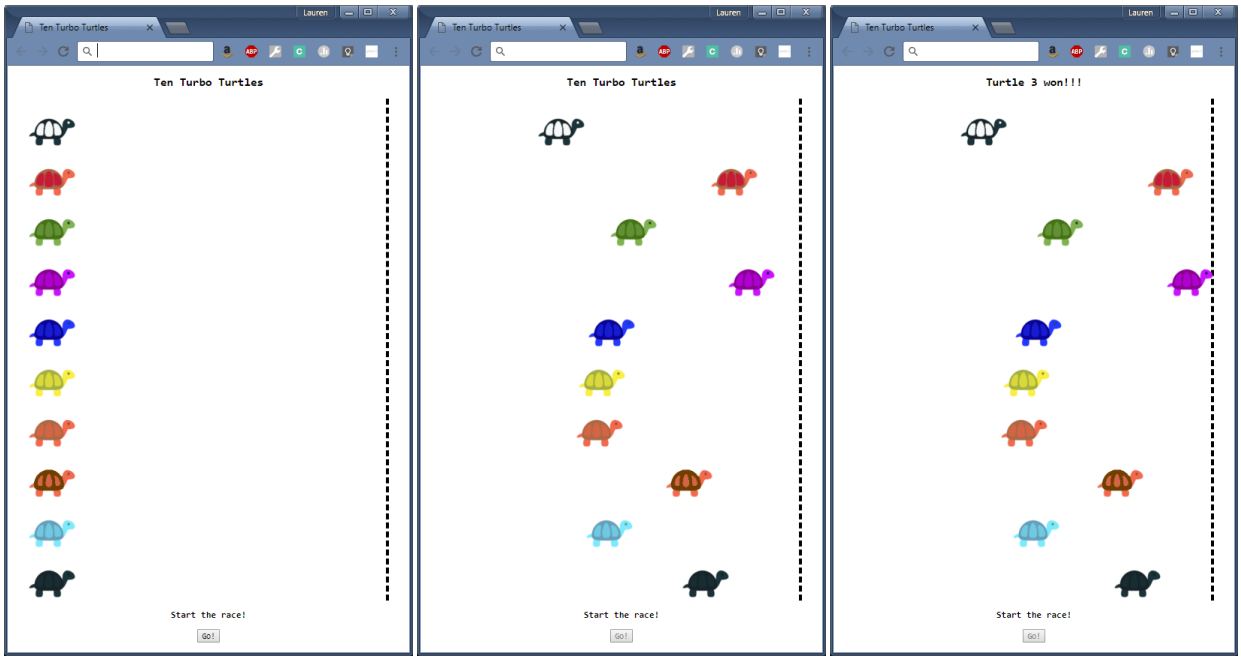
```
SELECT ci.name, continent, l.language
FROM cities ci, countries c, languages l
WHERE ci.country_code = c.code
AND c.code=l.country_code
AND l.official = 'T' AND c.name LIKE '%island%'
ORDER BY ci.name;
```

5. Turbo Turtles: The Championship Finale (20 pts)

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.onload = initialize;

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

```
/**
 * 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 */
}
```

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:

Solution:

```
let timers = [];
```

b)

Write the code to create a 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) {
```

Solution:

```
let newDiv = document.createElement("div");
newDiv.classList.add("racer");
newDiv.id = id;
let img = document.createElement("img");
img.src = image;
img.alt = alt;
newDiv.appendChild(img);
return newDiv;

}
```

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() {
```

Solution:

```
  // first add all of the turtles and the hare to the racetrack
  let racetrace = $("racetrack");
  for (let i = 0; i < timers.length; i++) {
    // create a turtle
    let name = "turtle" + i;
    let newRacer = createNewRacer(name,
                                  name + ".png",
                                  name + " image");
    // add it to the racetrack
    racetrack.appendChild(newRacer);
  }
  $("go").onclick = startRace;

}
```

d)

Write your code for the `startRace` function here

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

Solution:

```
  // disable the button!
  $("go").disabled = true;

  // set all the timers!
  let items = document.querySelectorAll(".racer");
  for (let i = 0; i < items.length; i++) {
    let speed = getRandomValue(1, 250);

    timers[i] = setInterval(function() {
      moveRacer(items[i]);
    }, speed);

    // could also do this as
    // timers[i] = setInterval(moveRacer, speed, items[i])
  }

  // GO!

}
```

e)

Write your code for the `stopAll` function here

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

Solution:

```
    for (let i = 0; i < timers.length; i++) {
      clearInterval(timers[i]);
    }

}
```

6. 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).

```
(function() {
```

Solution:

```
  window.onload = function() {
    $("day-btn").onclick = fetchFullMenu;
  };

  function fetchFullMenu() {
    fetch("planit.php?mode=day")
      .then(checkStatus)
      .then(JSON.parse)
      .then(populateFullMenu)
      .catch(console.log);
  }

  function populateFullMenu(responseData) {
    $("day-results").classList.remove("hidden");
    for (let key in responseData) {
      let item = responseData[key];
      qs("#" + key + " .name").innerText = item.name;
      qs("#" + key + " .description").innerText = item.description;
      let ul = qs("#" + key + " .food-groups");
      ul.innerHTML = "";
      for (let i = 0; i < item["food-groups"].length; i++) {
        let li = document.createElement("li");
        li.innerText = item["food-groups"][i];
        ul.appendChild(li);
      }
    }
  }

})();
```

7. *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).

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.

```
function get_meal_data($meal) {
```

Solution:

```
  $choices = glob("{$meal}/*.txt");
  $r = $choices[array_rand($choices)];
  $data = file($r, FILE_IGNORE_NEW_LINES);
  return array("name" => $data[0],
               "description" => $data[1],
               "food-groups" => explode(" ", $data[2]));

}
```

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
```

Solution:

```
if (isset($_GET["mode"])) {
  $mode = strtolower($_GET["mode"]);

  if ($mode === "day") {
    $result = array();
    $result["breakfast"] = get_meal_data("breakfast");
    $result["lunch"] = get_meal_data("lunch");
    $result["dinner"] = get_meal_data("dinner");
    header("Content-type: application/json");
    print(json_encode($result));

  } else {
    header("Content-type: text/plain");
    if ($mode == "meal") {
      if (isset($_GET["type"])) {
        $type = strtolower($_GET["type"]);
        $choices = glob("{$type}/*.txt");
        for ($i = 0; $i < count($choices) - 1; $i++) {
          $lines = file($choices[$i], FILE_IGNORE_NEW_LINES);
          print ("{$lines[0]}: {$lines[1]}\n");
        }
        $lines = file($choices[count($choices) - 1], FILE_IGNORE_NEW_LINES);
```

```
        print ("{$lines[0]}: {$lines[1]}");
      } else { # missing type
        header("HTTP 1.1 400 Invalid Request");
        die("Missing type parameter for meal request");
      }
    } else { # incorrect mode
      header("HTTP 1.1 400 Invalid Request");
      die("Please pass a mode of meal or day");
    }
  }
} else { # missing mode
  header("Content-type: text/plain");
  header("HTTP 1.1 400 Invalid Request");
  die("Please pass a mode of meal or day");
}

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

?>
```

8. 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?