CSE 154: Web Programming — Autumn 2017

Final Exam

Name: _____

UWNet ID: _____@uw.edu

TA (or section): _____

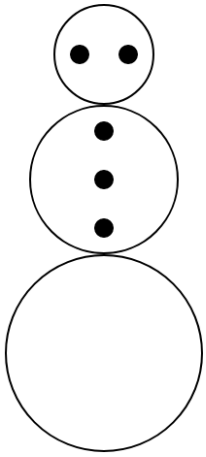
Rules:

- You have 110 minutes to complete this exam.
- You will receive a deduction if you open the test before the start or keep working after the instructor calls for papers.
- This test is open-book and open note.
- 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.
- You may use JavaScript function aliases like $ or qs **only if** you define them in your code.
- You may use the fetch syntax used in class in any JavaScript programs you write (you may assume checkStatus is included).
- 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
0. HTML/CSS		10
1. JavaScript		20
2. JavaScript/Ajax		20
3. PHP Web Service		20
4. Regex		9
5. SQL		12
6. Short Answer		9
7. Extra Credit		
Total		100

0. CSS is to HTML as Swag is to Snowman

Write the full HTML and CSS files to reproduce the following output:



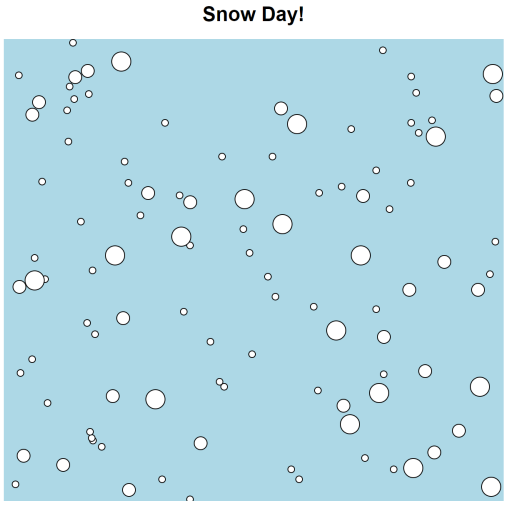
Each snowball has a 2px solid black border, a border-radius of 50%, and is centered on the page. The bottom snowball has a height and width of 200px, the middle has a height and width of 150px, and the top has a height and width of 100px. Each "spot" (eyes and buttons) has a height and width of 20px, a border-radius of 50%, and a black background. The top and middle snowballs have buttons that are equally-spaced apart (horizontally for the top and vertically for the middle snowball). You do not have to match our image exactly for the spacing between the spots, as long as they are spaced evenly and centered.

Hint: There are several ways to solve this problem using the different layout techniques we've learned in this class. Some will be easier to use than others.

Write your HTML and CSS solutions below.

1. SNOW Day in Seattle!!!

Write a JavaScript program `snow.js` to add to the existing HTML and CSS (next page). This program animates a snow scene with falling snow.



Screenshot During Animation

When the page is loaded, your JavaScript should start animations for continuously creating new snowballs and updating their position such that they give a vertical "falling" effect.

A new snowball (a div with classname `snowball`) should appear on the top of the `#snow-scene` div every 100ms, with a random x-coordinate position inside of this container. The snowballs should be randomly assigned a diameter of 10, 20, or 30 px;

Snowball positions are updated on the page every 50ms, moving down vertically. Snowballs that have a 30px diameter should move 10px down each update, while 20px snowballs should move 5px down, and 10 px snowballs should move 2px down. Whenever the top border of a snowball is no longer within the bounds of the `#snow-scene` container, the snowball should be removed from the DOM.

You may not use `setTimout` and may only create up to two intervals using `setInterval`.

Hint: You can use `domElement.style.cssAttribute` to set and retrieve css attributes.

```
<!-- HTML for Problem 1 -->
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Snow!!!</title>
    <link href="snow.css" rel="stylesheet" type="text/css" />
    <script src="snow.js" type="text/javascript"></script>
  </head>
  <body>
    <h1>Snow Day!</h1>
    <div id="snow-scene"></div>
  </body>
</html>
```

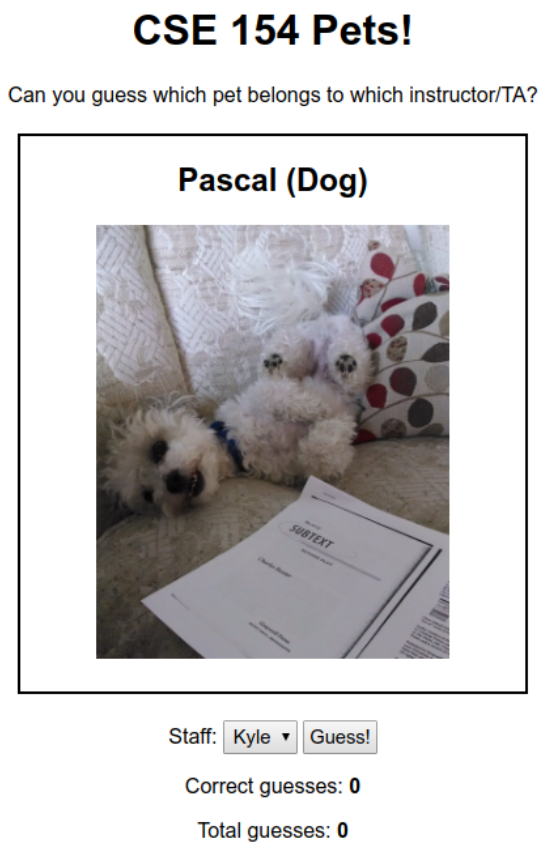
```
/* CSS for Problem 1 */
html, body { height: 100%; }

h1 {
  font-family: Helvetica, Arial, sans-serif;
  text-align: center;
}

.snowball {
  background-color: white;
  border: 1px solid black;
  border-radius: 50%;
  position: absolute;
}

#snow-scene {
  background-color: lightblue;
  height: 100%;
  margin: auto auto;
  position: relative;
  width: 800px;
}
```

Write your solution here.

Continue your solution here if needed.

2. Fetching Pets Who Fetch with Fetch

Write a JavaScript file `pets.js` that plays a guessing game with the client, displaying the name and image of a random pet owned by a CSE 154 staff member, and which keeps track of how many times the client correctly guesses the TA/instructor who owns the pet. Below is a screenshot of the page before a user has made any guesses:



Initial Behavior

When the page loads, you should make an AJAX request to `pets.php` with a query parameter of `mode=tas`. Use the returned (plain text) response to populate `#ta-list` with a new option tag for each TA/instructor name in the result. (Hint: remember that each option tag in a `<select>` dropdown should have a unique value attribute to determine what the current selection is).

Example response from `pets.php?mode=tas`:

```
Conner
Jeremy
Kyle
Lauren
Melissa
Sam
```

In addition to populating the dropdown with staff names, you should initialize the guessing game with the first random pet. You should get this data with a call to `pets.php`, passing the query parameter `mode=random`. Using the JSON response returned, you should update `#pet-name` and `#pet-type` to be the name and type of the random pet and update the source of `#pet-img` to be a randomly-selected image source path contained in the JSON's `images` array. If the array contains only one image, you should use this image as the source. When the pet image is first changed, remove the initial "hidden" class from `#pet-img` (you should not hide it again).

Example response from `pets.php?mode=random`:

```
{
  name : "Melissa",
  petname : "Mowgli",
  type : "Dog",
  age : "10 months",
  images: [ "Melissa/Mowgli/mowgli_at_school.jpg",
            "Melissa/Mowgli/mowgli_in_moose_sweater.jpg",
            "Melissa/Mowgli/mowglis_first_steps.jpg",
            "Melissa/Mowgli/sleep_puppy.jpg" ]
}
```

Once the initial pet information is populated on the page, the `#guess-btn` should be enabled (it is initially disabled with the class "disabled" - removing this class will enable the button).

Processing Guesses

When a user clicks the guess button, the button should be disabled again and the currently-selected TA/instructor name in `#ta-list` should be used to make a guess of the pet's owner. If the guess is correct for the current pet, increment the value for `#count` by one. Whenever a guess is made, `#total` should be incremented by 1.

Below is a screenshot after 6 guesses with 5 correct guesses (with a new random pet displayed):

After a guess has been made, a new random pet should be retrieved from `pets.php` to populate the `#pet-info` similar to case for the first random pet. The current selection in the drop-down should remain unchanged. Once data for a new random is retrieved successfully, re-enable the `#guess-btn`.

Provided HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <script type="text/javascript" src="pets.js"></script>
  <link href="pets.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <h1>CSE 154 Pets!</h1>
  <p>Can you guess which pet belongs to which instructor/TA?</p>

  <div id="pet-info">
    <h2><span id="pet-name"></span> (<span id="pet-type"></span>)</h2>
    <img class="hidden" id="pet-img" src="" alt="super cute pet" />
  </div>

  <div id="guess-ui">
    Staff:
    <select id="ta-list"></select>
    <button class="disabled" id="guess-btn">Guess!</button>
  </div>

  <div>
    <p>Correct guesses: <span id="correct">0</span></p>
    <p>Total guesses: <span id="total">0</span></p>
  </div>
</body>
</html>
```

You can start your JavaScript code here or on the next page.

Start or continue your JavaScript code here.

3. The Thing That We Fetch From For Fetching Pets Who Fetch

Write a PHP web service called `pets.php` which provides data about CSE 154 staff and their pets. For this problem, assume your PHP file is in the same directory as `pets.txt` and a collection of folders for each TA/instructor. On the rest of this page, we will provide a short overview of the files/directories you will work with. The implementation requirements for the web service will be given on the following page.

pets.txt

This file contains information about each staff member's pet on its own line, in the following format:

```
name petname pettype petage
```

where `name` is the staff member's first name, `petname` is the name of the pet owned by `name`, `pettype` is the type of pet, and `petage` is the age of the pet. You may assume `name`, `petname`, and `pettype` contain no spaces and have only English alphabet letters, but `petage` may have spaces and numbers (e.g. "1 month" or "8 years"). Note that some staff members have more than one pet, but each pet is on its own line.

An example of `pets.txt` is given below:

```
Melissa Mowgli Dog 10 months
Kyle Pascal Dog 9 years
Sam Cachaca Dog 3 years
Lauren Spot Cat 16 years
Lauren Jack Cat 16 years
Lauren Whitney Cat 16 years
Jeremy Coloratura RainbowPony 154 days
Conner Bailey Dog 6 months
```

Staff and pet directories

Any staff member who has a pet has a directory named for each of their pets. Each pet directory contains at least one `.jpg` photo (it may also include non-jpg file types). For example, Melissa's dog Mowgli is her only pet. So in the folder `Melissa/`, there is a folder called `Mowgli/` which contains the following files:

```
mowgli_at_school.jpg
mowgli_in_moose_sweater.jpg
mowglis_first_steps.jpg
mowglis_growth_chart.csv
sleepy_puppy.jpg
```

Manesh does not have a pet, so in the directory `Manesh/`, there are no directories for pets (although there may be other folders/files in his directory). When implementing your PHP, you may assume that the format of each staff member's name and their pet's name is exactly the format of the corresponding folder names.

Web Service Implementation

Your web service should accept two possible modes (case-insensitive):

mode=tas

If a mode of `tas` is passed as a GET parameter, your web service should output as plain text a list of all staff members who own pets, with each TA/instructor's name on its own line. Below is example output:

```
Conner
Jeremy
Kyle
Lauren
Melissa
Sam
```

mode=random

If a mode of `random` is passed as a GET parameter, your web service should output a JSON response with information about a random pet listed in `pets.txt` as is shown in Problem 2. The general format of the expected JSON response is provided below:

```
{
  name: "staffname",
  petname: "petname"
  type: "pettype",
  age: "petage",
  images: ["image01.jpg", "image01.jpg", ...]
}
```

Anything above in quotes should be replaced with the corresponding information unique to the randomly-chosen pet.

The array of images for the pet (the value for the JSON's `images` key) should contain all `.jpg` images found in the directory **`name/petname/`** given as full paths relative to the location of `pets.php`. For example, if Mowgli was the random pet result for a call to `pets.php?mode=random`, the result JSON would be identical to that from Problem 2. Each pet in `pets.txt` should have an equally-likely chance of being returned from this mode.

If either mode is not passed, your program should output an error with a status of "`HTTP/1.1 400 Invalid Request`" and a plain text error message, "Error: Please pass in a mode parameter of tas or random."

Start your PHP code here or on the next page

Start of continue your PHP code here

4. RegEx

a. Caaafffeeeinee!!!!!

It's finals week, and as true Seattlites, we need caffeine to survive it. Write a regular expression to match all Strings that contain the words "coffee", "mocha", or "green tea" containing no digits or special characters (space characters should be accepted). Both lower-case and upper-case letters are allowed for any letter in the String.

match:
Venti black coffee with five espresso shots
grande white chocolate MOCHA
Green tea rice
COFFEE BEANS
Coffeescript

don't match:
bubbletea
koFFing bean

b. Deoxy-what?

Write a regular expression to match DNA sequences that consist of codons (sequences of three nucleotides [A, C, T, or G]). The expression should only match sequences that start with the start codon 'ATG' and end with one of the following three stop codons: 'TAA,' 'TAG,' or 'TGA.' Any number of codons with any sequence of nucleotides may be between a start codon and a stop codon. Sequences that do not have complete codons (i.e., those which do not have nucleotide counts that are multiples of three) should not be accepted.

match:
ATGTAA
ATGTAATAATAG
ATGATCAAAAACTAG

don't match:
TAGTAG
TAGATG
ATGGTAAT

c. SELECT-em ALL!

Write a regular expression that accepts simple SQL select statements. That is, any SQL statement that:

- starts with "SELECT" followed by one or more comma-separated column names followed by
- by a single "FROM" statement followed by
- one or more comma-separated table names and
- is finished optionally with an semi-colon (;)

A valid column or table name contains only letter characters (no spaces, numerical, or special characters), except that a column name may optionally include a single '.' within the name string (not at the start or end). SELECT and FROM must be **both** all-lowercase or **both** all-uppercase.

match:
SELECT puppy FROM puppies;
select pokemon.name, pokemon.nickname, type FROM pokemon;
SELECT FrOm FROM SeLeCT
SELECT quilts, quarts, quinoa FROM Qstore;

don't match:
FROM puppies SELECT puppy;
SELECT puppy FROM p.uppies;
select .name, FROM pokemon;
SELECT puppy from puppies
Select foo fRom bar;
SELECT puppy FROM puppies WHERE puppy='mowgli';

5. Would You Like a Table With That Menu?

Consider the following tables in a database called "Cafe":

Menu					
mid	**item**	**price**	**cost**	**category**	**subcategory**
1	Brewed Coffee (Black)	1.25	0.65	Drink	Coffee
2	Espresso	3.75	1.15	Drink	Coffee
3	Jeremy's Unicorn Frappe	8.00	2.75	Drink	Coffee
4	Bubble Tea	3.00	1.75	Drink	Tea
5	Blueberry Muffin	3.00	1.20	Bakery	Muffin
6	Honey Toast	4.00	1.85	Bakery	Bread
7	Chocolate Chip Cookie	1.25	0.50	Bakery	Cookie
8	Chocolate Chip Cookie (Gluten-Free)	1.25	0.60	Bakery	Cookie
...	...	...	...	...	...

Customers		
cid	**firstname**	**lastname**
153	Sam	Kaufman
154	Kyle	Thayer
155	Jeremy	Zhang
156	Kelley	Chen
157	Melissa	Medsker
...	...	...

Orders					
oid	**mid**	**cid**	**date**	**time**	**qty**
1000	2	157	2017-11-29	03:00	154
1001	3	155	2017-12-02	13:00	1
1002	3	155	2017-12-08	13:30	50
1003	4	156	2017-12-09	09:30	4
1004	1	153	2017-12-10	01:00	12
...	...	...	...	...	...

The Menu table has a primary key of **mid**, the Customers table has a primary key of **cid**, and the Orders table has a primary key of **oid**. The **mid** and **cid** attributes in Orders correspond to the primary keys in Menu and Customers, respectively.

The questions are on the next page.

(a) Write a SQL query which returns the item name, date, time, and the last name of the customer for all Coffee Drink orders (Drink orders with subcategory 'Coffee') which were made in the year 2017. Results should be ordered by the most recent order first.

Hint: You can treat the `date` attribute as a normal SQL string to determine whether it is a date in 2017. `date` and `time` are comparable as expected. That is, the date `2017-12-08` (December 8, 2017) is considered 'greater than', or 'more recent' than `2017-05-02` (May 2nd, 2017) and the time `13:00` (13:00 or 1:00 PM) is considered 'more recent' in a given day than `01:00` (01:00 or 1:00 AM).

(b) Write a SQL query which returns the first and last name of the customer, time of order, menu item, and category for all customers who have both an order for a Drink and an order for a Bakery item in the database (they may have more than one order for both categories, but you should include all of the orders for customers who meet this requirement). There should be no duplicate rows with the same customer, order time, menu item, and category.

6. Short Answers

1. What is the difference between a relative link and an absolute link?

   _____

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

   _____

3. Provide one real-world example where a cookie would be more appropriate to use than a session

   _____

4. Provide one real-world example where a session would be more appropriate to use than a cookie

   _____

5. What is one advantage of using a SQL database over text files to store data?

   _____

6. What are two ways of making a website more accessible for users with different abilities?

   _____

7. What is the difference between a GET and POST request in PHP?

   _____

8. Why is it important to specify the content type header in PHP when outputting something?

   _____

9. Explain one way someone could do something bad by exploiting a vulnerability in your site if it wasn't written securely.

   _____

7. Extra Credit (1 extra credit point)

If your TA were an HTML tag, which one would they be, and why? You may also give us an artistic rendering of your TA as an HTML tag, if you prefer. (Drawing, poetry, etc). Any work that appears to have taken more than 1 minute of effort and is not offensive/inappropriate will receive the extra credit point.