CSE 154 Practice Exam from 16au

Name: _____ Quiz Section: _____
TA: _____ Student ID #: _____

Rules: You have 110 minutes to complete this exam. You will receive a deduction if you keep working after the instructor calls for papers. This test is open-book, but closed notes. You may not use printed/written notes or practice exams. You may not use any computing devices, including calculators, cell phones, or 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.

Good luck!

Problem:	Possible:	Score:
HTML	10	
CSS	15	
Javascript	15	
Javascript and AJAX	15	
PHP Webservice	15	
SQL	15	
Embedded PHP	10	
Regular Expressions	5	
	100	

HTML (10 points)

```
<html lang="en">
<head>
  <title> CSE 154 Hello, World! </title>
</head>
<body>

<div>
    <span>Hello <div>World!</div></span>
    <src="hello.jpg" alt="image of hello" img>
</div>

<p
    CSE 154 is the best class ever! #propagandamachine
</p>

<blockquote>
     The internet is a series of tubes.

</body>
</html>
```

This html document won't validate, and would generate many errors and warnings. However, it is possible to make it validate by changes to 5 lines in the html. Indicate what changes you could make to the html document 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 to match the explanations. No need for an essay -- 20 words or less should be plenty.

1. _____

2. _____

3. _____

4. _____

5. _____

CSS (15 points)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <link rel="stylesheet" type="text/css" href="index.css" />
</head>
<body>

<div id="1" class="menu">
  <a id="2" class="home-link" href="home.html">Home</a>
</div>

<div id="3" class="banner-advertisement">
  <p id="4">
    <a id="5" href="https://some.site.com/advertisement">Support our
sponsor!</a>
  </p>
</div>

<div id="6" class="content">
  <p id="7">
    <span id="8">Welcome to my home page. I mostly made it for the ad
revenue... so...</span>
  </p>
  <span id="9">
    Check out mah <a id="10" href="blog.html">blog</a>!
  </span>
</div>

<div id="11" class="menu">
  <a id="12" class="home-link" href="home.html">Home</a>
</div>

</body>
</html>
```

Query selectors:

For each of the following query selectors, indicate the `id`s of all of the html elements that would be found by the selector.

`span` _____

`.banner-advertisement` _____

`.home-link` _____

`div a` _____

`div p, div > span` _____

Implement index.css:

1. make the `.content` `<div>` float to the right, and consume 75% of the width of the page.
2. make the text size of the advertisement link 32pt. (Gotta make that money.)
3. make all `<p>` elements have no margin, border, or padding
4. make the background color of all menus black, and have a 3px border that is solid and colored rebeccapurple.
5. make the color of all links inside menus white

Javascript, AJAX,JSON, PHP, Webservices:

Use this html document for reference for the following 3 problems.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <script src="sesame-street.js" type="text/javascript"></script>
    <script src="fetch-pie.js" type="text/javascript"></script>
</head>
<body>

<h1>Whitaker's Desserts</h1>

<h2 id="cookie-header">Whitaker's Cookie Jar:</h2>
<ul id="cookie-jar">
    <li class="cookie">Chocolate Chip</li>
    <li class="cookie">Oatmeal raisin</li>
    <li class="cookie">Macaroon</li>
</ul>
<p id="cookie-count"></p>

<h2>Whitaker's Pie Cupboard:</h2>
<ul id="pie-cupboard">
</ul>
<button id="moar-pie">Moar Pie!</button>

</body>
</html>
```

Javascript (15 points)

Implement sesame-street.js
a. (Big Bird Yellow): When the page loads, apply a style to the #cookie-header <h2> that sets the text color to hex value #f7f16d

b. (Count Chocula): Implement javascript in sesame-street.js such that when the page loads, you find all of the cookie list items in the cookie-jar, count them, and set the text of the `#cookie-count <p>` to be.
`<# of cookies>! There are <# of cookies> cookie(s) in the cookie jar!`

(Replace '<# of cookies>' with the correct number).

c. (Cookie Monster hungry): Implement javascript logic to remove the last cookie in Whitaker's cookie jar every 30 seconds. Make sure to update the #cookie-count when you remove a cookie: to accomplish this, you may assume that your solution from part(b) is correct and functional.

Javascript, AJAX: (15 points)

Implement fetch-pie.js. fetch-pie.js should allow the user to click the 'Moar Pie!' button to fetch Whitaker's pies from a web service.

Make an AJAX get request to a webservice that lives at:
`https://whitakers.pi.es/getPies.php`

It returns a JSON document with this structure:

```
{
    'pies' => [
        { 'type' => 'pumpkin' },
        { 'type' => 'banana-cream' },
        ... more pie ...
    ]
}
```

Upon receiving the JSON, use javascript to insert one list item per pie into the `#pie-cupboard <ul>`. Each pie list item should have class 'pie'. The list item's text should contain only the type of pie (exactly as it appears in the 'type' attribute). Replace or remove any pies that were present in `#pie-cupboard`. For example, with the above JSON, you should insert the following list items into the `#pie-cupboard <ul>`:

```
<li class="pie">pumpkin</li>
<li class="pie">banana-cream</li>
```

If there is an error retrieving the pies, insert one list item into the `#pie-cupboard <ul>` with class 'error', and text that you choose that describes that an error occurred while fetching the pies. Replace or remove any pies that were present in `#pie-cupboard`.

PHP Webservice (15 points)

Implement getPies.php:
Write a php webservice that reads Whitaker's pies out of a file called 'pies.txt', and prints out JSON content representing the pies in the following form:

```
{
    'pies' => [
        { 'type' => 'pumpkin' },
        { 'type' => 'banana-cream' },
        ... more pie ...
    ]
}
```

pies.txt contains one line per type of pie, and the reason that Whitaker likes it. The type of each pie is guaranteed to not have a ':' character.

```
pumpkin: all of thanksgiving nostalgia
banana-cream: excellent for throwing at people
```

Your php program should also accept a query parameter 'limit' that caps the number of pies returned. You may assume that user passes in a valid value for limit, that is, a non-negative integer. However, if the user does not pass a limit query parameter, you should assume that they want a limit of 10 pies.

For example the, a call to:

```
https://my.host.com/getPies.php?limit=4
```

Would return the first 4 pies in pies.txt (or fewer if pies.txt has fewer than 4 pies)

Note: this is the corresponding webservice for the previous problem.

SQL (15 points)

All from the Simpsons. You can view the schema here:
https://webster.cs.washington.edu/cse154/query/

1. Select all the rows and columns from the teacher table.
2. Select only the grade column from the grades table where the course id is greater than 10006.
3. Select the course ids of all the courses that students named Bart took.

Embedded PHP: (10 points)

Write a pair of pages that work together to allow the user to upload an image, img_upload.html and img_submit.php.

img_upload.html should be a full and well-formed html page containing a form with a file upload input and a submit button. When the user submits the form, it should post the form to img_submit.php, located in the same directory.

When the user posts a file to img_submit.php, your program should check to see whether the filename of the file that was given ends in .jpg or .gif. If it doesn't, the page should respond with HTTP/1.1 status code 400, and then die() with an error message describing the problem.

You should save the file by moving it into the same directory as your program, saving the image as the original filename that was given by the POST request. Then, it should render a complete HTML document with the image in the document as an img element. The alt text of the img should be "user uploaded image". The title of the html page that is rendered should be:
"CSE 154: Viewing file <<filename>>"

Error handling: if the file fails the is_uploaded_file check, or if the filename of the file that the user uploaded doesn't end with .jpg or .gif, you should respond with a header that indicates a status code of 400, and die() with an appropriate error message.

Regular Expressions: (5 points)

Write regular expressions that match the descriptions.

1. Bob Loblaw: Write a regular expression that matches the name ‘Bob Loblaw’ anywhere in a string. Also, there can any number of ‘b’s in the middle of ‘Loblaw’:

_____

Matches:
Bob Loblaw
Bob Lobbblaw
Bob Lobbbbbbbbbblaw
We aren’t here to talk nonsense to **Bob Lobblaw**.

Does NOT match:
Bbob Loblaw
Bob Loboblaw

2. Write a regular expression that matches any dollar amount, anywhere in a string:

_____

Matches:
$100
$200.00
Hello**$2**
$99999.999999

Does NOT match:
$ABC
1234
$ab1234

3. Mr. or Mrs. F: Write a regular expression that matches an entire string representing someone with the last name ‘F’, and any title: Mrs., Ms., Miss, or Mr.

_____

Matches:
Mrs. F
Mr. F
Miss F
Ms. F

Does NOT match:
Jr. F
Mr. Frank
Ms Frank
SMr. F