

CSC 337, Fall 2013

Assignment 6

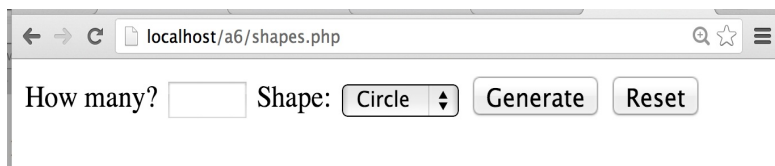
Due: Wednesday, November 6 at 22:00:00

In this assignment you'll be working with HTML forms and making some use of PHP arrays. I'll also ask you to take the first step on a final project.

Problem 1. (10 points) `shapes.php`

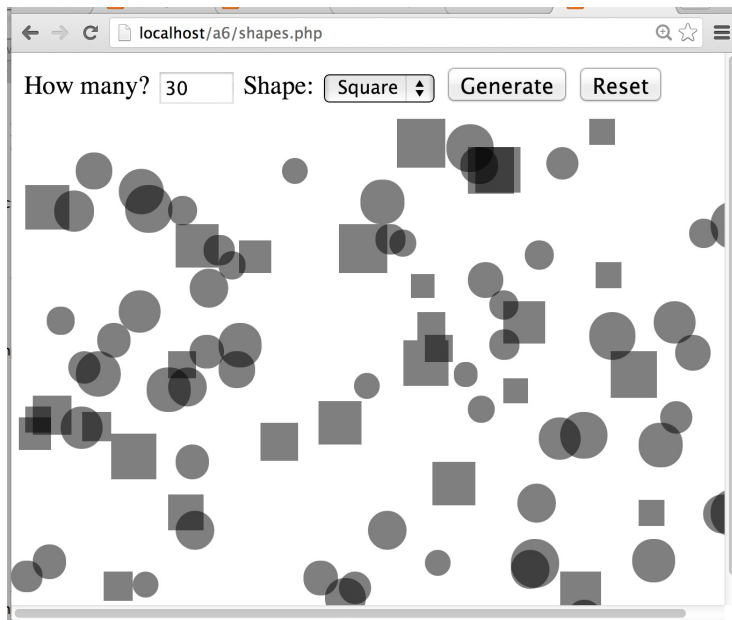
In this problem you are to create a PHP web application that generates a series of randomly sized and placed shapes using a form with "sticky" values and a hidden field. You'll probably need to use CSS for positioning the shapes but you are not required to style the form.

You'll have some flexibility on the design but here's how my solution looks when first hit. Only the top part of the browser window is shown; the rest is blank.



The user is expected to enter an integer number of shapes to generate. The user can choose to generate circles or squares. When the user clicks the Generate button the specified number of shapes are generated.

Here's what the user might see after entering 30 for "How many?", clicking Generate twice, changing to Square, and then clicking Generate again:



A total of 90 shapes are present—60 circles and 30 squares. Note that the user's last-entered count (30) and type (Square) are still present. Those are "sticky" values—whatever the user last entered/selected persists for the next run. An example of the technique used for this is shown in the slides.

IMPORTANT: The data for the shapes generated by previous clicks of Generate are collected in a hidden field and then used to reproduce the previously generated shapes along with newly generated

shapes.

Here's a sample of what that hidden field looks like in my version after two Generates:

```
<input type=hidden name=history
value='C.77.89.19;C.16.36.16;S.57.25.15;S.9.66.27;S.10.84.23;'>
```

The first Generate requested two circles. The second Generate requested three squares. The first circle is absolutely positioned at `top:77%` and `left:89%` and is 19 pixels in diameter. The first square is at `57%,25%` and is 15 pixels on each side.

The challenging parts of this problem are the sticky values and the accumulation of generated shapes. In my solution I've chosen to generate a variable number of squares or circles. You can choose to do the same but **you're free to vary other characteristics instead**. For example, instead of letting the user choose between circles and squares, you might let them pick between a small, medium or large variation in size, but only generate circles.

Here's what your solution must do:

1. Provide a form with two "sticky" values, one of which must be chosen with a `<select>` element. (My `<select>` chooses between squares and circles.)
2. Accumulate results across multiple clicks of a Generate button using an `<input>` element with `type=hidden`.
3. Have a Reset button that clears previous results.

The value in the hidden field can grow quite long. For that reason, use a POST, not GET.

Problem 2. (8 points) `load_entries.php`

The **third** problem on this assignment is to produce version 2 of our blog, which will have a form for adding entries and associating tags with them. **This problem** asks you to write a PHP function (not a whole program!) named `load_entries` that reads a file in the (new) format of `blog.txt` and returns an array of blog entries that are in turn arrays themselves.

Here's a sample input file: (`$a6/entries.txt`)

```
% cat entries1.txt
I've started a blog!
tags: x,y,z
2013-08-28
Line 1
Line 2
.end
Phone trouble...
2013-09-19
First (and last) line.
.end
```

Here is a test program for the `load_entries` function: (`$a6/test_load_entries.php`)

```
% cat test_load_entries.php
<?php
include("load_entries.php");

var_dump(load_entries("entries1.txt"));
```

Here is the output produced by `test_load_entries.php`, with `entries1.txt` shown beside it.

```
% php test_load_entries.php
array(2) {
  [0]=>
  array(4) {
    ["title"]=> string(20) "I've started a blog!"
    ["date"]=> string(10) "2013-08-28"
    ["text"]=> string(14) "Line 1
Line 2
"
    ["tags"]=>
    array(3) {
      [0]=> string(1) "x"
      [1]=> string(1) "y"
      [2]=> string(1) "z"
    }
  }
  [1]=>
  array(4) {
    ["title"]=> string(16) "Phone trouble..."
    ["date"]=> string(10) "2013-09-19"
    ["text"]=> string(23) "First (and last) line.
"
    ["tags"]=>
    array(0) {
    }
  }
}
```

```
% cat entries1.txt
I've started a blog!
tags: x,y,z
2013-08-28
Line 1
Line 2
.end
Phone trouble...
2013-09-19
First (and last) line.
.end
```

The `var_dump()` call in `test_load_entries.php` produces ALL the output above.

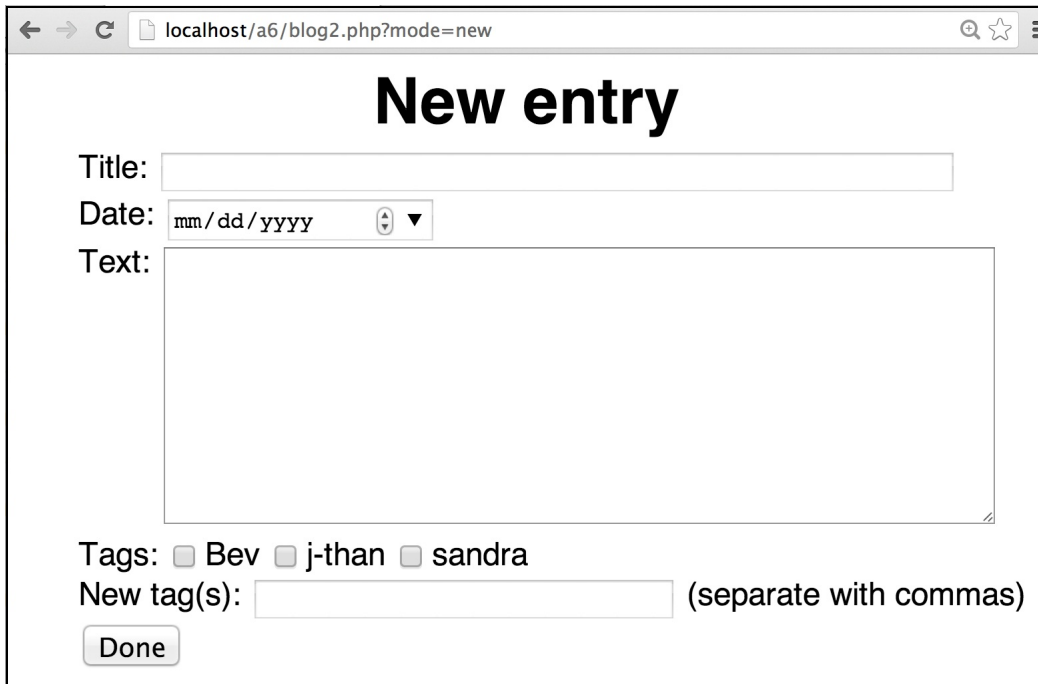
`load_entries()` produces no output; it returns an array of arrays. We can see that we have an outer array with two values, one for each of the two entries in `entries1.txt`. This outer array is much like a two-element list in Python or Java, indexed by integer keys 0 and 1. In turn, the first value in that two-element array, with key 0, is an array with the string-valued keys "title", "date", "text", and "tags". The values associated with the first three keys correspond to the text of the blog entry's title, date, and multi-line body, with embedded newlines. The value associated with the fourth key, "tags", is an array with three strings, representing tags x, y, and z. It was produced by using `explode(", ", ...)` and `trim(...)` on the text of the line "tags: x, y, z". If an entry has tags, they will appear as the second line of the entry. If that line doesn't start with "tags: ", then the entry has no tags, like the second entry in `entries1.txt`.

My intention and my advice is for you to first write `load_entries.php` and then use it in `blog2.php`.

Problem 3. (32 points) blog2.php

NOTE: This problem brings a lot of things together in one place. Simply understanding this problem and how the elements fit together may well amount to one-third to one-half of your total time on the problem. You'll probably need to read through this write-up several times.

For this problem you are to add some new features to assignment 5's `blog.php`. One of the new features is a form that lets the user enter data for a blog entry field-by-field rather than editing `blog.txt`. Here's the form, which has little styling:



The screenshot shows a web browser window with the address bar containing `localhost/a6/blog2.php?mode=new`. The page title is "New entry". The form consists of the following elements:

- Title:** A text input field.
- Date:** A date input field with a dropdown arrow and a calendar icon.
- Text:** A large text area for the entry content.
- Tags:** A row of three checkboxes labeled "Bev", "j-than", and "sandra".
- New tag(s):** A text input field followed by the text "(separate with commas)".
- Done:** A button at the bottom left of the form.

Note that the form is not very secure. As the browser's address field implies, hitting `/a6/blog2.php?mode=new` is all that's required to make a new entry! The **Title:** and **New tags(s):** fields are simply `<input type=text ...>` elements. The date is entered with an `<input type=date ...>`, which will bring up a calendar in Chrome. A `<textarea>` element is used for the body of the entry. (Look it up!)

Tags

Another feature you are to add is support for tags on entries. On the form above, **Tags:** is followed by a list of checkboxes, one for each known tag. The set of known tags is the union of all tags that appear on the `"tags: ..."` line of the entries. You might produce the set of known tags by doing

```
$entries = load_entries("blog.txt");
```

then looping through them with

```
foreach ($entries as $entry) { ...
```

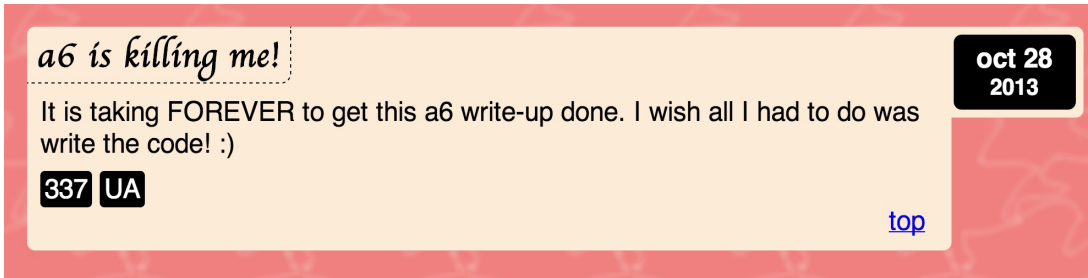
For each entry, get `$entry["tags"]` and build up an array perhaps named `$known_tags`.

Note that one way to produce a union of two arrays in PHP is to use `array_unique()` and `array_merge()`:

```
$c = array_unique(array_merge($a, $b));
```

Come and see us sooner rather than later if you have trouble understanding the computation of the set of known tags.

Tags on an entry are shown like this:



Filtering on a tag

Clicking a tag causes the blog entries to be filtered so that only entries having that tag are shown. Clicking on the UA tag above might produce this view of the blog:



Note the URL shown: `/a6/blog2.php?filter=UA`. That white-on-black UA tag is actually just a styled link:

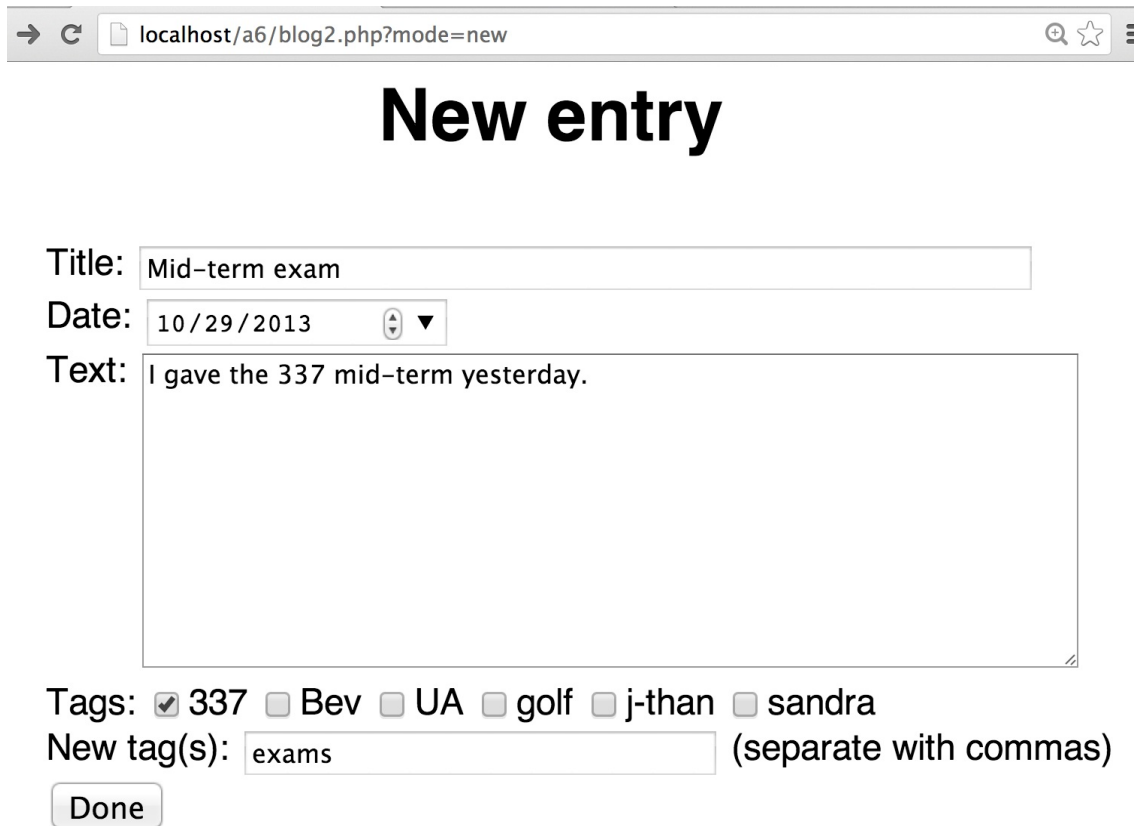
```
<a href="blog2.php?filter=UA">UA</a>
```

Clicking that link does an HTTP GET and in turn `blog2.php` is run with `$_GET["filter"]` equal to "UA". Again using `load_entries(...)` you might loop through the entries and include only those where `in_array("UA", $entry["tags"])` is true.

Clicking the white on black X following "Posts tagged UA" should hit `blog2.php`, causing the full set of entries to be displayed.

Back to the form

Let's revisit the data entry form. Here's a form that's been filled out:



→ ↻ localhost/a6/blog2.php?mode=new

New entry

Title:

Date:

Text:

Tags: 337 Bev UA golf j-than sandra

New tag(s): (separate with commas)

When the Done button is clicked, an HTTP GET is done. Here's the URL that's hit:

```
http://localhost/a6/blog2.php?mode=add&title=Mid-term+exam&date=2013-10-29&text=I+gave+the+337+mid-term+yesterday.&tags%5B%5D=337&newtags=exams
```

Here's what `var_dump($_GET)` shows:

```
array(6) {
  ["mode"]=> string(3) "add"
  ["title"]=> string(13) "Mid-term exam"
  ["date"]=> string(10) "2013-10-29"
  ["text"]=> string(34) "I gave the 337 mid-term yesterday."
  ["tags"]=>
  array(1) {
    [0]=>
    string(3) "337"
  }
  ["newtags"]=>
  string(5) "exams"
}
```

Note the relationship between (1) the data in the form (2) the data in the URL and (3) the data in the `$_GET` array.

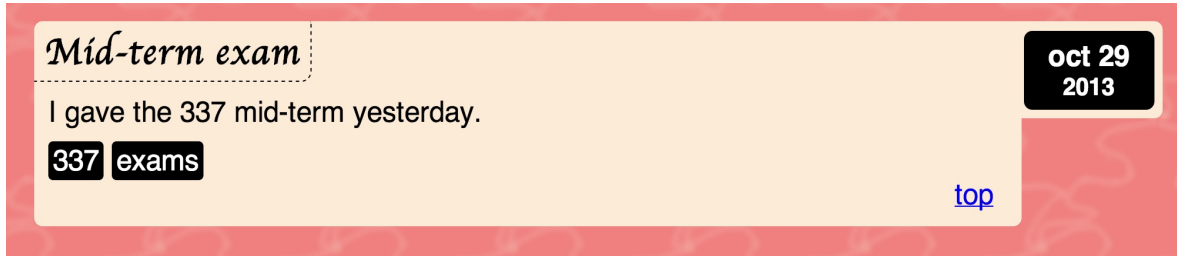
The presence of the URL parameter `mode=add`, supplied by this hidden field:

```
<input type="hidden" name="mode" value="add">
```

causes code to be executed that appends the following lines to `blog.txt`:

```
Mid-term exam
tags: 337,exams
2013-10-29
I gave the 337 mid-term yesterday.
.end
```

Along with the previously existing entries, we'll see this new one, too:



Note the end-to-end correspondence of values. For example, follow "Mid-term exam" from the form, to the GET, to `$_GET`, to the file, to final rendering.

Appending to a file

To append data to a file we can use `fopen(..., "a")` and `fputs(...)`, the counterpart of `fgets(...)`. Here's a program that appends one line to `test.txt`:

```
% cat append.php
<?php
$f = fopen("test.txt", "a");
fputs($f, "Appended!\n");
fclose($f);
```

Execution:

```
% date > test.txt
% php append.php
% cat test.txt
Tue Oct 29 00:31:45 MST 2013
Appended!
% php append.php
% cat test.txt
Tue Oct 29 00:31:45 MST 2013
Appended!
Appended!
```

Instead of appending to `test.txt`, you'll be appending to `blog.txt`!

Bookmarking a GET for testing

Here's that' GET from above:

<http://localhost/a6/blog2.php?mode=add&title=Mid-term+exam&date=2013-10-29&text=I+gave+the+337+mid-term+yesterday.&tags%5B%5D=337&newtags=exams>

If I bookmark that GET and hit that bookmark, a duplicate of that entry will be made. Things like that make for headaches with production systems but it also makes for easy debugging. If there's a bug in my code for adding an entry, **I can hit that URL again and again to test—I don't need to fill out the form again and again.**

Suggested strategy for solving this problem

Here's a possible way to approach this problem:

1. First do problem 2, and get a working `load_entries()`.
2. Modify your `a5/blog.php` to use `load_entries()`. If you'd like, start with my solution, in `$a6/whm-blog.php`. Note that the `blog.txt` date format has changed, to match what's produced by `type=date` input elements in Chrome. (For the curious: Search for date at caniuse.com to see which browsers support `type=date`.)
3. Add display of tags, making them links that reference `blog2.php?filter=TAG`
4. Get hits on `blog2.php?filter=TAG` working, displaying only entries with that tag.
5. Just for testing, make `blog2.php?mode=tags` render a page that just shows the full set of tags found in `blog.txt`.
6. Get hits on `blog2.php?mode=new` simply rendering the data entry form.
7. Get a click on the form's Done button to hit `blog2.php?mode=add&...` with the right URL parameters. (Use `var_dump($_GET)` to check it.)
8. Almost done—now just implement the code that handles `mode=add` and appends the data to `blog.txt`. (See the "Appending to a file" section above.) Since you're writing to `blog.txt` you'll probably foul it up some number of times—I surely did. Have a backup of it handy to copy in after each boo-boo.

You'll find various screenshots in `$a6/blog2-*.png`. Be sure to test with `$a6/blog.txt`, not the old one in `$a5`.

My `blog2.php` has about 180 lines of PHP code, not counting comments and a multi-line `echo` (like on slide 41) that outputs some boilerplate and an embedded stylesheet. My `load_entries.php` has 32 lines of code at present.

Problem 4. (5 Project Points) `project1.html`

One of the thousand things I learned from my graduate school advisor, Ralph Griswold, is that people do their best work when they're working on something they're excited about. Here's a chance to maybe work on something you're excited about.

For 200 of the semester's 600 assignment points you are to create a web application of your own design. The final version will likely be due at 10:00pm on the last day of classes but for the first five of those 200 points I want you to come up with one or more possible ideas for your project.

Your application should make use of HTML, CSS, PHP, and JavaScript but the relative proportions will vary depending on what your app does. You'll of course need to use HTML to provide some sort of structure for the pages of your app. Pages should have at least a little styling. The server-side code should be in PHP but if you can convince me you're already a PHP expert and want to learn something new, I'll consider that. Your application must make some non-trivial use of disk-based storage, but it need not use MySQL (which we'll be talking about soon!); it might use a simple text file like `blog2.php`, or maybe a "NoSQL" database that interests you. Your application must make some non-trivial use of JavaScript.

Depending on what you choose as a project, the size of the "pie slices" for HTML, CSS, PHP, and JavaScript will vary. For example, your app might do some very significant back-end computation with PHP and have relatively minimal styling and client-side interactivity with JavaScript. On the other hand you might have a minimal back-end but lots of JavaScript and styling. This class is *Web Programming* so no matter what the the mix of PHP and JavaScript is, your application should require a significant amount of programming; a nicely styled but mostly static set of pages won't be acceptable.

Size-wise, I'd like your project to be something that requires about four times the work of the assignments we've had thus far, including this one. I figure that translate that into about 30-45 hours of work on the project for a CS junior. We'll continue to have weekly assignments (more or less) but they'll be smaller. You'll need to have the discipline to put some time into your project on a weekly basis, although assignments will have some small project milestones blended in with them.

One challenge with picking a project is that we haven't yet seen MySQL and JavaScript, so how are you to know what you can do with them? You'll see that MySQL is a relational database and what it facilitates is storing data in a tabular form, much like a set of spreadsheets. In an upcoming assignment, we'll convert the blog from using `blog.txt` to storing entries in a database. We'll have a table for entries, a table for tags, and a table that associates entries and tags. In brief, you can think of MySQL as providing the ability to create, read, update, and delete data records of various sorts.

JavaScript and the jQuery JavaScript library are the last things that we'll study in this course. JavaScript lets us do computations in the browser. For example, a5's `toss.php` could be rewritten in JavaScript and run entirely in the browser. A JavaScript-based version of a5's `blog.php` might simply get `blog.txt` from a server and generate all the HTML in the browser. JavaScript also lets us manipulate HTML and CSS. For example, we could write JavaScript code to reorder blog entries based on some criteria. Instead of a hitting a PHP program on a server to produce a reordered page, JavaScript would just manipulate the element tree, saving a "round-trip" to the server.

In a nutshell, however, you'll be able to create simple versions of anything you see on the web that's based on HTML/CSS/JavaScript (i.e., not using Adobe's Flash Player).

A lot of students will start from scratch but if you've already got a web project in progress on your own, I'll consider letting you build on that. Perhaps there's an opportunity to put a web-based face on a project for some other class, like 335.

I'll permit teams of two if the nature of an application allows clear separation of who did what. For example, if you wanted to create a small version of SurveyMonkey.com, one person might do a survey builder and the other might do results reporting.

To earn the five points of this problem, I'd like you to spend some time thinking about a possible project and write a few sentences about it. You're not locking into anything here; I'm just trying to get you to start the process.

As a tangible example of what I'd like you to write for this problem, here's a project I've had in mind for a while, with a little context:

"While working with Dr. Beal in SISTA on a version of AnimalWatch for the visually impaired I learned to read braille by sight. There are lots of braille flashcard programs on the web but none that I've found handle braille contractions. Contractions are things like dots 3-4-5 representing "ar" and the two cell sequence (4-5-6, 2-4-5-6) representing "world". Also, nothing I've found lets me make custom sets, like one of just punctuation symbols, for example.

"I propose to make a braille flashcard system that handles contractions, provides saveable custom sets of cards, and does some tracking of progress. If time permits I'll add support for multiple users and sharing of custom sets."

Submit your idea as `project1.html` in your `a6.zip`.

If you don't have any ideas, come and see us!

Problem 5. Extra Credit observations.html

Submit a styled-as-you-like HTML file named `observations.html` with...

(a) (1 point extra credit) An estimate of how long it took you to complete this assignment. To facilitate programmatic extraction of the hours from all submissions, please enclose the number of hours in a span with `class=hours`, like this:

```
I spent <span class=hours>8.5</span> hours on this assignment.
```

Other comments about the assignment are welcome, too. Was it too long, too hard, too detailed? Speak up! I appreciate all feedback, favorable or not.

(b) (1-3 points extra credit) Cite an interesting course-related observation (or observations) that you made while working on the assignment. The observation should have at least a little bit of depth. Think of me saying "Good!" as one point, "Interesting!" as two points, and "Wow!" as three points. I'm looking for quality, not quantity.

Turning in your work

Use the D2L Dropbox named `a6` to submit a zip file named `a6.zip` that contains all your work. If you submit more than one `a6.zip`, we'll grade your final submission. Here's the full list of deliverables:

```
shapes.php
load_entries.php
blog2.php
project1.html
observations.html (for extra credit)
```

Note that all characters in the file names are lowercase.

Have all the deliverables in the uppermost level of the zip. It's ok if your zip includes other files, too.

Miscellaneous

The HTML slides, the CSS slides, PHP slides through 147, in-class examples, and a little reading in HFHC about the `textarea` element should provide all you need to complete this assignment.

Point values of problems correspond directly to assignment points in the syllabus. In total this assignment represents 5.5% of your final grade in this course.

§a6 follows the convention of §aN on the previous assignments.

Remember that late assignments are not accepted and that there are no late days; but if circumstances beyond your control interfere with your work on this assignment, there may be grounds for an extension. See the syllabus for details.

My estimate is that it will take a typical CS junior from eight to twelve hours to complete this assignment. If you're a CS senior, you'll probably do it in less. If you've taken only a single programming class, you might be on the high end or beyond.

Keep in mind the point value of each problem; don't invest an inordinate amount of time in a problem or become incredibly frustrated before you ask for a hint or help. Remember that the purpose of the assignments is to build understanding of the course material by applying it to solve problems. If you reach the seven-hour mark, regardless of whether you have specific questions, it's probably time to touch base with us. Give us a chance to speed you up! **Our goal is that everybody gets 100% on this assignment AND gets it done in an amount of time that is reasonable for them.**

None of your results need to validate with either the HTML5 validator or the "Jigsaw" CSS validator, but you may find the CSS validator to be helpful in diagnosing mysterious problems with CSS. (The typical mysterious CSS problem is that a declaration is ineffective, like those on slide CSS slide 16.)

Remember that I award Bug Bounty points; if you find problems, there's a potential reward for reporting them. I prefer that bugs be reported by mail to me. I'll put up a "FAQs and Corrections" post on Piazza and update it as things come in.

Use the a6 folder for any Piazza posts about the assignment.