Problem Set 8: Mashup

due by 7:00pm on Fri 11/13

Lest you have heard scary things, allow me to confess that Fall 2008's Problem Set 8 was harder than intended. (Don't, um, tell Fall 2008's students.) Fall 2009's version is new and improved, a much better finale. Don't drop it!

Goals.

- Introduce you to XML, RSS, JavaScript, and Ajax.
- Expose you to objects and methods.
- Have you learn a real-world API.

Recommended Reading.

- `http://www.w3schools.com/xml/`
- `http://www.w3schools.com/rss/`
- `http://www.w3schools.com/js/`
- `http://en.wikipedia.org/wiki/Ajax_(programming)`
- `http://en.wikipedia.org/wiki/JSON`

NOTICE.

For this problem set, you are welcome and encouraged to consult "outside resources," including books, the Web, strangers, and friends, as you teach yourself more about XHTML, CSS, PHP, SQL, and JavaScript, so long as your work overall is ultimately your own. In other words, there remains a line, even if not precisely defined, between learning from others and presenting the work of others as your own.

You may adopt or adapt snippets of code written by others (whether found in some book, online, or elsewhere), so long as you cite (in the form of XHTML, CSS, PHP, or JavaScript comments) the origins thereof.

And you may learn from your classmates, so long as moments of counsel do not devolve into "show me your code" or "write this for me." You may not, to be clear, examine the source code of classmates. If in doubt as to the appropriateness of some discussion, contact the staff.

Academic Honesty.

All work that you do toward fulfillment of this course's expectations must be your own unless collaboration is explicitly allowed (e.g., by some problem set or the final project). Viewing or copying another individual's work (even if left by a printer, stored in an executable directory, or accidentally shared in the course's virtual terminal room) or lifting material from a book, website, or other source—even in part—and presenting it as your own constitutes academic dishonesty, as does showing or giving your work, even in part, to another student.

Similarly is dual submission academic dishonesty: you may not submit the same or similar work to this course that you have submitted or will submit to another. Nor may you provide or make available solutions to problem sets to individuals who take or may take this course in the future. Moreover, submission of any work that you intend to use outside of the course (*e.g.*, for a job) must be approved by the staff.

You are welcome to discuss the course's material with others in order to better understand it. You may even discuss problem sets with classmates, but you may not share code. In other words, you may communicate with classmates in English, but you may not communicate in, say, C. If in doubt as to the appropriateness of some discussion, contact the staff.

You may even turn to the Web for instruction beyond the course's lectures and sections, for references, and for solutions to technical difficulties, but not for outright solutions to problems on problem sets or your own final project. However, failure to cite (as with comments) the origin of any code or technique that you do discover outside of the course's lectures and sections (even while respecting these constraints) and then integrate into your own work may be considered academic dishonesty.

All forms of academic dishonesty are dealt with harshly.

Grades.

Your work on this problem set will be evaluated along three primary axes.

Correctness. To what extent is your code consistent with our specifications and free of bugs?
Design. To what extent is your code written well (*i.e.*, clearly, efficiently, elegantly, and/or logically)?
Style. To what extent is your code readable (*i.e.*, commented and indented with variables aptly named)?

Getting Started.

☐ Last problem set EVER.

☐ For this problem set, consider downloading and installing **Firefox**, **Firebug**, **Live HTTP Headers**, and **Web Developer** (in that order), each of which is available on the course's website under **Software**. Once installed, Firebug, Live HTTP Headers, and Web Developer will appear as options in Firefox's **Tools** menu. See if you can figure out how they each work, simply by playing. Odds are you'll find that all are valuable (though not necessary for this problem set).

☐ For this problem set, your work must ultimately behave the same on at least two major browsers:

- ☐ Google Chrome 2.x
- ☐ Firefox 3.x
- ☐ Internet Explorer 7.x or 8.x
- ☐ Opera 9.x
- ☐ Safari 4.x

Be sure, then, to test your work thoroughly with at least two browsers. It is fine, though, to rely on just one operating system. And it's fine if you notice slight aesthetic differences between the two browsers. Make sure that your teaching fellow knows which browsers to use whilst evaluating your work.

☐ SSH to `cloud.cs50.net` and recursively copy `~cs50/pub/src/psets/pset8/` into your own `~/public_html/` directory. (Remember how?) Then `cd` to `~/public_html/pset8/`. (Remember how?) Then run `ls`. You should see the below.

```
constants.php  index.html  mashup.js     proxy.php
import         logo.gif    progress.gif  styles.css
```

All of the work that you do for this problem set will reside in `~/public_html/pset8/`. Go ahead and `chmod` your files as follows. (Remember how?)

- ☐ `constants.php` and `proxy.php` should be readable and writable only by you
- ☐ `import` should be readable, writable, and executable only by you
- ☐ everything else should be readable and writable by you and only readable by everyone else

Now open up constants.php with Nano (or your favorite text editor) and notice that the values of three variables (`DB_NAME`, `DB_USER`, and `DB_PASS`) are currently missing. Fill in the blanks with the same values you used for Problem Set 7 (*i.e.*, those from `http://www.cs50.net/me/`).

Now head to the URL below, where `username` is your own username.[1] You should see a map![2]

`http://cloud.cs50.net/~username/pset8/`

[1] Note that this URL is equivalent to `http://www.cs50.net/~username/pset8/index.html`.
[2] The search box won't yet work!

Google Mashup.

☐ Your mission for this problem set is to implement a mashup that integrates Google Maps with Google News with a MySQL database containing 42,086 zip codes (and more). But first, some inspiration!

If you've a friend who owns a Nintendo Wii (that's connected to the Internet), see if you can invite yourself over there to do some "research." Ask your friend to start up the **News Channel**, then click **National News** with the Wiimote, then click any of the articles, then click **Globe** in the screen's bottom-right corner. Notice how you can spin the Earth by clicking **A** and dragging, thereby revealing stacks of articles from different cities and geographic areas. And by zooming in and out with **+** and **–**, you can reveal more or fewer stacks. By clicking a stack's icon, you can then read local news.

If you haven't said friend, watch this instead, paying close attention between 01:04 and 01:50:

`http://www.youtube.com/watch?v=uO6J8ryTKYk`

The challenge ahead isn't to implement precisely that interface but the spirit thereof in the (largely) two-dimensional world of Google Maps. Specifically, your mashup will present users with a Google Map, atop which will be a form. Upon submitting an address via that form, users will be whisked away to that location on the map, which will be sprinkled with markers representing news articles pertaining to the five (or fewer) largest cities within view. Clicking a marker will trigger a balloon to appear, inside of which will be links to those articles. Having trouble envisioning all that? Take a peek at the staff's solution, which (unlike your version at the moment) is fully functional:

`http://cloud.cs50.net/~cs50/pset8/`

By problem set's end, you too will have a tool whose URL you can share with family and friends back home (that they might actually find useful)!

Alright, where to begin?

☐ Surf on over to Google Maps at `http://maps.google.com/`. Input something like `Cambridge, MA` or `02138` into the page's form and click **Search Maps**. You should find yourself whisked away to a familiar area. Well that was easy. Notice, though, that the page's URL did not change. I wonder how they did that.[3]

Now input `28.410,-81.584` instead. Perhaps you'd rather be there?

It looks like Google Maps knows about cities, states, and zip codes as well as GPS coordinates (*i.e.*, longitude and latitude). Interesting.

[3] Cough cough, Ajax.

Better yet, Google Maps has an API via which we can embed and control maps in our own sites:

`http://code.google.com/apis/maps/`

But more on that in a bit. If unfamiliar with longitude and latitude, incidentally, feel free to read up, albeit in more detail than is necessary for this problem set:

`http://en.wikipedia.org/wiki/Longitude`
`http://en.wikipedia.org/wiki/Latitude`

☐ If only it were as easy to find the latest news in Cambridge as it is to find maps! Actually, maybe it is. Surf on over to Google News at `http://news.google.com/`. Click the link to **Advanced news search** to the right of the search box, and notice how you can **Return only articles about a local area**. Input something like `Cambridge, MA` or `02138`, and you should see news local to Cambridge. Take a look at that webpage's source code, though, as by selecting **View Source** (or the like) from, probably, your browser's **View** menu. Quite the mess, huh? Can't be much fun to try to "screen-scrape" all that XHTML, CSS, and JavaScript (by writing a program to parse it) in order to extract articles' titles and URLs for our mashup. Let's look for an easier way.

Close the window containing the webpage's source code and scroll to the bottom of the webpage itself. Ooo, an **RSS** link (to the right of the orange icon)! Click that icon or link. Odds are you'll see a webpage with a bunch of articles related to Cambridge. But it's not really a webpage. It's actually an RSS "feed" that your browser is presenting to you as though it were a webpage (because most browsers today are also RSS "readers").

Just like XHTML, RSS is a tag-based markup language. (Both are flavors of XML.) Whereas XHTML is used to structure and stylize webpages, though, RSS is used to syndicate news (and blog posts, tweets, and the like). What's nice about RSS is that it's simple to parse. In fact, RSS is an acronym for "Really Simple Syndication" (though some people prefer "Rich Site Summary").

Here's what an RSS feed generally looks like (sans data):

```
<rss version="2.0">
    <channel>
        <title></title>
        <description></description>
        <link></link>
        <item>
            <guid></guid>
            <title></title>
            <link></link>
            <description></description>
            <category></category>
            <pubDate></pubDate>
        </item>
        [...]
    </channel>
</rss>
```

In other words, an RSS feed contains a root element called `rss`, the child of which is an element called `channel`. Inside of `channel` are elements called `title`, `description`, and `link`, followed by one or more elements called `item`, each of which represents an article (or blog post, tweet, or the like). Each `item`, meanwhile, contains elements called `guid`, `title`, `link`, `description`, `category`, and `pubDate`. Of course, between most of these start tags and end tags should be actual data (*e.g.*, an article's actual title). If you'd like to learn more, head to:

```
http://cyber.law.harvard.edu/rss/rss.html
```

Yup, Harvard's own law school hosts the specification for the latest version of RSS.

When you clicked Google's **RSS** link, then, your browser downloaded a dynamically generated RSS feed, parsed it, and then displayed it as though it were XHTML (so that you can click each article's link). If you try to view the page's source, you'll see either XHTML or RSS; it depends on your browser.

Anyhow, take notice of the URL to which Google's **RSS** link led you (which, unfortunately, wraps on to two lines):[4]

```
http://news.google.com/news?pz=1&cf=all&ned=us&hl=en&as_scoring=r&geo=02138&as_ma
xm=11&as_qdr=a&as_drrb=q&as_mind=6&as_minm=10&cf=all&as_maxd=5&output=rss
```

Within that otherwise cryptic string should be a familiar value (*i.e.*, `02138`). Interesting. I bet we'll be able to fetch (via PHP) an RSS feed for any zip code we want simply by altering the value of the parameter called `geo`, after which we can parse the RSS (very easily) with PHP. Interesting, indeed.

More on that, too, in a bit.

☐ Alright, clearly we can search for and pan to different cities via Google Maps. And we can also fetch articles about cities via Google News. But, after panning to some city, how do we figure out the five largest cities within view? If only we had a big list of cities, states, zip codes, GPS coordinates, and population counts to tie everything together…

☐ Turns out we do! Look what we found with a bit of Googling.

```
http://www.zip-codes.com/zip-code-database.asp
```

Look how much data you can get for $79.95.[5] Don't worry, we'll pick up the tab. Notice, though, that the data comes in multiple formats, among them CSV.[6] That's perfect, because we can easily parse that programmatically![7]

[4] Don't worry if your browser changes `http://` to `feed://`.
[5] Why go Standard when you can have Deluxe? Actually, we wanted population counts, so we got upsold.
[6] `http://en.wikipedia.org/wiki/Comma-separated_values`
[7] Keep an eye out for such things in the future!

To get a sense of what we bought, download this sample:

`http://www.zip-codes.com/files/sample_database/zip-codes-database-DELUXE-SAMPLE.zip`

Inside that archive, you'll find a file called `zip-codes-database-DELUXE-SAMPLE.csv` (among others). Go ahead and open it with Excel (or any old text editor). Notice that the database contains 50 fields (*i.e.*, columns), each of which is defined for you in the CSV file's first row. If you open up `ZipCodeDatabaseSpecifications-Deluxe.pdf` from that same archive, you'll find that pages 2 and 3 elaborate on those fields' types. That same PDF, for reference, is available at the URL below.

`http://www.zip-codes.com/files/documents/ZipCodeDatabaseSpecifications-Deluxe.pdf`

You're welcome to examine the actual CSV file that we bought, but you may find that its 80,114 rows won't fit in some versions of Excel:[8]

`http://www.cs50.net/pub/share/pset8/zip-codes-database-DELUXE.csv`

By default, incidentally, Excel doesn't show leading zeroes in CSV files. And so some of the zip codes in these CSV files' first columns might appear to be fewer than five digits, even though they are not. If you open those same files with any old text editor, you'll see leading zeroes!

☐ Okay, so we can get maps from Google Maps, news from Google News, and zip codes (and more) from Zip-codes.com. It's time to start wiring these puzzle pieces together with a bit of PHP and JavaScript! That CSV file is a bit unwieldy, though, with its 80,114 rows. Moreover, you can't really search a CSV file (other than by iterating over each of its rows, a la Linear Search). So let's begin by importing that data into a MySQL database, where we can search for cities via `SELECT`.

☐ Thanks to its 80,114 rows and 50 columns, the CSV file we bought is 31 MB. Yet most of those fields you won't even need. Interesting though it may be to know how many Hawaiians lived in 02138 as of the 2000 Census, you don't really need to know that for this problem set.[9] Let's save some time and space by importing into a MySQL database only the fields that we need.

But let's use the right tool for the job. It's a pain to parse text files in C, so let's use PHP instead.

[8] Someone decided to limit older versions of Excel to 2^{16} = 65,536 rows (and 2^8 = 256 columns)!
[9] But the answer is 25!

Implement in `~/public_html/pset8/import`, a "script" (*i.e.*, an interpreted program) that imports these fields (and only these fields) from `zip-codes-database-DELUXE.csv` into a MySQL database:[10]

i. `ZipCode`
ii. `Population`
iii. `Latitude`
iv. `Longitude`
v. `State`
vi. `City`

The table into which you import these fields should be called `zips`, and it must live in a database called `username_pset8` that we've pre-created for you. Your table's fields, meanwhile, should be named (and capitalized) just as they are in the CSV file: `ZipCode`, `Population`, `Latitude`, `Longitude`, `State`, and `City`. You may connect to your database using the same username (`DB_USER`) and password (`DB_PASS`) that you used for Problem Set 7.

What data types to use for each of your fields? Look at page 2 of 16 of that PDF from Zip-codes.com (*i.e.*, `ZipCodeDatabaseSpecifications-Deluxe.pdf`) for some hints. Take care not to allocate more space than you need to for fields. And go ahead and define `ZipCode` as a `PRIMARY` key. Realize, though, that some rows in the CSV file are nearly identical. (Apparently, some cities have aliases.) So take care to import only those rows that Zip-codes.com considers "primary records." Ignore all non-primary records.

How to determine whether some row is a primary record? Again consult page 2 of 16 of that PDF for a discussion of `PrimaryRecord`. But there's no need to import `PrimaryRecord` itself into your table.

Once you have a schema (*i.e.*, design) in mind, you'll, of course, need to `CREATE` the table, certainly before you can `INSERT` any rows into to with your script. You are welcome to `CREATE` the table "at runtime" via a call to `mysql_query` in your own script or in advance via phpMyAdmin. Recall that phpMyAdmin is available at the URL below.

`https://cs50.net/phpMyAdmin/`

You'll probably want to use phpMyAdmin quite a bite while developing your script. Odds are you'll make a mistake at least once and want to `ALTER` or `DELETE` rows from your table so as to start fresh. You'll likely find phpMyAdmin's **Browse**, **Structure**, **SQL**, **Empty**, and/or **Drop** tabs of particular help.

Note, incidentally, that we did not ask you to name your script `import.php`. Whereas webservers generally require that PHP scripts' names end in `.php` for execution, your shell (*i.e.*, your prompt) only requires that they start with a "shebang," `#` followed by `!`, followed by the full path to `php` (PHP's interpreter).

[10] If you find, while implementing your mashup, that you would like to make use of other fields from the CSV file, you may modify `import` and `zips` to accommodate. For space's sake, though, do not import more fields than you actually plan to use.

How to find that full path? Execute the below.

```
which php
```

Aha! You should see that `php` lives in `/usr/bin/php` on `cloud.cs50.net`. Go ahead, then, and put precisely this line atop `import`:[11]

```
#!/usr/bin/php
```

Take care not to put any characters at all (even whitespace) before this shebang. Even with the shebang present, you still need to tell `php` to interpret what follows as actual code (rather than just text, a la XHTML, that should be outputted raw). The second line of your file should thus be:

```
<?
```

And your last line should be:

```
?>
```

Somewhere between those tags you'll want to have a line like

```
require_once("constants.php");
```

so that you have access to those constants you defined earlier.

Okay, implement `import`! Rather than access that CSV file via its URL on `cs50.net`, though, read from the cloud's local copy in `/home/cs50/pub/share/pset8/` (much like you did `card.raw` for Problem Set 5 and `words` for Problem Set 6). After all, networks are even slower than disks. Odds are, you'll want to befriend PHP functions like `fopen`, `fgetcsv`, and `fclose`. And don't forget old friends like `mysql_connect`, `mysql_select_db`, `mysql_query`, and `mysql_fetch_assoc`.

Once done, you should be able to populate that table called `zips` in `username_pset8` by executing the command below after `chmod`'ing `import 700`:[12]

```
./import /home/cs50/pub/share/pset8/zip-codes-database-DELUXE.csv
```

As this usage implies, you should accept one (and only one) command-line argument: the path to a file to import. You might thus want to read up on, at least, `$argv`:

```
http://php.net/manual/en/reserved.variables.argv.php
```

Incidentally, you might want to try importing a much smaller file than ours first, lest you fill your table with thousands of mistakes. Just fill a text file with some values and commas!

[11] Alternatively, you could omit the shebang and execute `import` with `` `/usr/bin/php import` ``. But don't.
[12] It's annoying to type such long paths character by character, so do take advantage of tab completion: `http://en.wikipedia.org/wiki/Command_line_completion`

Even though `zip-codes-database-DELUXE.csv` contains 80,114 rows, you should find that it contains only 42,086 primary records, so your MySQL table, ultimately, should have 42,086 rows.

Once you've imported all 42,086 primary records, leave `zips` alone. You'll need it later.

☐ Okay, let's take a tour of the rest of this problem set's distro. Open up `index.html` first. It's this file that you saw when you visited `http://cloud.cs50.net/~username/pset8/` earlier.

Notice how the `head` element loads three files into memory. The first, `styles.css`, contains some CSS classes and selectors. The third, `mashup.js`, contains some JavaScript, which you'll soon augment with more. The second, meanwhile, comes from Google and implements (part of) the Google Maps API. In fact, if you'd like to see a whole lot of obfuscated JavaScript, pull up that URL (which, unfortunately, wraps on to two lines) in a browser:

```
http://maps.google.com/maps?file=api&amp;v=2&amp;sensor=false&amp;key=ABQIAAAA8ig
Yd929VTmOEMLNjNyP1xQIE4MyTYdaqjM5EsvAZQBbaRMlYRS9jJaf64VoDAABoTCl-_zJ-d13vg
```

Notice that one of the parameters in that URL is called `key`, the value of which is a long, cryptic string. We were given that string by Google when we signed up to use the API on `cloud.cs50.net` via the form here:[13]

```
http://code.google.com/apis/maps/signup.html
```

Now take a look at the `body` element below `head`. Notice how it registers two event handlers (*i.e.*, functions) for when the page loads and unloads; both are defined in `mashup.js`. The page itself is structured with four `div` elements. Atop the page is a logo, below which is a form. That form doesn't actually lead anywhere (*i.e.*, the value of action is empty), but submission thereof does induce a call to `go`, a function defined in `mashup.js`. Below the form is a progress indicator that's hidden by default. To see what you're missing, head to

```
http://cloud.cs50.net/~username/pset8/progress.gif
```

where `username` is your own username. Cute, eh? We made it ourselves.[14] The last `div` in `index.html` is just a placeholder for Google's map. Notice that we assigned to each `div` a unique `id` so that we can address each in `styles.css` and `mashup.js`.

☐ Now take a peek at `styles.css`. Mostly aesthetics in there; some comments explain. You're welcome to make changes as you see fit.

[13] If you use the same API on your own server some day, you'll need to sign up for your own `key`. No need to sign up for the API for this problem set; we've done so for you.
[14] Not true.

☐ Alright, now open `mashup.js`. It's on this file that you'll soon focus most of your attention. But for now, let's get a sense of what's there.

Atop this file are a few global variables, one of which defines some GPS coordinates. Let's personalize your map. Find the GPS coordinates of your hometown. This site (or perhaps Google or Wikipedia) might help:

`http://itouchmap.com/latlong.html`

Modify the definition of `home` accordingly, save your changes, and then return to

`http://cloud.cs50.net/~username/pset8/`

and reload. If you don't see your home, time to double-check your hometown's coordinates.

Neat, eh? Okay, let's skim the rest of `mashup.js`. It looks like we've written a function called `addMarker` whose purpose in life is to add a marker (*i.e.*, red icon) at a given point (*i.e.*, `GLatLng`) that, when clicked, shows some XHTML. Sounds useful.

Ah, next in the file is that function called `go`; looks like we've left most of its implementation to you.

Next up is load, that function that gets called the moment `index.html` is loaded. It looks like it's this function that installs Google's map inside that `div` placeholder, though there's still work to be done in a bit.

Below `load` is `resize`, a function we wrote to resize your map anytime you alter the dimensions of your browser's window.

And then there's `unload`, which unloads Google's map.

Finally there's `update`, whose purpose in life is to lay down markers for the five (or fewer) largest cities within view. Looks like it also plants a marker atop your hometown. But it looks like it doesn't (yet) fetch any articles, but, before long, it's this function that will be making Ajax calls to `proxy.php`.

☐ It's now time to open up `proxy.php`. (How's that for a segue!) Much like we implemented a proxy to Yahoo Finance for you last week, so have we implemented a proxy to Google News.[15] Whereas Problem Set 7's proxy was was a function (`lookup`), this proxy lives on the Web at

`http://cloud.cs50.net/~username/pset8/proxy.php`

so that you can query it via Ajax. Let's see what it does.

[15] Don't worry if our proxy returns articles that are slightly older than those listed at Google News itself. The cloud caches responses from `news.google.com` for a few minutes in order to reduce load on Google's servers.

Open up `proxy.php` with Nano. This proxy first connects to your MySQL database and then declares an (empty) associative array. It then checks that it's been passed two parameters (`sw` and `ne`), each of which should be a comma-separated pair of floating-point values. In other words, this proxy expects to be queries like this one (which, unfortunately, wraps on to two lines):

```
http://cloud.cs50.net/~username/pset8/proxy.php?sw=42.34382782918463,
-71.19930267333984&ne=42.40799515480466,-71.03038787841797
```

Together, `sw` and `ne` define a map's bounds, a rectangle defined by the GPS coordinates of a map's bottom-left and top-right coordinates, respectively. As you may have guessed, `proxy.php` proceeds to use those coordinates to determine the five (or fewer) largest cities within view (*i.e.*, within bounds). And it does so by querying `zips`, that table you made! It then queries Google News for articles about each of those cities, thereafter parsing Google's RSS using PHP's SimpleXML API.[16] Ultimately, the proxy returns a JavaScript array with (up to) five JavaScript objects within, each of which represents a city, within which is a JavaScript array of articles. In other words, the proxy returns JSON![17]

It's a bit hard to picture all those arrays, so let's look at some actual JSON. If you haven't already, go ahead and pull up this URL (or similar) in your browser, where, as always, `username` is your own username:

```
http://cloud.cs50.net/~username/pset8/proxy.php?sw=42.34382782918463,
-71.19930267333984&ne=42.40799515480466,-71.03038787841797
```

Wow, what a mess. Let's try pretty-printing that same output. Head to

```
http://www.cerny-online.com/cerny.js/demos/json-pretty-printing
```

in a separate window, paste into that page's **Input** box all of that JSON, then click **Print pretty.** Much clearer output, no?

If you scroll down, you should see that the five largest cities between `42.34382782918463, -71.19930267333984` and `42.40799515480466,-71.03038787841797` are Cambridge (02138), Somerville (02143), Boston (02108), Brighton (02135), and Everett (02149). Each city is represented as a JavaScript object (per the `{` and `}` around each); together, they compose a JavaScript array (per the `[` and `]` in which they're enclosed). Within each object are six keys (and values): `ZipCode`, `City`, `State`, `Latitude`, `Longitude`, and `articles`, the last of whose values is a JavaScript array, each of whose members is a JavaScript object with two keys (`link` and `title`) and values. Realize that we created this structure in PHP using associative arrays, thereafter converting our structure to JSON via `json_encode`.[18] Though confusing at first (and maybe second) glance, what's nice about JSON is just how compact it is. Consider how much more verbose RSS is!

[16] `http://php.net/manual/en/book.simplexml.php`

[17] Generally, you should output a MIME type of `application/json` for JSON, but we're using `text/plain` so that you can see the proxy's output in your browser.

[18] PHP does support objects, but associative arrays whose keys are non-numeric are treated as though they are objects anyway by `json_encode`, so the end result is the same.

☐ Alright, it's your turn again. At the moment, your map is rather lacking in features:

`http://cloud.cs50.net/~username/pset8/`

You can't toggle between **Map**, **Satellite**, and **Hybrid** mode as you can with the real Google Maps. And you can't even zoom! Let's fix. Head to the URL below for some background on the Google Maps API:

`http://code.google.com/apis/maps/documentation/introduction.html`

Don't fret if you don't understand everything. That page should paint a reasonable picture of how the API works, though. Allow me to suggest that you click **View example** wherever you see it (thereafter viewing the example's source code), as you might find it easier to learn from examples than from the documentation's prose. Next head to

`http://code.google.com/apis/maps/documentation/controls.html`

and read up on the "controls" that you can add to your map. Go ahead and add at least one control to your map. Odds are you'll want to modify `load` in `mashup.js`. Feel free to customize your map further as you see fit!

☐ Incidentally, you may find this Google Maps API Reference helpful as you proceed:

`http://code.google.com/apis/maps/documentation/reference.html`

☐ Okay, it's time to make your search box work. Go ahead and read up on Google's "geocoding" service at:

`http://code.google.com/apis/maps/documentation/services.html#Geocoding`

You can probably stop reading after clicking **View example (geocoding-simple.html)**.

Now make that search box work! Specifically, empower users to search for and pan to a particular city by inputting an address (however precise or imprecise) into your search box. Rely on Google's geocoding service for this feature; do not try to query your own `zips` table or contact our proxy. Unlike our proxy (which is designed to find the five largest cities within some bounds), Google's geocoding service is designed to convert addresses (however precise or imprecise) to GPS coordinates (*i.e.*, `GLatLng` objects).

Odds are you'll want to incorporate base your implementation of `go` in `mashup.js` on the source code for `geocoding-simple.html`. But no need to plant a marker like `geocoding-simple.html` does.

☐ Alright, you're in the home stretch now. It's time to finish your mashup.

At the moment, your search box only whisks the user away to their desired location. But notice that `go` already calls `update`, unless you deleted that line, in which case it's time to go add it back! But, at the moment, update only plants one marker for your hometown. It's time to plant up to five more, one for each of the largest cities within view. Clicking one of those markers should trigger an info window (*i.e.*, balloon) to appear, the contents of which are the city's name plus some links to breaking news in that area.[19] Although the formatting thereof is entirely up to you, you're welcome to look to the staff's solution for inspiration:

```
http://cloud.cs50.net/~cs50/pset8/
```

You may also want to refer back to Week 9's Ajax examples for hints on how to contact our proxy.[20] And don't forget that we've written an `addMarker` function for you. (Did you notice that we ourselves used it to plant your hometown's marker?)

Oh, don't forget about that progress indicator that's hidden by default! Take care to reveal that progress indicator whenever your code is waiting for our proxy to respond to a query from you; hide it again once you've received a response.

☐ And now for a personal touch. Head to

```
http://www.ajaxload.info/
```

and generate your own progress indicator. Click **Download it!**, rename it `progress.gif`, and then upload it via SFTP to `~/public_html/pset8/`, overwriting the distro's own `progress.gif`. Take care to `chmod` your version `644` if necessary. Head back to

```
http://cloud.cs50.net/~username/pset8/
```

and reload the page. Perform a search and ensure that you see your new indicator. (If it's not aligned quite as well as the original, you may need to tweak `styles.css`.) All set?

Ah. Such a tacky note to end on.

☐ Under no circumstances should your mashup generate JavaScript runtime errors. (Be sure to check functions' return values as is appropriate!) For help chasing down bugs in your JavaScript code, incidentally, you may find FireBug's **Console** tab invaluable.

[19] No need to worry about race conditions, whereby the user induces multiple Ajax calls at once (as by searching too fast or resizing their window too much).
[20] Alternatively, you're welcome to use Google's `GXmlHttp` object:
`http://code.google.com/apis/maps/documentation/services.html#XML_Requests`

Submitting Your Work.

☐ Don't forget that your work must behave the same in at least two major browsers!

☐ Ensure that your work is in `~/public_html/pset8/`. Then execute the command below, where `username` is your own username, in order to dump your database to disk for us; input your database's password (*i.e.*, the value of `DB_PASS`) when prompted.

```
mysqldump -u username -p username_pset8 > ~/public_html/pset8/username_pset8.sql
```

Now submit your work by executing the command below.

```
cs50submit pset8
```

Thereafter, follow any on-screen instructions until you receive visual confirmation of your work's successful submission. You may re-submit as many times as you'd like; each resubmission will overwrite any previous submission. But take care not to re-submit after the problem set's deadline, as only your latest submission's timestamp is retained.

☐ For simplicity, your TF may want to examine your code in situ, so don't modify your work even after you submit without first checking with your TF.

kthxbai