Problem Set 7: C$50 Finance

due by noon on Thu 11/3

Per the directions at this document's end, submitting this problem set involves submitting source code via `submit50` as well as filling out a Web-based form, which may take a few minutes, so best not to wait until the very last minute, lest you spend a late day unnecessarily.

Be sure that your code is thoroughly commented
to such an extent that lines' functionality is apparent from comments alone.

Goals.

- Get you on teh interwebs.
- Introduce you to HTML, CSS, PHP, and SQL.
- Teach you how to teach yourself new languages.

Recommended Reading.

- `http://www.w3schools.com/html/`
- `http://www.w3schools.com/css/`
- `http://www.w3schools.com/php/`
- `http://www.w3schools.com/sql/`

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 HTML, CSS, PHP, and SQL, 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 CSS, HTML, or PHP 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 in writing by the course's instructor. Collaboration in the completion of problem sets is not permitted unless otherwise stated by some problem set's specification.

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 or soliciting the work of another individual. 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 course's instructor.

You may 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. If the course refers some matter to the Administrative Board and the outcome for some student is *Admonish*, *Probation*, *Requirement to Withdraw*, or *Recommendation to Dismiss*, the course reserves the right to impose local sanctions on top of that outcome for that student that may include, but not be limited to, a failing grade for work submitted or for the course itself.

Grades.

Your work on this problem set will be evaluated along four axes primarily.

Scope. To what extent does your code implement the features required by our specification?
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)?

All students, whether taking the course Pass/Fail or for a letter grade, must ordinarily submit this and all other problem sets to be eligible for a passing grade (*i.e.*, Pass or A to D–) unless granted an exception in writing by the course's instructor.

Getting Started.

☐ Only two problem sets to go!

☐ Even though HTML and CSS are standards (maintained by the World Wide Web Consortium, otherwise known as the W3C), browser manufacturers sometimes disagree when it comes to their interpretations of those standards. And so websites don't always behave or look the same across all browsers, at least not without some effort (particularly when JavaScript is involved, as you may discover in Problem Set 8). For this problem set, your work must ultimately behave the same on the latest version of at least two major browsers:

- ☐ Chrome
- ☐ Firefox
- ☐ Internet Explorer
- ☐ Opera
- ☐ Safari

Be sure, then, to test your work thoroughly with at least two of those browsers. One could be Firefox within the appliance; the other could be some browser on your own computer (besides Firefox). Or you can use two different browsers on your own computer. It's fine if you notice slight aesthetic differences between the two browsers so long as your work functions properly on both. Make sure that your teaching fellow knows which browsers to use whilst evaluating your work.

☐ Launch VirtualBox (as by double-clicking its icon wherever it's installed), and then boot the appliance (as by single-clicking it in VirtualBox's lefthand menu, and then clicking **Start**).

Upon reaching John Harvard's desktop, open a terminal window (remember how?) and type the below, followed by Enter:

```
sudo yum -y update
```

Input **crimson** if prompted for John Harvard's password. For security, you won't see any characters as you type. Realize that updating the appliance in this manner requires Internet access. If on a slow connection (or computer), it might take a few minutes to update the appliance. Don't worry if the process seems to hang if it decides to update a "package" called `cs50-appliance`; that one can take several minutes.

If you see messages like **Couldn't resolve host** or **Cannot retrieve metalink for repository**, those simply mean that the appliance doesn't currently have Internet access. Sometimes that happens if you've just awakened your computer from sleep or perhaps changed from wireless to wired Internet or vice versa. If your own computer does have Internet access (which you can confirm by trying to visit some website in a browser on your own computer) but the appliance does not (which you can confirm by trying to visit the same with Firefox within the appliance), try restarting the appliance (as by clicking the green icon in its bottom-right corner, then clicking **Restart**).[1] If,

[1] Alternatively, you can try typing:

upon restart, the appliance still doesn't have Internet access, head to `https://manual.cs50.net/FAQs` followed by `http://help.cs50.net/` for help!

Once the appliance has been updated, you should see **Complete!** in your terminal window. If there was nothing to update, you'll see **No packages marked for Update** instead.

☐ Just to be sure that everything worked, go ahead and execute that very same command again in a terminal window (though not while its first invocation is still running):

```
sudo yum -y update
```

Again, input **crimson** if prompted for John Harvard's password. (If only a few minutes have passed since the last update, you might not even be prompted.) You should now see **No packages marked for Update**, which means that your appliance is now up-to-date! If you see some error instead, try once more, try to restart the appliance and then try once more, then head to `https://manual.cs50.net/FAQs` followed by `http://help.cs50.net/` as needed for help!

☐ Now go ahead and open up a terminal window (whether by opening gedit via **Menu > Programming > gedit** or by opening Terminal itself via **Menu > Programming > Terminal**). Then execute

```
mkdir ~/public_html
```

in order to create a directory called `public_html` within John Harvard's home directory. Then download this problem set's distro into that directory by executing

```
cd ~/public_html
git clone http://cdn.cs50.net/2011/fall/psets/7/pset7.git
```

You should see **`Cloning into pset7...`** and then your prompt again. If you instead see **`fatal`** followed by **`not found: did you run`**, odds are you made a typo. Best to try again!

Once successful, you should find that you have a brand-new `pset7` directory inside of your home directory's `public_html` directory. You can confirm as much with:

```
cd ~/public_html
ls
```

Next make each of `~`, `~/public_html`, and `~/public_html/pset7` "world-executable" by executing

```
chmod a+x ~
chmod a+x ~/public_html
chmod a+x ~/public_html/pset7
```

`sudo service network restart`
But if that doesn't work, best to restart the appliance.

so that the appliance's web server (and you, from your own computer) will be able to access your work.

Now navigate your way to `~/public_html/pset7` (wherein lies, recall, this problem set's distro) by executing the command below.

```
cd ~/public_html/pset7
```

Then run `ls`. You should see the below.

```
css/  images/  includes/  index.php  login.php  login2.php  logout.php
```

Let's make two of those directories world-executable. In fact, let's `chmod` both at the same time (by enumerating them on the command line, one after the other) to save some keystrokes:

```
chmod a+x css images
```

Now let's make some files within those directories world-readable:

```
chmod a+r css/* images/*
```

Recall that `*` is a "wildcard character," so `css/*` means "all files within the css directory" and `images/*` means "all files within the images directory."

For security's sake, don't make `includes` world-executable or its contents world-readable, as they shoudn't be accessible to the whole world (only to your PHP code, as you'll soon see). In fact, even more secure would be to put `includes` outside of `public_html` altogether, but for simplicity, let's leave it where it is.

Just to be clear now, realize that you have not created a directory called `pset7` in your home directory for this problem set. Rather, you have created a directory called `pset7` in `~/public_html/`. All of the work that you do for this problem set will reside in `~/public_html/` and `~/public_html/pset7/`.

☐ Alright, time for a test! Open up Firefox within the appliance and visit:[2]

```
http://localhost/~jharvard/pset7/
```

You should find yourself redirected to `login.php`, where an apology (regarding **jharvard_pset7**) awaits! Not to worry. That just means you don't have a database called **jharvard_pset7** yet. (If you instead see **Forbidden**, odds are you missed a step earlier; best to try all those `chmod` steps again.) Let's create the database that's missing.

[2] Incidentally, you can also access C$50 Finance within the appliance at `http://127.0.0.1/~jharvard/pset7/`, since 127.0.0.1 is the appliance's (and most computers') "loopback" address.

Head to

```
http://localhost/phpMyAdmin/
```

using Firefox within the appliance to access phpMyAdmin, a Web-based tool (that happens to be written in PHP) with which you can manage MySQL databases. Log in as John Harvard if prompted (with a username of **jharvard** and a password of **crimson**). You should then find yourself at phpMyAdmin's main page. In phpMyAdmin's top-left corner, you should see **No databases**. Normally, you can create a database by clicking phpMyAdmin's **Databases** tab, but you can also execute some SQL commands manually. Go ahead and visit

```
http://cdn.cs50.net/2011/fall/psets/7/pset7.sql
```

using Firefox within the appliance, and you should see a whole bunch of SQL. Highlight it all, then select **Edit > Copy** (or hit ctrl-C), then return to phpMyAdmin. Click phpMyAdmin's **SQL** tab, and paste everything you copied into that page's big text box. Skim what you just pasted to get a sense of the commands you're about to execute, then click **Go**. You should then see a green banner, proclaiming **Your SQL query has been executed successfully**. In phpMyAdmin's top-left corner, you should now see link to a database called **jharvard_pset7**, beneath which is a link to a table called **users**. But more on those later.

Return to

```
http://localhost/~jharvard/pset7/
```

and reload that page. Instead of an apology, you should now see the login page for your own copy of C$50 Finance! As excited as you may be, don't try to register or log in just yet.

☐ Recall that the appliance also has a private IP address, 192.168.56.50, via which you can access it, both within the appliance itself and from your own computer (but not from some other computer on the Internet). Confirm as much by visiting

```
http://192.168.56.50/~jharvard/pset7/
```

using Firefox within the appliance. You should again find yourself redirected to C$50 Finance's login page.

Now open up a browser on your own computer and visit the same URL:

```
http://192.168.56.50/~jharvard/pset7/
```

You should again see the same. Note that you cannot access the appliance from your own computer via the `localhost` URL, since, when using a browser on your own computer, `localhost` refers to your own computer, which probably isn't running a web server!

If unable to access the appliance from your own computer via 192.168.56.50, be sure that you haven't disabled any of the appliance's network adapters (perhaps long ago). To check, shut down

the appliance (if running), then single-click the appliance in VirtualBox's lefthand menu to highlight it, click **Settings**, click **Network**, and make sure that **Enable Network Adapter** is checked for each of **Adapter 1**, **Adapter 2**, and **Adapter 3**. Then re-start the appliance. If you encounter any error messages, consult

```
https://manual.cs50.net/VirtualBox
```

for some troubleshooting tips. And turn to `help.cs50.net` if you still need a hand!

Home, sweet home page.

☐ It's time to make yourself a home page! Create a file called `index.html` with gedit, save it in `~/public_html`, and fill that file with valid HTML (version 5). In other words, whip yourself up a home page. Works of art, though encouraged, are by no means required. I didn't exactly set the artistic bar very high in lecture, after all. So long as your HTML is valid, your home page may contain as much or as little actual content as you would like.

When ready to examine your masterpiece (or work in progress), save your file, then execute the command below within `~/public_html`. (Recall that the `l` in `-l` is a lowercase L.)

```
ls -l
```

The output you see should resemble the below.

```
-rw-r--r-- 1 jharvard students   50 Oct 28 19:01 index.html
drwx--x--x 5 jharvard students 4096 Oct 28 19:01 pset7/
```

In the past, you've probably ignored the sequence of ten symbols (mostly hyphens) prefixing each line of `ls`'s long output. No longer! Recall that each sequence represents a set of "permissions" that govern who (besides you) can read (`r`), write (`w`), or execute (`x`) access some file or directory.[3,4] Linux lets you specify separate permissions for a file's or directory's owner (*i.e.*, you), for a file's or directory's group (*e.g.*, you plus all other students), and for the "world" (*i.e.*, anyone with access to your appliance).

Confirm that your homepage is indeed world-readable (at least by anyone with access to the appliance) by visiting the URL below, either with Firefox within the appliance or with a browser on your own computer.

```
http://192.168.56.50/~jharvard/
```

[3] To read a file means to, well, read its contents; to read a directory means to list its contents. To write to a file means to change its contents; to write to a directory means to add another file or directory to it. To execute a file means to run it like a program; to execute a directory means to enter it, as with `cd`.

[4] The first symbol in a sequence indicates whether the permissions describe a directory (`d`) or a symbolic link (`l`), the latter of which is an alias of sorts.

Wow, that page is ugly. (Okay, maybe it's not.) Make any improvements you'd like to `index.html`. You may certainly, but need not, employ CSS. Anytime you save changes, be sure to reload your page in your browser. You should not need to run `chmod` again for this particular file.

Ultimately, be sure that your HTML is valid according to the W3C's Markup Validation Service. To determine as much, visit

```
http://validator.w3.org/
```

using Firefox within the appliance, click **Validate by Direct Input**, then copy/paste your HTML from gedit into that website's text box, then click **Check**. Or, ctrl- or right-click somewhere on your web page (in Firefox within the appliance), then select **Web Developer > Tools > Validate Local HTML**, which will automate that same process. Do not select **Validate HTML**, since that option will instead send a URL that isn't publicly accessible to the validator rather than your actual HTML.

If you see a **Software error**, odds are the validator itself is just having some technical difficulties. (It seems to have a bug that occasionally rears its head.) If you encounter such, try reloading the page, clicking **Resend** if necessary.

Ultimately, the **Result** of checking your page for validity via the W3C's validator should be **Passed** or **Tentatively passed**, in which case you should see a friendly green banner. Warnings are okay. Errors (and big red banners) are not.

☐ Okay, time for a heads-up. Anytime you create a new file or directory in `~/public_html` or some subdirectory therein for this problem set, you'll want to set its permissions with `chmod`. Thus far, we've relied on `a+r` and `a+x`, but let's empower you with more precise control over permissions.

Henceforth, for any non-PHP file, `file`, that you create (or upload), execute

```
chmod 644 file
```

so that it's accessible via a browser (if that's indeed your intention).

For any PHP file, `file`, that you create, execute

```
chmod 600 file
```

so that it's accessible only by you (and the appliance's webserver). Recall that we don't want visitors to see the contents of PHP files; rather, we want them to see the output of PHP files once executed (or, rather, interpreted) by the appliance's web server.

And for any directory, `directory`, that you create, execute

```
chmod 711 directory
```

so that its contents are accessible via a browser (if that's indeed your intention).

What's with all these numbers we're having you type? Well, `644` happens to mean `rw-r--r--`, and so all non-PHP files are to be readable and writable by you and just readable by everyone else; `600` happens to mean `rw-------`, and so all PHP files are made readable and writable only by you; and `711` happens to mean `rwx--x--x`, and so all directories are to be readable, writable, and executable by you and just executable by everyone else. Wait a minute, don't we want everyone to be able to read (*i.e.*, interpret) your PHP files? Nope! For security reasons, PHP-based web pages are interpreted "as you" (*i.e.*, under John Harvard's username) in the appliance.[5]

Okay, still, what's with all those numbers? Well, think of `rw-r--r--` as representing three triples of bits, the first triple of which, to be clear, is `rw-`. Imagine that `-` represents `0`, whereas `r`, `w`, and `x` represent `1`. And, so, this same triple (`rw-`) is just `110` in binary, or `6` in decimal! The other two triples, `r--` and `r--`, then, are just `100` and `100` in binary, or `4` and `4` in decimal! How, then, to express a pattern like `rw-r--r--` with numbers? Why, with 644.

Actually, this is a bit of a white lie. Because you can represent only eight possible values with three bits, these numbers (`6`, `4`, and `4`) are not actually decimal digits but "octal." So you can now tell your friends that you speak not only binary, decimal, and hexadecimal, but octal as well.

Yahoo!

☐ If you're not quite sure what it means to buy and sell stocks (*i.e.*, shares of a company), surf on over to the URL below for a tutorial.

`http://www.investopedia.com/university/stocks/`

You're about to implement C$50 Finance, a Web-based tool with which you can manage portfolios of stocks. Not only will this tool allow you to check real stocks' actual prices and portfolios' values, it will also let you buy (okay, "buy") and sell (fine, "sell") stocks![6]

☐ Allow me to share some excerpts from my spam folder with you.

```
Savvy Stock Investor:

A5 Laboratories (ticker: AFLB.OB) Has Just Announced What Could Be One Of The Greatest
Medical Discoveries Of The 21st Century

This could be one of the most important medical advances in 80 years.

Mark my words, AFLB.OB may have the single greatest breakthrough in drug technology in my
lifetime.

Do NOT MISS this one.

Here's to solid investing,

Tim Fields,
Untapped Wealth
```

[5] For the curious, we're using suPHP (`http://www.suphp.org/`) with Apache (`http://httpd.apache.org/`).
[6] Per Yahoo's fine print, "Quotes delayed [by a few minutes], except where indicated otherwise."

☐ Wow, what a nice guy this Tim must be, giving me such a hot stock tip. Stupid Gmail for calling this spam! Let's get in on this opportunity now. Head on over to Yahoo! Finance at the URL below.

```
http://finance.yahoo.com/
```

Type the symbol for A5 Laboratories, `AFLB.OB`, into the text field in that page's top-left corner and click **GET QUOTES**. Odds are you'll see a table like the below.[7]

A5 LABORATORIES INC. (OTC BB: AFLB.OB)

Last Trade:	**0.01**	Day's Range:	**0.01 - 0.01**
Trade Time:	**9:30AM EDT**	52wk Range:	**0.01 - 0.49**
Change:	**0.01 (49.50%)**	Volume:	**16,000**
Prev Close:	**0.02**	Avg Vol (3m):	**59,417**
Open:	**0.01**	Market Cap:	**2.56M**
Bid:	**N/A**	P/E (ttm):	**N/A**
Ask:	**N/A**	EPS (ttm):	**-0.05**
1y Target Est:	**N/A**	Div & Yield:	**N/A (N/A)**

Quotes delayed, except where indicated otherwise. Currency in USD.

Wow, looks like this stock only costs a penny! That must be a good thing. Anyhow, notice how Yahoo reports not only a stock's most recent price (**Last Trade**) but also when the stock last changed hands (**Trade Time**), the percent by which the stock's price changed over the course of the most recent business day (**Change**), the most recent (business) day's opening price (**Open**), the most recent (business) day's high and low prices (**Day's Range**), and more.

Moreover, scroll down to the page's bottom, and you should see a toolbox like that below.

Looks like Yahoo lets you download all those values. Go ahead and click **Download Data** to download a file in CSV format (*i.e.*, as comma-separated values). Open the file in Excel or any text editor (*e.g.*, gedit), and you should see a "row" of values, all excerpted from that table. It turns out that the link you just clicked led to the URL below.

```
http://download.finance.yahoo.com/d/quotes.csv?s=AFLB.OB&f=sl1d1t1c1ohgv&e=.csv
```

[7] Ha, seems 14 people "Like" this stock!

Notice how A5's symbol is embedded in this URL (as the value of the HTTP parameter called `s`); that's how Yahoo knows whose data to return. Notice also the value of the HTTP parameter called `f`; it's a bit cryptic (and officially undocumented), but the value of that parameter tells Yahoo which fields of data to return to you. If curious as to what they mean, head to the URL below.

```
http://www.gummy-stuff.org/Yahoo-data.htm
```

It's worth noting that a lot of websites that integrate data from other websites do so via "screen scraping," a process that requires writing programs that parse (or, really, search) HTML for data of interest (*e.g.*, air fares, stock prices, *etc.*). Writing a screen scraper for a site tends to be a nightmare, though, because a site's markup is often a mess, and if the site changes the format of its pages overnight, you need to re-write your scraper.[8]

Thankfully, because Yahoo provides data in CSV, C$50 Finance will avoid screen scraping altogether by downloading (effectively pretending to be a browser) and parsing CSV files instead. Even more thankfully, we've written that code for you!

In fact, let's turn our attention to the code you've been given.

☐ Navigate your way to `~/public_html/pset7` and open up `index.php` with gedit. (Remember how?) You'll see HTML for a pretty simple page, the same page you tried to pull up earlier when testing your framework (just before you were redirected to `login.php`). Notice how we've structured the page, with three `div` elements inside of `body`, each of which happens to have a unique `id` attribute. Notice next the references to `styles.css` and `logo.gif`. Those files can be found in `~/public_html/pset7/css` and `~/public_html/pset7/images`, respectively. We placed those two files into subdirectories in the interests of keeping `~/public_html/pset7` tidy. As you proceed to implement C$50 Finance, you're welcome to drop additional files into either directory.

Go ahead and open up `styles.css`. That file defines styles for various HTML elements so that we don't have to include `style` attributes for those same elements in every PHP file. No need to master CSS for this problem set, but do know that you should not have more than one `div` element per page whose `id` attribute has a value of `bottom`, more than one `div` element per page whose `id` attribute has a value of `middle`, or more than one `div` element per page whose `id` attribute has a value of `top`; an `id` must be unique. In any case, you are welcome to modify `styles.css` as you see fit.

Anyhow, notice next that `index.php` "requires" (*i.e.*, includes) a file called `common.php` that can be found in `~/public_html/pset7/includes`. Any PHP file that you create for this problem set that's meant to be visited by a user must also contain, before any other code, that very same line, excerpted below.

```
require_once("includes/common.php");
```

[8] See `https://manual.cs50.net/Screen_Scraping` if curious as to how it can be done nonetheless.

Note that if you decide to place PHP files within subdirectories of `~/public_html/pset7`, you may need to specify a different path for `common.php` (*e.g.*, `../includes/common.php`).

Let's take a look at the code we're requiring via `require_once`. Navigate your way to `~/public_html/pset7/includes` and open up `common.php` with gedit. Because `index.php` requires this file (via that call to `require_once`), every one of this file's lines will be executed before anything else in `index.php`. The first few lines of actual code in `common.php` ensure that you'll be informed of errors in your own code via your browser. The call to `session_start` ensures that you'll have access to `$_SESSION`, a "superglobal" variable via which we'll remember that a user is logged in.[9,10] The next few lines "require" yet three other files; we'll return to those shortly. The next lines of code ensure that users will be required to log in to access most pages. The last lines of code ensure that you're connected to your database, where you'll store users' portfolios.

Alright, now open `constants.php` with gedit. In this file have we defined some global constants. Because all of your PHPs shall require `common.php`, which, in turn, requires `constants.php`, you will have access to this file's globals from each of your PHPs. Among those constants are `DB_USERNAME` and `DB_PASSWORD`, whose values are `jharvard` and `crimson`, respectively. By default, the appliance comes not only with MySQL preinstalled and running but also with an account for John Harvard. (Technically, that account is MySQL-specific and unrelated to John Harvard's account on the appliance itself, but we've made the former's username and password identical to the latter's for simplicity.) Also among those constants is `DB_SERVER`, the value of which is `localhost` because MySQL is indeed running on the local host (*i.e.*, on the appliance itself). And the value of `DB_NAME` is `jharvard_pset7`, which refers to that database that you created with phpMyAdmin earlier. The value of `YAHOO`, meanwhile, should look pretty familiar (though we did alter its parameters so that `s` would be last). Missing from `YAHOO`, though, is a value for that parameter `s`. Let's see why that is.

Open up `stock.php` with gedit, and you'll see something that resembles a C `struct`. Indeed, this code defines a structure (a "class" in PHP) called `Stock`, whose purpose is to encapsulate data related to a stock.[11] Although Yahoo provides more fields than those encapsulated in this structure, our framework, out of the box, provides only the basics.

Now take a look at `helpers.php` with gedit. You need not understand how all of that code works, but make sure you understand what its functions can do for you by reading, at least, comments therein. Notice, in particular, how `lookup` expects, as its sole argument, a stock's symbol, which it appends to `YAHOO` using PHP's concatenation operator (`.`) in order to download the right CSV.

[9] The calls to `preg_match` and `session_set_cookie_params` tell PHP to associate cookies with your `pset7` directory specifically.

[10] Even though HTTP is a "stateless" protocol, whereby browsers are supposed to disconnect from servers as soon as they're done downloading pages, "cookies" allow browsers to remind servers who they (or, really, you) are on subsequent requests for content. PHP uses "session cookies" to provide you with `$_SESSION`, an associative array in which you can store any data to which you'd like to have access for the duration of some user's visit. The moment a user ends his or her "session" (*i.e.*, visit) by quitting his or her browser, the contents of `$_SESSION` are lost for that user specifically because the next time that user visits, he or she will be assigned a new cookie!

[11] By convention, classes' names are usually capitalized.

Finally, take a peek at `apology.php` with gedit. This file serves as a "template" for `apologize` in `helpers.php` so that, via just one function, you can apologize to users for (*i.e.*, report) all sorts of problems.

Alright, let's examine just three more files, each of whose names ends in `.php`. Navigate back to `~/public_html/pset7`, and open up `login.php` with gedit. Recall that you were redirected to this page when you tried to pull up `index.php` with your browser. Notice that this file doesn't contain too much code. In fact, much like `index.php`, it's almost entirely HTML. But it's that HTML that implements that login page that you saw. Note that it lays out a form using a table.[12] Notice next the line excerpted below.

```
<form action="login2.php" method="post">
```

This line instructs your browser to "submit" the form's data to `login2.php` via POST. It must be `login2.php`, then, that handles actual authentication of users. Let's check. Open up `login2.php`.

It turns out that `login2.php` isn't terribly long. Its first line of code, just like `index.php` and `login.php`, requires that file called `common.php`. Its next lines of code "escape" the user's inputted username for safety using `mysql_real_escape_string`, lest C$50 Finance's database fall victim to a "SQL injection attack," whereby a user submits SQL instead of a username and/or password. See `http://www.php.net/mysql_real_escape_string` for reference.

The next line of code prepares a string of SQL as follows.

```
$sql = "SELECT * FROM users WHERE username='$username'";
```

To be clear, suppose that President Skroob tries to log into C$50 Finance by inputting his username and password.[13] That line of code will assign to `$sql` the value below.

```
SELECT * FROM users WHERE username='skroob'
```

Perhaps needless to say, `login2.php`'s next line of code executes that `SELECT` with `mysql_query`, storing the "result set" (*i.e.*, rows returned) in a variable called `$result`. Only if Skroob's `username` exists in the table, though, will the database return an actual row. And, so, if `mysql_num_rows` returns `1`, Skroob indeed has an account. But we still need to confirm that he knows his own password, lest somebody else be trying to hack in! And so we "fetch" the returned row from the result set so as to compare a "hash" of Skroob's inputted password against a hash in the database. (Recall that it's best to store hashes of passwords rather than passwords themselves so that if a server is compromised by some adversary, he or she does not obtain everyone's password.) If those hashes indeed match, `login2.php` proceeds to "remember" that Skroob is logged in by storing his numeric user ID (`id`) in `$_SESSION`; it then redirects him to

[12] Many developers frown upon using tables for layout, but they're worth using sometimes.
[13] `http://en.wikipedia.org/wiki/Spaceballs#Spaceballs`

`index.php`, where his portfolio (once you implement it) awaits![14] If, however, Skroob inputted the wrong username or password, he is instead informed via a call to `apologize`.

Incidentally, why does this redirection back to `index.php`, upon successful authentication, not result in an infinite loop? Well, recall that `index.php` requires `common.php`, which contains the following code.

```
if (!preg_match("/\/login|logout|register)\d*\.php$/", $_SERVER["PHP_SELF"]))
{
    if (!isset($_SESSION["id"]))
        redirect("login.php");
}
```

Though a bit scary (I prefer "elegant"), that code simply asks whether `$_SESSION["id"]` has been assigned any value (*e.g.*, Skroob's user ID). If not, it must be that no one's logged in, else `login2.php` would have assigned it a value, and so we had best redirect traffic to `login.php` (by calling `redirect`, a function defined in `helpers.php`). If, though, `$_SESSION["id"]` is indeed set, we won't redirect but will, instead, leave the logged-in user wherever he or she is. Of course, if the user is already at `login.php`, `logout.php`, or `register.php`, this code won't redirect either, thanks to the "regular expression" that we've passed to `preg_match`.[15]

Now, how do we enable Skroob to log out? Why, by including in `index.php` a hyperlink to `logout.php`! Take a look at the latter. Note that it calls `logout`, a function defined in `common.php`. (We call that same function atop `login.php`.) Alternatively, Skroob can simply quit his own browser, as `$_SESSION` is lost in that case as well.

☐ Phew, that was a lot. Go have a snack.

☐ Alright, let's talk about that database we keep mentioning. So that you have someplace to store users' portfolios, the appliance comes with a MySQL database (`jharvard_pset7`).[16] We've even pre-populated it with a table called `users`! Let's take a look.

Head back to

`http://192.168.56.50/phpMyAdmin/`

to access phpMyAdmin. Log in as John Harvard if prompted (with a username of **jharvard** and a password of **crimson**). You should then find yourself at phpMyAdmin's main page, in the top-left corner of which is that table called **users**. Click the name of that table to see its contents. Ah, some familiar folks. In fact, there's President Skroob's username and a hash of his password (which is the same as the combination to his luggage)!

[14] Rather than "remember" users by way of their usernames (which are, by nature, strings), you'll see that we instead rely, for efficiency's sake, on "user IDs" (which are integers) that uniquely identify users.
[15] `http://en.wikipedia.org/wiki/Regular_expression`
[16] MySQL is a free, open-source database that CS50 apps, Facebook, and lots of other sites use.

Now click the tab labeled **Structure**. Ah, some familiar fields. Recall that `login2.php` generates queries like the below.

```
SELECT id FROM users WHERE username='skroob'
```

As phpMyAdmin makes clear, this table called users contains three fields: `id` (the type of which is an `INT` that's `UNSIGNED`) along with `username` and `password` (each of whose types is `VARCHAR`). It appears that none of these fields is allowed to be `NULL`, and the maximum length for each of each of `username` and `password` is `255`. A neat feature of `id`, meanwhile, is that it will `AUTO_INCREMENT`: when inserting a new user into the table, you needn't specify a value for `id`; the user will be assigned the next available `INT`. Finally, per the rows below **Indexes**, it appears that this table's `PRIMARY` key is `id`, the implication of which is that (as expected) no two users can share the same user ID.[17] Of course, `username` should also be unique across users, and so we have also defined it to be so (per the additional **Yes** under **Unique**). To be sure, we could have defined `username` as this table's primary key. But, for efficiency's sake, the more conventional approach is to use an `INT` like `id`. Incidentally, these fields are called "indexes" because, for primary keys and otherwise unique fields, databases tend to build "indexes," data structures that enable them to find rows quickly by way of those fields.

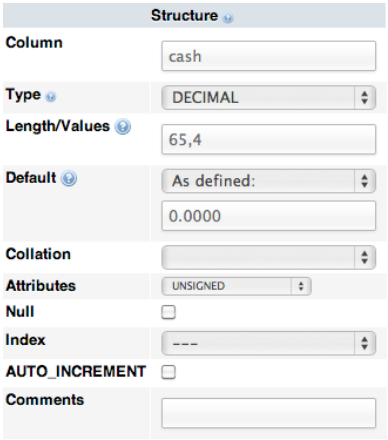
Make sense? Okay, head on back to

```
http://192.168.56.50/~jharvard/pset7/login.php
```

and try to log in as President Skroob (whose username and password should be quite familiar by now). If successful, you'll find yourself at `index.php`, where (for the moment) very little awaits.

☐ Head on back to phpMyAdmin and click the tab labeled **Structure** for that table called `users`. Let's give each of your users some cash. In that page's middle is a form with which you can **Add** ... **column(s)** ... **After** another. Click the radio button immediately to the left of **After**, select `hash` from the drop-down menu, then click **Go**.

Via the form that appears, define a field of type `DECIMAL` called `cash` using the settings depicted below, then click **Save**.



[17] A primary key is a field with no duplicates (*i.e.*, that is guaranteed to identify rows uniquely).

If you pull up the documentation for MySQL at

```
http://dev.mysql.com/doc/refman/5.5/en/numeric-types.html
```

you'll see that the `DECIMAL` data type is used to "store exact numeric data values." A length of `65,4` for a `DECIMAL` means that values for `cash` can have no more than 65 digits in total, 4 of which can be to the right of the decimal point. (Ooo, fractions of pennies. Sounds like *Office Space*.)

Okay, return to the tab labeled **Browse** and give everyone $10,000.00 manually.[18] The easiest way is to click **Check All**, then click **Change** to the right of the pencil icon. On the page that appears, change 0.0000 to 10000.0000 for each of your users, then click **Go**. Won't they be happy!

☐ It's now time to code! Let's empower new users to register.

Return to a terminal window and navigate your way to `~/public_html/pset7`. Execute

```
cp login.php register.php
```

followed by

```
cp login2.php register2.php
```

to jumpstart this process.[19] Then use `chmod` to ensure all permissions are set properly. Open up `register.php` and change the `head`'s `title` as you see fit. Then change the value of `form`'s `action` attribute from `login2.php` to `register2.php`. Next add an additional row to the HTML table containing a new field called `password2`. (We want registrants to type passwords twice to discourage mistakes.) Finally, change the `submit` button's value from `Log In` to `Register` and make that bottommost anchor (*i.e.*, link) point back to `login.php` (so that users can navigate away from this page if they already have accounts).

Alright, let's take a look at your work! Bring up

```
http://192.168.56.50/~jharvard/pset7/login.php
```

and click that page's link to `register.php`. If that page appears broken (or perhaps simply ugly), feel free to make further tweaks, saving your changes, thereafter reloading the page.

[18] In theory, we could have defined `cash` as having a default value of `10000.000`, but, in general, best to put such settings in code, not your database, so that they're easier to change.

[19] You are welcome, particularly if among those more comfortable, to stray from these filename conventions and structure your site as you see fit, so long as your implementation adheres to all other requirements.

Once the page looks okay, return to your text editor and open up `register2.php`. Needless to say, you need to replace the code there so that it actually registers users. Allow us to offer some hints.

i. If `$_POST["username"]` or `$_POST["password"]` is blank or if `$_POST["password"]` does not equal `$_POST["password2"]`, you'll want to return to the registrant a page that apologizes, explaining at least one of the problems.
ii. To insert a new user into your database, you might want to pass `mysql_query` a string like
```
INSERT INTO users (username, hash, cash) VALUES('$username', '$hash', 10000.00)
```
where
```
$username = mysql_real_escape_string($_POST["username"]);
```
and
```
$hash = crypt($_POST["password"]);
```
though we leave it to you to decide how much cash your code should give to new users. Note that, because `$username` and `$hash` are strings, they need to be quoted in SQL queries; a number like `10000.00` does not.
iii. Know that `mysql_query` will return `false` if your `INSERT` fails (as can happen if, say, `username` already exists).[20] Of course, if you cannot `INSERT`, you should certainly apologize to the user, as by calling `apologize`.
iv. If, though, your `INSERT` succeeds, know that you can find out which `id` was assigned to that user with a call to `mysql_insert_id` right after your call to `mysql_query`.[21]
v. If registration succeeds, you might as well log the new user in (as by "remembering" that `id` in `$_SESSION`), thereafter redirecting to `index.php`.

All done with the above? Ready to test? Head back to

```
http://192.168.56.50/~jharvard/pset7/register.php
```

and try to register a new username. If you reach `index.php`, odds are you done good! Confirm as much by returning to phpMyAdmin, clicking once more that tab labeled **Browse** for the table called `users`. May that you see your new user. If not, it's time to debug!

You may have noticed, incidentally, that `helpers.php` provides a function called `dump` that spits out to your browser the value(s) in any variable you pass it, using `~/public_html/pset7/includes/dump.php` as its template. For instance, if you'd like to dump the entire contents of `$_POST`, simply add

```
dump($_POST);
```

to your code temporarily wherever you'd like, much like you used to use `printf` whilst debugging C code. Note that `dump` is meant only for debugging, not apologies to users.

Be sure, incidentally, that any HTML generated by `register.php` and `register2.php` is valid by checking it with the W3C's validator. Take care to validate the HTML that your PHP code outputs to a browser, not the PHP code itself. In other words, ctrl- or right-click somewhere on

[20] See `http://www.php.net/mysql_query` for reference.
[21] See `http://www.php.net/mysql_insert_id` for reference.

your web page (in Firefox within the appliance), then select **Web Developer > Tools > Validate Local HTML**.

☐ Do bear in mind as you proceed further that you are welcome to play with and learn from the staff's implementation of C$50 Finance, available at the URL below.

```
https://www.cs50.net/finance/
```

In particular, you are welcome to register with as many (fake) usernames as you would like in order to play. And you are welcome to view our pages' HTML and CSS (by viewing our source using your browser) so that you might learn from or improve upon our own design. If you wish, feel free to adopt our HTML and CSS as your own.

But do not feel that you need copy our design. In fact, for this problem set, you may modify every one of the files we have given you to suit your own tastes as well as incorporate your own images and more. In fact, may that your version of C$50 Finance be nicer than ours!

☐ Now let's empower users to get quotes for individual stocks. Go ahead and create a new pair of files, these two named `quote.php` and `quote2.php`. It's up to you whether you want to create these from scratch or base them on your files for logins and registrations. Ultimately, though, `quote.php` should present users with a form (that gets submitted to `quote2.php`) that expects a stock's symbol in a text field. Upon receipt of that symbol, `quote2.php` must inform the user of the current price of the stock described by that symbol.

But how now? Well, recall that function called `lookup` in `helpers.php`. Why not invoke it with code like the below?

```
$s = lookup($_POST["symbol"]);
```

Assuming `$_POST["symbol"]` is non-`NULL` and contains a symbol for an actual stock, `lookup` will return an "object" of type `stock`. (Recall that `stock` was defined in `stock.php`.) If you think of `$s` as a pointer (a "reference" in PHP), you can access (or, better yet, print) individual fields in that object with code like the below.

```
print($s->price);
```

Incidentally, remember that your PHP code need not appear only at the top of `.php` files. In fact, you can intersperse PHP and HTML, as in the below, provided `$s` has already been assigned elsewhere (*e.g.*, atop your file) the return value of `lookup`.

```
<div style="text-align: center">
  A share of <? print($s->symbol); ?> currently costs $<? print($s->price); ?>.
</div>
```

Actually, if all you want to do is print some value(s), you can use this more concise syntax instead:

```
<div style="text-align: center">
  A share of <?= $s->name ?> currently costs $<?= $s->price ?>.
</div>
```

However, be sure to display prices to at least two decimal places but no more than four. You might find PHP's `number_format` function of assistance.

Of course, if the user submits an invalid `symbol` (for which `lookup` returns `NULL`), be sure to apologize to the user, explaining the problem!

Be sure, too, that any HTML generated by `quote.php` and `quote2.php` is valid, per the W3C's validator.

☐ And now it's time to do a bit of design. At present, your database has no way of keeping track of users' portfolios, only users themselves.[22] It doesn't really make sense to add additional fields to `users` itself in order to keep track of the stocks owned by users (using, say, one field per company owned). After all, how many different stocks might a user own? Better to maintain that data in a new table altogether so that we do not impose limits on users' portfolios or waste space with potentially unused fields.

Exactly what sort of information need we keep in this new table in order to "remember" users' portfolios? Well, we probably want a field for users' IDs (`id`) so that we can cross-reference holdings with entries in `users`. We probably want to keep track of stocks owned by way of their symbols since those symbols are likely shorter (and thus more efficiently stored) than stocks' actual names.[23] And we probably want to keep track of how many shares a user owns of a particular stock. In other words, a table with three fields (`id`, `symbol`, and `shares`) sounds pretty good, but you're welcome to proceed with a design of your own. Whatever your decision, head back to phpMyAdmin and create this new table, naming it however you see fit. To create a new table, click **Create table** in phpMyAdmin's top-left corner. (You may need to click your database's name (*i.e.*, `jharvard_pset7`) in the top-left corner first.) On the screen that appears, input a name for your table and then define (in any order) each of your fields. By default, phpMyAdmin will allow you to create two fields at once; to add a third, click **Go** in the page's bottom-right corner before clicking **Save**.

If you decide to go with three fields (namely `id`, `symbol`, and `shares`), realize that `id` should not be defined as a primary key in this table, else each user could own no more than one company's stock (since his or her `id` could not appear in more than one row). Realize, too, that you shouldn't let some `id` and some `symbol` to appear together in more than one row. Better to consolidate users' holdings by updating shares whenever some user sells or buys more shares of some stock he or she already owns. A neat way to impose this restriction while creating your table is to define a "joint primary key" by selecting an **Index** of `PRIMARY` for both `id` and `symbol`. That way,

[22] By "portfolio," we mean a collection of stocks (*i.e.*, shares of companies) that some user owns.

[23] Of course, you could also assign unique numeric IDs to stocks and remember those instead of their symbols. But then you'd have to maintain your own database of companies, built up over time based on data from, say, Yahoo. It's probably better (and it's certainly simpler), then, to keep track of stocks simply by way of their symbols.

`mysql_query` will return `false` if you try to insert more than one row for some pair of `id` and `symbol`. We leave it to you, though, to decide your fields' types.[24] When done defining your table, click **Save**!

☐ Before we let users buy and sell stocks themselves, let's give some shares to Skroob and friends at no charge. Click, in phpMyAdmin's left-hand frame, the link to `users` and remind yourself of your current users' IDs. Then click, in phpMyAdmin's left-hand frame, the link to your new table (for users' portfolios), followed by the tab labeled **Insert**. Via this interface, go ahead and "buy" some shares of some stocks on behalf of your users by manually inserting rows into this table. (You may want to return to Yahoo! Finance to look up some actual symbols.) No need to debit their `cash` in `users`; consider these shares freebies.

Once you've bought your users some shares, let's see what you did. Click the tab labeled **SQL** and run a query like the below, where `tbl` represents your new table's name.[25]

```
SELECT * FROM tbl WHERE id = 7
```

Assuming `7` is Skroob's user ID, that query should return all rows from `tbl` that represent the the president's holdings. If the only fields in table are, say, `id`, `symbol`, and `shares`, then know that the above is actually equivalent to the below.

```
SELECT id, symbol, shares FROM tbl WHERE id = 7
```

If, meanwhile, you'd like to retrieve only Skroob's shares of A5, you might like to try a query like the below.

```
SELECT shares FROM tbl WHERE id = 7 AND symbol = 'AFLB.OB'
```

If you happened to buy Skroob some shares of that company, the above should return one row with one column, the number of shares. If you did not get him in on that deal, the above will return an empty result set.

Incidentally, via this **SQL** tab, you could have inserted those "purchases" with `INSERT` statements. But phpMyAdmin's GUI saved you the trouble.

Alright, let's put this knowledge to use. It's time to let users peruse their portfolios! Overhaul `index.php`, in such a way that it reports each of the stock's in a user's portfolio, including number of shares and current value thereof, along with a user's current cash balance. You are welcome, though not required, to make use of the `stock` class's other data. Needless to say, `index.php` will need to invoke `lookup` much like `quote2.php` did, though perhaps multiple times. Know that a PHP script can certainly invoke `mysql_query` multiple times, even though, thus far, we've seen it used in each file no more than once. Similarly can you call `mysql_fetch_array` multiple times, particularly in loops.

[24] If you include `id` in this table, know that its type should match that in users. But don't specify `auto_increment` for that field in this new table, as you only want auto-incrementation when user IDs are created (by `register2.php`) for new users. And don't call your table `tbl`.
[25] Incidentally, because `2` is a number (just as `10000.00` was earlier), you need not enclose it in quotes like you do strings.

For instance, if your goal is simply to display, say, President Skroob's holdings, one per row in some HTML table, you can generate rows with code like the below.[26]

```
<?
    $result = mysql_query("SELECT symbol, shares FROM tbl WHERE id = 7");
    while ($row = mysql_fetch_array($result))
    {
        $s = lookup($row["symbol"]);
        print('<tr>');
        print('<td>' . $s->name . '</td>');
        print('<td>' . $row["shares"] . '</td>');
        print('</tr>');
    }
?>
```

Alternatively, you can avoid using the concatenation operator (`.`) via syntax like the below:

```
<?
    $result = mysql_query("SELECT symbol, shares FROM tbl WHERE id = 7");
    while ($row = mysql_fetch_array($result))
    {
        $s = lookup($row["symbol"]);
        print("<tr>");
        print("<td>{$s->name}</td>");
        print("<td>{$row["shares"]}</td>");
        print("</tr>");
    }
?>
```

Note that, in the above version, we've surrounded the lines of HTML with double quotes instead of single quotes so that the variables within (`$n->name` and `$row["shares"]`) are interpolated (*i.e.*, substituted with their values) by PHP's interpreter; variables between single quotes are not interpolated. And we've also surrounded those same variables with curly braces so that PHP realizes they're variables; variables with simpler syntax (*e.g.*, `$foo`) do not require the curly braces for interpolation.[27] Anyhow, though commonly done, generating HTML via calls to print isn't terribly elegant. An alternative approach, though still a bit inelegant, is code more like the below.

```
<? $result = mysql_query("SELECT symbol, shares FROM tbl WHERE id = 7"); ?>

<? while ($row = mysql_fetch_array($result)): ?>

    <? $s = lookup($row["symbol"]); ?>

    <tr>
        <td><?= $s->name ?></td>
        <td><?= $row["shares"] ?></td>
    </tr>

<? endwhile ?>
```

[26] Note that developers tend to use single quotes around HTML, lest the HTML itself contain double quotes, as around attributes' values.

[27] It's fine to use double quotes inside those curly braces, even though we've also used double quotes to surround the entire argument to `print`.

As for what HTML to generate, look, as before, to

```
https://www.cs50.net/finance/
```

for inspiration or hints. But do not feel obliged to mimic our design. Make this website your own! Although any HTML and PHP code that you yourself write should be pretty-printed (*i.e.*, nicely indented), it's okay if lines exceed 80 characters in length. HTML that you generate dynamically (as via calls to `print`), though, does not need to be pretty-printed.

As with `quote2.php`, be sure to display portfolio's values and cash balances to at least two decimal places but no more than four.

Incidentally, though we keep using President Skroob in examples, your code should work for whichever user is logged in.

As always, be sure that the HTML generated by `index.php` is valid.

☐ And now it is time to implement the ability to sell in, say, `sell.php` and `sell2.php`. We leave the design of the former, in particular, to you. But know, for the latter, that you can delete rows from your table (on behalf of, say, Skroob) with SQL like the below.

```
DELETE FROM tbl WHERE id = 7 AND symbol = 'AFLB.OB'
```

We leave it to you to infer exactly what that statement should do. Of course, you could try the above out via phpMyAdmin's **SQL** tab. Now what about the user's cash balance? Odds are, your user is going to want the proceeds of all sales. So selling a stock involves updating not only your table for users' portfolios but users as well. We leave it to you to determine how to compute how much cash a user is owed upon sale of some stock. But once you know that amount (say, $500), SQL like the below should take care of the deposit (for, say, Skroob).[28]

```
UPDATE users SET cash = cash + 500 WHERE id = 7
```

It's fine, for simplicity, to require that users sell all shares of some stock or none, rather than only a few. Needless to say, try out your code by logging in as some user and selling some stuff. You can always "buy" it back manually with phpMyAdmin.

As always, be sure that your HTML is valid!

[28] Of course, if the database or web server happens to die between this `DELETE` and `UPDATE`, Skroob might lose out on all of that cash. You need not worry about such cases! It's also possible, because of multithreading and, thus, race conditions, that a clever Skroob could trick your site into paying out more than once. You need not worry about such cases either! Though, if you're so very inclined, you can employ MyISAM locks or InnoDB tables with SQL transactions. See `http://dev.mysql.com/doc/refman/5.5/en/innodb.html` for reference.

☐ Now it's time to support actual buys. Implement the ability to buy, in, say, `buy.php` and `buy2.php`.[29] The interface with which you provide a user is entirely up to you, though, as before, feel free to look to

```
https://www.cs50.net/finance/
```

for inspiration or hints. Of course, you'll need to ensure that a user cannot spend more cash than he or she has on hand. And you'll want to make sure that users can only buy whole shares of stocks, not fractions thereof. For this latter requirement, know that a call like

```
preg_match("/^\d+$/", $_POST["shares"])
```

will return `true` if and only if `$_POST["shares"]` contains a non-negative integer, thanks to its use of a regular expression. See `http://www.php.net/preg_match` for details. Take care to apologize to the user if you must reject their input for any reason. In other words, be sure to perform rigorous error-checking. (We leave to you to determine what needs to be checked!)

When it comes time to store stocks' symbols in your database table, take care to store them in uppercase (as is convention), no matter how they were inputted by users, so that you don't accidentally treat, say, `aflb.ob` and `AFLB.OB` as different stocks. Don't force users, though, to input symbols in uppercase.

Incidentally, if you implemented your table for users' portfolios as we did ours (with that joint primary key), know that SQL like the below (which, unfortunately, wraps onto two lines) will insert a new row into table unless the specified pair of `id` and `symbol` already exists in some row, in which case that row's number of shares will simply be increased (say, by `10`).

```
INSERT INTO table (id, symbol, shares) VALUES(7, 'AFLB.OB', 10)
ON DUPLICATE KEY UPDATE shares = shares + VALUES(shares)
```

As always, be sure to bang on your code. And be sure that your HTML is valid!

☐ And now for your big finale. Your users can now buy and sell stocks and even check their portfolio's value. But they have no way of viewing their history of transactions.

Enhance your implementations for buying and selling in such a way that you start logging transactions, recording for each:

- ☐ Whether a stock was bought or sold.
- ☐ The symbol bought or sold.
- ☐ The number of shares bought or sold.
- ☐ The price of a share at the time of transaction.
- ☐ The date and time of the transaction.

[29] As before, you need not worry about interruptions of service or race conditions.

Then, by way of a file called `history.php`, enable users to peruse their own history of transactions, formatted as you see fit. Provide a hyperlink to that file somewhere in `index.php`. Be sure that your HTML is valid!

☐ Phew. Glance back at `index.php` now and, if not there already, make that it somehow links to, at least, `buy.php`, `history.php`, `logout.php`, `quote.php`, and `sell.php` (or their equivalents) so that each is only one click away from a user's portfolio!

☐ And now the icing on the cake. Only one feature to go, but you get to choose. Implement at least one (1) of the features below. You may interpret each of the below as you see fit; we leave all design decisions to you. Just take care to make clear to your TF (as via an appropriately named hyperlink in `index.php`) which feature you tackled. And be sure that your HTML is valid.

- ☐ Empower users (who're already logged in) to change their passwords.
- ☐ Empower users who've forgotten their password to reset it (as by having them register with an email address so that you can email them a link via which to do so).
- ☐ Email users "receipts" anytime they buy or sell stocks.
- ☐ Empower users to deposit additional funds.

For tips on how to send email programmatically, see:

`https://manual.cs50.net/Sending_Mail`

CS50 in the Cloud.

☐ Once you think you're all done, it's time to invite one or more friends to try out your site. Of course, your website isn't technically on the Web just yet, since it lives on your own computer within the appliance. But recall that you have, thanks to Problem Set 1, a CS50 Cloud account. Not only does that account allow you to log into the course's website and run `submit50`, it also provides you with access to `cloud.cs50.net` on which you can host your implementation of C$50 Finance!

If you would like to upload your work to `cloud.cs50.net` so that friends and family can try it out via their own computers, head to

`https://manual.cs50.net/Cloud`

for instructions. If you'd rather not, that's okay too, but at least invite someone over to try out your site on the appliance itself.

Encourage them to try breaking it, like a good adversary would. Under no circumstances should they (or we) be able to crash your code (*i.e.*, trigger some warning or error from PHP's own interpreter). You'd best catch and/or apologize for any error that a user's input, malicious or otherwise, might induce! And you'd best scrub all user input for safety's sake, as with `mysql_real_escape_string`!

Another Big Board.

☐ Just for fun, why don't we give you some cash out of our own pocket. How does $10K for each of you sound? If you would like, surf on over to

`https://www.cs50.net/finance/`

and you'll find that $10K awaits you if you follow the link to **play the BIG BOARD**. We shall see, come the course's final lecture on Mon 11/21, who exits this course with the most money in hand.[30] Last year's winner pocketed $34,261,456,955.42. (He's now a CA.)

Sanity Checks.

Before you consider this problem set done, best to ask yourself these questions and then go back and improve your code as needed! Do not consider the below an exhaustive list of expectations, though, just some helpful reminders. The checkboxes that have come before these represent the exhaustive list! To be clear, consider the questions below rhetorical. No need to answer them in writing for us, since all of your answers should be "yes!"

☐ Is your homepage's HTML valid according to `validator.w3.org`?
☐ Is the HTML generated by all of your PHP files valid according to `validator.w3.org`?
☐ Do your pages detect and handle invalid inputs properly?
☐ Are you recording users' histories of transactions properly?
☐ Did you add one (1) additional feature of your own?
☐ Does every page (other than your login, logout, and registration pages) require authentication?
☐ Did you choose appropriate data types for your database tables' fields?
☐ Are you displaying any dollar amounts to at least two decimal places but no more than four?
☐ Are you storing stocks' symbols in your table(s) in uppercase?
☐ Does your site behave the same in at least two major browsers?

As always, if you can't answer "yes" to one or more of the above because you're having some trouble, do turn to `help.cs50.net`!

[30] Money may not be legal tender.

How to Submit.

In order to submit this problem set, you must first execute a command in the appliance and then submit a (brief) form online.

☐ Recall that you obtained a CS50 Cloud account (*i.e.*, username and password) for Problem Set 1. If you don't remember your username and/or password, head to `https://cloud.cs50.net/` look up the former and/or change the latter. You'll be prompted to log in with your HUID (or XID) and PIN.

☐ Just in case we updated `submit50` since you started this problem set, open a terminal window and execute the below, inputting **crimson** if prompted for John Harvard's password.

```
sudo yum -y update
```

If there was something to update, you should see **Complete!** after a few seconds or minutes. If there was nothing to update, you should instead see **No packages marked for Update**. If you see any errors, try the command once more, try to restart the appliance and then try once more, then head to `https://manual.cs50.net/FAQs` followed by `http://help.cs50.net/` as needed for help!

☐ To actually submit, first open a terminal window and execute:

```
cd ~/public_html
```

Then execute:

```
ls
```

At a minimum, you should see `index.html` and `pset7`. If not, odds are you skipped some more steps earlier! If everything is as it should be, you are ready to submit your source code and database to us. First execute

```
cd ~/public_html/pset7
mysqldump -u jharvard -p jharvard_pset7 > jharvard_pset7.sql
```

in order to "dump" your MySQL database to a file called `jharvard_pset7.sql` (so that we can re-create it on our end). Input a password of **crimson** if prompted. Then execute:

```
submit50 ~/public_html
```

When prompted for **Course**, input **cs50**; when prompted for **Repository**, input **pset7**. When prompted for a username and password, input your CS50 Cloud username and password. For security, you won't see your password as you type it. That command will essentially upload your entire `~/public_html` directory to CS50's repository, where your TF will be able to access it. The command will inform you whether your submission was successful or not. If provided with the URL of a PDF of your code (which further confirms its submission), right-click (or ctrl-click) the link, then choose **Open Link** from the menu that appears to open the PDF in Document Viewer.

You may re-submit as many times as you'd like; we'll grade your most recent submission. But take care not to submit after the problem set's deadline, lest you spend a late day unnecessarily or risk rejection entirely.

If you run into any trouble at all, let us know via `help.cs50.net` and we'll try to assist! Just take care to seek help well before the problem set's deadline, as we can't always reply within minutes!

If you decided to upload your work to `cloud.cs50.net` but ended up making some changes there to your code, be sure that you made those changes on your appliance as well so that you ultimately submit the right version of your work.

☐ Anytime after lecture on Mon 10/31 but before this problem set's deadline, head to the URL below where a short form awaits:

`https://www.cs50.net/psets/7/`

Once you have submitted that form (as well as your source code and database), you are done!

This was Problem Set 7.