Problem Set 7: C$50 Finance

due by 7:00pm on Fri 11/6

Goals.

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

Recommended Reading.

- `http://www.w3schools.com/xhtml/`
- `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 XHTML, 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 XHTML, CSS, 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 (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.

□ 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 prepare for some fun. First, make your home directory "world-executable" by executing the command below.

```
chmod 711 ~
```

Now create a directory called `public_html` in your home directory with `mkdir`. And then make it world-executable as well by executing the command below.

```
chmod 711 ~/public_html/
```

Then execute the command below to copy this problem set's framework into `~/public_html/`.

```
cp -r ~cs50/pub/src/psets/pset7/ ~/public_html/
```

And now make your copy world-executable as well:

```
chmod 711 ~/public_html/pset7/
```

Next, navigate your way to `~/public_html/pset7/` 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:

```
chmod 711 css/ images/
```

And some files within world-readable:

```
chmod 644 css/* images/*
```

Note 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/`.

Now execute the command below.

```
cd includes/
```

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

```
apology.php  common.php  constants.php  helpers.php  stock.php
```

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. Head to the URL below.

```
http://www.cs50.net/me/
```

You may be prompted to authenticate. Ultimately, you'll reach a page that provides you with personalized values for each of those variables. Fill in the blanks in `constants.php` by copying and pasting these values, taking care to keep the variables' values enclosed in quotes. Then save your changes and quit Nano. You just configured your copy of C$50 Finance to use your very own database! We'll explain the rest of that file in a bit.

Now surf on over to the URL below, where `username` is your own username.[1]

```
http://cloud.cs50.net/~username/pset7/
```

You should find yourself redirected to the login page for your own copy of C$50 Finance! If anything seems broken, try all those `chmod` steps again. Don't try to register or log in just yet!

[1] Note that this URL is equivalent to `http://www.cs50.net/~username/pset7/index.php`.

Home, sweet home page.

□ It's time to make yourself a home page! Navigate your way to ~/public_html/. Create a file called index.html in that directory using Nano and fill that file with valid XHTML.[2] 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 XHTML 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 and quit Nano. Before doing anything else, though, execute the command below.

```
ls -l
```

The output you see should resemble the below.

```
-rw------- 1 username students   50 Oct 30 19:00 index.html
drwx--x--x 5 username students 4096 Oct 30 19:00 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! It turns out 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.*, you plus anyone with access to cloud.cs50.net or to, in the case of files and directories inside ~/public_html/, the World Wide Web).

By default, files that you create with Nano are readable and writable only by you, their owner. In fact, take a look at index.html's current set of permissions: -rw-------. This sequence confirms that index.html is indeed readable and writable (but not executable) by you, as the second symbol through fourth represent owner permissions. That the rest of those symbols are hyphens means that neither your group nor the world have any permissions at all, as the fifth symbol through seventh represent group permissions, and the eighth symbol through tenth represent world permissions.

Well that's no good! If index.html belongs on the Web, you'd best give everyone permission to read it! Go ahead and execute the below.

```
chmod 644 index.html
```

Now go ahead and execute again the below.

```
ls -l
```

[2] Specifically, XHTML 1.0 Transitional or XHTML 1.0 Strict.

[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 symbolic link (l).

The output you see should now resemble the below.

```
-rw-r--r-- 1 username students   50 Oct 30 19:00 index.html
drwx--x--x 5 username students 4096 Oct 30 19:00 pset7/
```

The difference, of course, is that both your group and the world can now read (but not write or execute) your home page. Confirm as much by surfing over to the URL below, where username is, again, your username.

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

Wow, that page is ugly. (Okay, maybe it's not.) But the point is that you are now on the Web! Make any improvements you'd like to `index.html`. You may certainly, but need not, employ CSS. Anytime you save changes with Nano, simply reload your page in your browser. You should not need to run `chmod` again for this particular file.

Ultimately, just be sure that your page is valid (or "tentatively valid") XHTML. You will likely find the W3C's Markup Validation Service of assistance, the URL of which is below.

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

Oh, by the way, it's no fun to be on the Web if nobody knows it. Go tweet or update your Facebook profile or at least email someone the URL of your new home![5]

□ Okay, time for a heads-up. Anytime you create a new file in `~/public_html/` or some subdirectory therein (as with Nano or SFTP) for this problem set, be sure to set its permissions with `chmod`, just as we've done with the files you already have. Specifically, moving forward, you'll want to execute

```
chmod 644 foo
```

for every non-PHP file, `foo`, within `~/public_html/`,

```
chmod 600 bar
```

for every PHP file, `bar`, within `~/public_html/` and

```
chmod 711 baz
```

for every directory, `baz`, within `~/public_html/`.

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

[5] You're not going to, are you.

everyone to be able to read (*i.e.*, interpret) your PHP files? Nope! For security reasons, PHP-based webpages are interpreted "as you" (*i.e.*, under your username) on `cloud.cs50.net`, no matter who pulls them up in a browser.[6]

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!

□ Do not forget that this course has an anonymized bulletin board! Particularly for this problem set, turn to it for counsel and hints.

□ 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![7]

□ Allow me to share something from my spam folder with you.

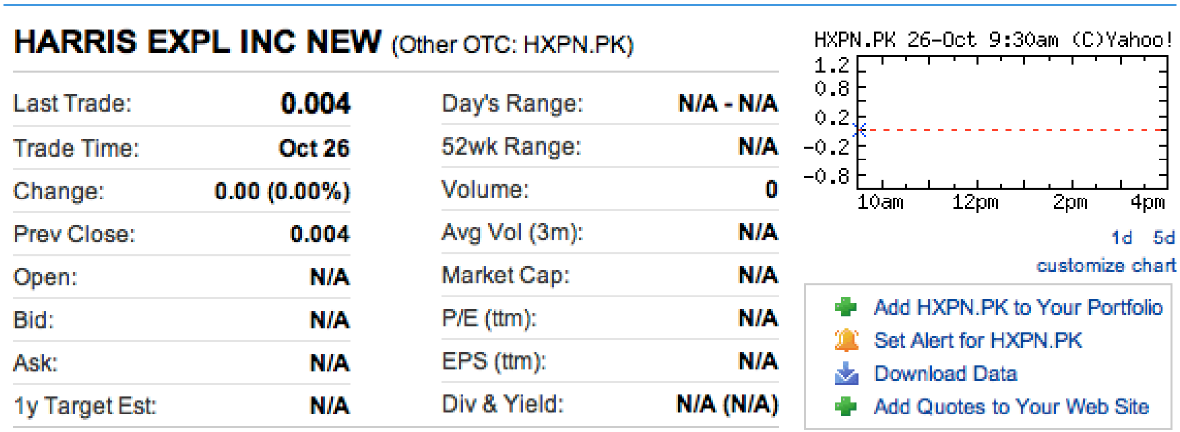
```
HXPN IS MAKING GREAT PROGRESS! GET ON THIS TRAIN NOW!

Company: Harris Exploration Inc
Symbol: HXPN.PK
Price: 0.50
5-day Target: $3
Rating: Strong Buy

* Harris Exploration Inc. Announces Advanced Zone Discovery
HXPN COULD INFLATE YOUR PORTFOLIO GREATLY! THIS ONE HAS AMAZING
POTENTIAL! ADD IT TO YOUR RADAR...!
```

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

☐ Talk about a hot stock tip, 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 Harris Exploration Inc., `HXPN.PK`, 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.[8]



HARRIS EXPL INC NEW (Other OTC: HXPN.PK)

Last Trade:	**0.004**	Day's Range:	**N/A - N/A**
Trade Time:	**Oct 26**	52wk Range:	**N/A**
Change:	**0.00 (0.00%)**	Volume:	**0**
Prev Close:	**0.004**	Avg Vol (3m):	**N/A**
Open:	**N/A**	Market Cap:	**N/A**
Bid:	**N/A**	P/E (ttm):	**N/A**
Ask:	**N/A**	EPS (ttm):	**N/A**
1y Target Est:	**N/A**	Div & Yield:	**N/A (N/A)**

Add HXPN.PK to Your Portfolio
Set Alert for HXPN.PK
Download Data
Add Quotes to Your Web Site

Needless to say, I lost a lot of money because of that spam. 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. Notice, too, that Yahoo lets you download this data. 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.*, Notepad or TextEdit), 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=HXPN.PK&f=sl1d1t1c1ohgv&e=.csv
```

Notice how Harris'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 or XHTML for data of interest (*e.g.*, air fares, stock prices, *etc.*). Writing a screen scraper for a site tends to

[8] Never mind all those **N/A**s. The penny stock isn't doing so well.

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.
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 Nano. You'll see XHTML 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 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.

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");
```

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 Nano. 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.

[9] The calls to preg_match and session_set_cookie_params tell PHP to associate cookies with http://cloud.cs50.net/~username/pset7/ instead of just http://cloud.cs50.net/.

[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!

Alright, now open constants.php with Nano again. In this file have we defined some global constants, three of whose values you yourself provided earlier. 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. Notice the URL for Yahoo! Finance, which 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 Nano, and you'll see something that resembles a C struct. Indeed, this code defines a structure (a "class" in PHP) for stocks. 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 Nano. 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.

Finally, take a peek at apology.php. 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, only three files remain to examine, each of which ends in .php. Navigate back to ~/public_html/pset7/, and open up login.php with Nano. 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 XHTML. But it's that XHTML that implements that login page that you saw. Note that it lays out a form using an "invisible" table. In fact, go ahead and change

```
border="0"
```

to

```
border="1"
```

in the start tag for that table. Save the file, then revisit the URL below, reloading if necessary.

```
http://cloud.cs50.net/~username/pset7/login.php
```

The table should now be visible. Notice next the far more important 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 with Nano.

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 input 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.

```
$sql = "SELECT uid FROM users WHERE username='$username' AND password='$password'");
```

To be clear, suppose that President Skroob tries to log into C$50 Finance with his username and password.[11] That line of code will assign to $sql the value below.

```
SELECT uid FROM users WHERE username='skroob' AND password='12345'
```

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 and password are correct, though, will the database return an actual row. And, so, if mysql_num_rows returns 1, Skroob has successfully authenticated. Our code "remembers" as much by storing his numeric user ID (uid) in $_SESSION; it then redirects him to index.php, where his portfolio (once you implement it) awaits![12] If, however, his password (or perhaps username) was invalid, he is instead informed accordingly.

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("/(:?log(:?in|out)|register)\d*\.php$/", $_SERVER["PHP_SELF"]))
{
    if (!isset($_SESSION["uid"]))
        redirect("login.php");
}
```

Though a bit scary (I prefer "elegant"), that code simply asks whether $_SESSION["uid"] 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["uid"] 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.[13]

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 with Nano. Note that it calls logout, a function defined

[11] http://en.wikipedia.org/wiki/Spaceballs#Spaceballs

[12] 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.

[13] http://en.wikipedia.org/wiki/Regular_expression

in `common.php`. (We call that same function atop `login.php`.) Alternatively, Skroob can simply close his own browser, as `$_SESSION` is lost in that case as well.

□ Phew, that was a lot. Snack time.

□ Alright, let's talk about that database we keep mentioning. So that you have someplace to store users' portfolios, we've taken the liberty of creating a MySQL database just for you for this problem set. We've even pre-populated it with one table! If you don't remember your database's username and password, glance at `constants.php` or return to the URL below.

```
http://www.cs50.net/me/
```

Then head to

```
http://cs50.net/phpMyAdmin/
```

and provide that same username and password when prompted by phpMyAdmin, a Web-based tool (that happens to be written in PHP) with which you can manage MySQL databases. You'll ultimately find yourself at phpMyAdmin's main page. Feel free to click **Change password** in order to change your password; just take care to update the value of `DB_PASS` in `constants.php` accordingly.

You'll notice in phpMyAdmin's top-left corner that we've actually created three databases for you, only one of which is meant for this problem set.[14] Click the link to `username_pset7` (next to which there's a parenthetical `1`, which confirms that a table already awaits you). On the page that appears, you'll see, again at top-left, that the table's called `users`. Click the name of that table to see the contents of that table. Ah, some familiar folks. In fact, there's President Skroob's username and password!

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

```
SELECT uid FROM users WHERE username='skroob' AND password='12345'
```

As phpMyAdmin makes clear, this table called users contains three fields: `uid` (the type of which is an `UNSIGNED INT`) 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 `uid`, meanwhile, is that it is auto-incrementing: when inserting a new user into the table, you needn't specify a value for `uid`; the user will be assigned the next available `INT`. Click **Details...** and notice, finally, the box labeled **Indexes**. It appears that this table's `PRIMARY` key is `uid`, the implication of which is that (as expected) no two users can share the same user ID.[15] Of course, `username` should also be unique across users, and so we have also defined it as just that. To be sure, we could have defined

[14] Not only do you have a second database for Problem Set 8, we also created a third, just in case you want one for your final project.

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

username as this table's primary key. But, for efficiency's sake, the more conventional approach is to use an INT like uid. 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.

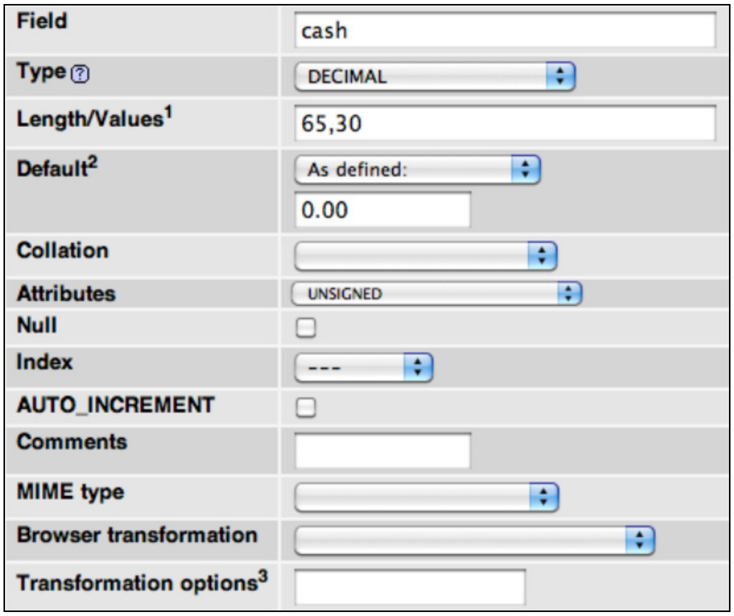
Make sense? Okay, head on back to

```
http://cloud.cs50.net/~username/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** ... **field(s)** ... **After** another. Click the radio button immediately to the left of **After**, select password from the drop-down menu, then click **Go**.

Via the form that appears, add a field of type DECIMAL called cash, using the settings depicted below.



Field	cash
Type	DECIMAL
Length/Values[1]	65,30
Default[2]	As defined: 0.00
Collation	
Attributes	UNSIGNED
Null	□
Index	---
AUTO_INCREMENT	□
Comments	
MIME type	
Browser transformation	
Transformation options[3]	

If you pull up the documentation for MySQL at

```
http://dev.mysql.com/doc/refman/5.0/en/numeric-types.html
```

you'll see that the DECIMAL data type is "used to store exact numeric data values." A length of 65,30 for a DECIMAL means that values for cash can have 65 significant digits to the left of their decimal point and 30 digits to the right. Don't even think about stealing fractions of pennies from users (a la Office Space, a la Superman 3) with that much precision.

Tempted though you may be to start users off with $999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999.99.000000000000000000000000000000 in cash, do go with a default of $0.00 (or, more precisely, $0.000000000000000000000000000000).[16] Click **Save** to make all this so. Then return to the tab labeled **Browse** and give everyone $10,000.00 manually. The easiest way is to click **Check All**, then click the icon on that same line that looks like pencil. On the page that appears, change 0.000000000000000000000000000000 to 10000.00 for each of your users, then click **Go**. Won't they be happy![17]

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

Return your attention to `cloud.cs50.net` 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.[18] Then use `chmod` to ensure all permissions are set properly. Open up `register.php` with Nano 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 XHTML 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://cloud.cs50.net/~username/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 using Nano, saving your changes, thereafter reloading the page.

Once the page looks okay, head back to Nano 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, password, cash) VALUES('$username', '$password', 10000.00)
```
though we leave it to you to decide how much cash your code should give to new users.

[16] Don't input the dollar sign into phpMyAdmin, though.
[17] Of course, they'd be happier with all 65 significant digits.
[18] 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.

iii. Know that `mysql_query` will return `FALSE` if your `INSERT` fails (as can happen if, say, `username` already exists).[19] Of course, if you cannot `INSERT`, you should certainly apologize.

iv. If, though, your `INSERT` succeeds, know that you can find out which `uid` was assigned to that user with a call to `mysql_insert_id` right after your call to `mysql_query`.[20]

v. If registration succeeds, you might as well log the new user in (as by "remembering" that `uid` in `$_SESSION`), thereafter redirecting to `index.php`.

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

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

Recall, incidentally, that `helpers.php` provides a function called `dump` that spits out to your browser the value(s) in any variable you pass it. For instance, if you'd like to dump the entire contents of `$_POST`, simply include

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

temporarily wherever you'd like. Note that this function is meant for debugging, not apologies to users.

□ 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.

```
http://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' XHTML 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 XHTML 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!

Oh, you're even welcome to rename your baby.

```
http://googlefont.com/
```

□ Do not forget that the course has a bulletin board!

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

☐ 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 XHTML, as in the below, provided $s has already been assigned elsewhere (*e.g.*, atop your file) the return value of lookup.

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

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

☐ 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.[21] 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 (uid) so that we can cross-reference holdings with entries in users. We probably want to keep track of stocks owned by way of their

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

symbols since those symbols are likely shorter (and thus more efficiently stored) than stocks' actual names.[22] 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 (`uid`, `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 your database's name (*i.e.*, `username_pset7`) in phpMyAdmin's top-left corner. Then, in the page's right-hand frame, specify the new table's **Name** and **Number of fields**, then click **Go**. On the screen that appears, define (in any order) each of your fields.

If you decide to go with three fields (namely `uid`, `symbol`, and `shares`), realize that `uid` 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 `uid` could not appear in more than one row). Realize, too, that you shouldn't let some `uid` 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 `uid` and `symbol`. That way, `mysql_query` will return `FALSE` if you try to insert more than one row for some pair of `uid` and `symbol`. We leave it to you, though, to decide your fields' types.[23] 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.[24]

```
SELECT * FROM tbl WHERE uid=2
```

Assuming `2` 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, `uid`, `symbol`, and `shares`, then know that the above is actually equivalent to the below.

```
SELECT uid,symbol,shares FROM tbl WHERE uid=2
```

[22] 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.

[23] If you include `uid` 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`.

[24] Incidentally, because `2` is a number (just as 10000.00 was earlier), you need not enclose it in quotes like you do strings.

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

```
SELECT shares FROM tbl WHERE uid=2 AND symbol='HXPN.PK'
```

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 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.

For instance, if your goal is simply to display, say, President Skroob's holdings, one per row in some XHTML table, you can generate rows with code like the below.[25]

```
<?
    $result = mysql_query("SELECT symbol,shares FROM tbl WHERE uid=2");
    while ($row = mysql_fetch_array($result))
    {
        $s = lookup($row["symbol"]);
        print('<tr>');
        print('<td>' . $s->name . '</td>');
        print('<td>' . $row["shares"] . '</td>');
        print('</tr>');
    }
?>
```

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

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

```
<?
    $result = mysql_query("SELECT symbol,shares FROM tbl WHERE uid=2");
    while ($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 XHTML 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.

Anyhow, though commonly done, generating XHTML 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 uid=2"); ?>

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

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

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

<? endwhile ?>
```

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

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

for inspiration or hints. But do not feel obliged to mimic our design. Make this website your own! Although any XHTML 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. XHTML that you generate dynamically (as via calls to `print`), though, does not need to be pretty-printed.

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

□ 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 uid=2 AND symbol='HXPN.PK'
```

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).[26]

```
UPDATE users SET cash=cash+500 WHERE uid=2
```

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.

□ Now it's time to support actual buys. Implement the ability to buy, in, say, `buy.php` and `buy2.php`.[27] The interface with which you provide a user is entirely up to you, though, as before, feel free to look to

```
http://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!)

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

[26] Of course, if the database or webserver 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/5.0/en/innodb.html` for reference.

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

a new row into table unless the specified pair of uid 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 (uid,symbol,shares) VALUES(2,'HXPN.PK',10)
ON DUPLICATE KEY UPDATE shares=shares+VALUES(shares)
```

As always, be sure to bang on your code.

☐ 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.

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.

☐ 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.

☐ Empower users to change their passwords.
☐ Empower users who've forgotten their password to receive reminders via email.
☐ Email users "receipts" anytime they buy or sell stocks.
☐ Empower users to deposit additional funds.

☐ Once you think you're all done, it's time to invite one or more friends to try out your site. 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!

□ 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

```
http://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 11/23, who exits this course with the most money in hand.[28]

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

Submitting Your Work.

□ Ensure that your home page is in `~/public_html/` and your implementation of C$50 Finance is in `~/public_html/pset7/`.[29] 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_pset7 > ~/public_html/pset7/username_pset7.sql
```

Now submit your work by executing the command below.

```
cs50submit pset7 ~/public_html/
```

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.

[28] Money may or may not be legal tender.

[29] If you have files in `~/public_html/` other than those required for this problem set and you do not wish `cs50submit` to copy those files (particularly if private) into our account, simply remove them from `~/public_html/` temporarily before submitting your work.