Computer Science 50: Introduction to Computer Science I
Harvard College
Fall 2008

Problem Set 7: C$50 Finance

due by 7:00 P.M. on Friday, 21 November 2008

Goals.

- Get you on teh interwebs.
- Introduce you to XHTML, CSS, PHP, and SQL.
- Teach you how to teach yourself other languages.
- Play a different BIG BOARD.

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, magazine, 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. 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.

☑ Do not read this sentence![1]

☐ For this problem set, consider downloading and installing **Firefox**, **Firebug**, and **Live HTTP Headers** (in that order), each of which is available on the course's website under **Software**. Once installed, Firebug and Live HTTP Headers 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 both are valuable.[2]

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

- ☐ Google Chrome 0.x
- ☐ Firefox 3.x
- ☐ Internet Explorer 7.x
- ☐ Opera 9.x
- ☐ Safari 3.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. 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 $HOME
```

Now create a directory called `public_html` in your home directory. And then make it world-executable as well by executing the command below.[3]

```
chmod 711 $HOME/public_html/
```

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

```
cp -r ~cs50/pub/distributions/pset7/ ~/public_html/
```

And now make your copy world-executable as well:

```
chmod 711 $HOME/public_html/pset7/
```

[1] Pwned.
[2] Speaking of tools, if you would like to be adventurous and develop this whole project on your own Mac or PC, you might also like XAMPP, also available on the course's website under **Software**. Just realize that, come submission time, your code and database must ultimately work and live on `cloud.cs50.net`.
[3] Recall that `$HOME` is equivalent to a tilde.

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://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 single 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.[4]

```
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, re-try `chmod`! Don't try to register or log in just yet!

[4] Note that this URL is equivalent to `http://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! 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 Nov 14 19:02 index.html
drwx--x--x 5 username students 4096 Nov 14 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! 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.[5,6] 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
```

[5] 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`.
[6] 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 Nov 14 19:02 index.html
drwx--x--x 5 username students 4096 Nov 14 19:01 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 FAS 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 update your Facebook profile or at least email someone the URL of your new home!

☐ Okay, a head's 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 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 `cs50.net`, no matter who pulls them up in a browser.[7]

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.

Show me the money!

☐ 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![8]

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

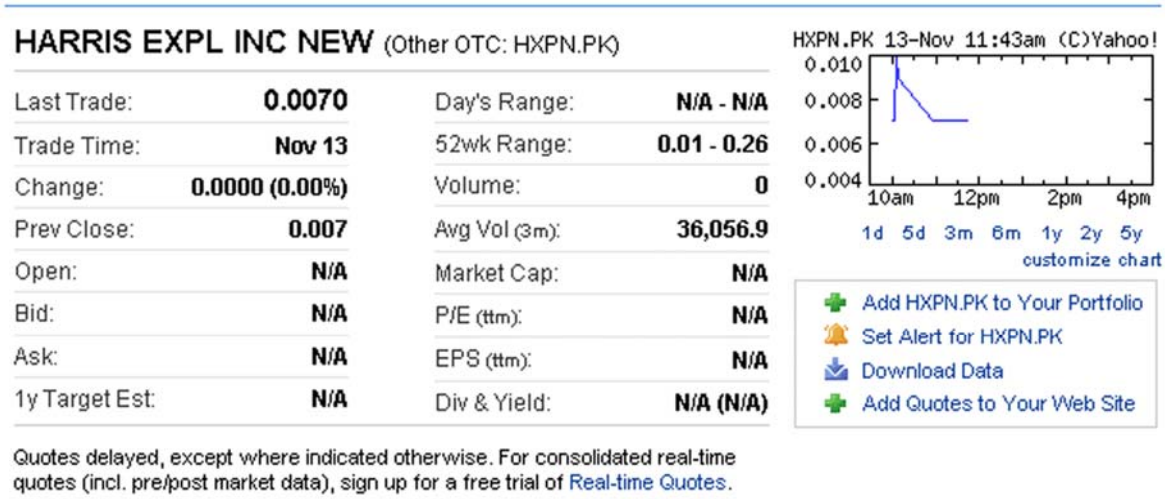
```
HXPN IS MAKING GREAT PROGRESS! GET ON THIS TRAIN NOW!

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...!
```

[7] For the curious, we're using suPHP (`http://www.suphp.org/`) with Apache (`http://httpd.apache.org/`).
[8] Actually, quotes will be slightly delayed since we won't be paying for, say, Bloomberg.

☐ 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.



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

Last Trade:	**0.0070**	Day's Range:	**N/A - N/A**
Trade Time:	**Nov 13**	52wk Range:	**0.01 - 0.26**
Change:	**0.0000 (0.00%)**	Volume:	**0**
Prev Close:	**0.007**	Avg Vol (3m):	**36,056.9**
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)**

Quotes delayed, except where indicated otherwise. For consolidated real-time quotes (incl. pre/post market data), sign up for a free trial of Real-time Quotes.

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.

Here, if curious, are the fields we think can appear in that otherwise cryptic string, along descriptions thereof.[9]

```
a       Ask
a2      Average Daily Volume
a5      Ask Size
b       Bid
b4      Book Value
b6      Bid Size
c       Change & Percent
c1      Change
c3      Commission
c5      Pre Market Change and Percent
c8      After Hours Change and Percent
d       Dividend/Share
d1      Last Trade Date
e       Earnings/Share
e7      EPS Est. Current Yr
e8      EPS Est. Next Year
e9      EPS Est. Next Quarter
f6      Float Shares
g       Day's Low
g1      Holdings Gain & Percent
g3      Annualized Gain
g4      Holdings Gain
h       Day's High
i       More Info
j1      Market Capitalization
j4      EBITDA
j5      Change from 52-Week Low
j6      Percent Change from 52-Week Low
k3      Last Trade Size
k4      Change From 52-wk High
k5      Percent Chg From 52-wk High
l       Last Trade (With Time)
l1      Last Trade (Price Only)
l2      High Limit
l3      Low Limit
l8      Last Trade with Time (Pre/After Market)
m       Day's Range
m3      50-day Moving Average
m4      200-day Moving Average
m5      Change From 200-day Moving Average
m6      Percent Change From 200-day Moving Average
m7      Change From 50-day Moving Average
m8      Percent Change From 50-day Moving Average
n       Name
n4      Notes
o       Open
p       Previous Close
p1      Price Paid
p5      Price/Sales
```

[9] Thanks to Chris Thorpe of SEAS for this list. Do let me know if you find this list incomplete or inaccurate.

```
p6      Price/Book
q       Ex-Dividend Date
r       P/E Ratio
r1      Dividend Pay Date
r5      PEG Ratio
r6      Price/EPS Est. Current Year
r7      Price/EPS Est. Next Year
s       Symbol
sl      Symbol
s1      Shares Owned
s7      Short Ratio
t1      Last Trade Time
t6      Trade Links
t7      Ticker Trend
t8      1-Year Target Price
v       Volume
v1      Holdings Value
w       52-Week Range
w1      Day's Value Change
y       Dividend Yield
```

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 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 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 next few lines require yet three other files; we'll return to those shortly. 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.[10,11] 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 Nano. 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 URLs for MSN and Yahoo, the latter of which should look awfully familiar. 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 many 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.

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

[11] 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 `$_SESSION`, an associative array in which you can store any data to which you'd like to have access for the duration of some user's visit. The moment, say, Lord Dark Helmet ends his "session" (*i.e.*, visit) by closing his browser, the contents of `$_SESSION` are lost for Lord Dark Helmet specifically because the next time he visits, he'll be assigned a new cookie!

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, if you'd like, 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 the case, then, that `login2.php` handles 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 = sprintf("SELECT uid FROM users WHERE username='%s' AND password='%s'",
               $username, $password);
```

To be clear, suppose that Lord Dark Helmet tries to log into C$50 Finance. That call to `sprintf` will create and return the following string.

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

See `http://www.php.net/sprintf` for reference.

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 Lord Dark Helmet's username is `dhelmet` and password is `12345`, though, should the database return an actual row. And, so, if `mysql_num_rows` returns `1`, Lord Dark Helmet 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 `$_SESSION["uid"]` has been assigned any value (*e.g.*, Lord Dark Helmet's user ID). If not, it must be that no one's logged in, else `login2.php` would have assigned it a value, and so we had best redirect traffic to `login.php` (by calling `redirect`, a function defined in `helpers.php`). If, though, `$_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`.

Now, how do we enable Lord Dark Helmet to log out? Why, with `logout.php`! Take a look at that file with Nano. Note that all we need do is obliterate his session via a sequence of statements.[13] Alternatively, Lord Dark Helmet can simply close his own browser, as `$_SESSION` is lost in that case as well.

☐ Phew, that was a lot. Time for a snack?

☐ 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://cs50.net/me/`

Then head to

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

[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] See `http://www.php.net/session_destroy` for the source of this trick.

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.[14] 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.[15] 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 its structure. Notice anything familiar? You should! Recall that `login2.php` generates, via `sprintf`, queries like the below.

```
SELECT uid FROM users WHERE username='dhelmet' 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.[16] 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 `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, let's see if any users exist! Click the tab labeled **Browse** to see the contents of this table. Ah, some familiar folks. In fact, there's Lord Dark Helmet's username and password! Head on back to

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

and try to log in as Lord Dark Helmet. If successful, you'll find yourself at `index.php`, where (for the moment) very little awaits.

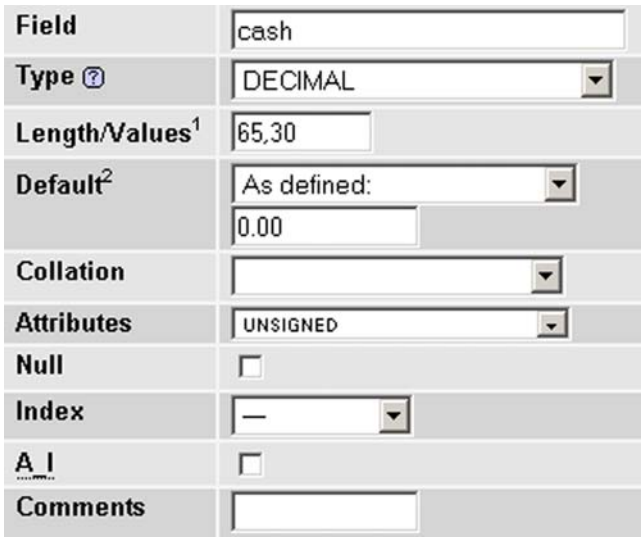
[14] If you walk away from phpMyAdmin for too long, it might "forget" that you're logged in and start pestering you for your username and password. If it then starts rejecting your password altogether, try closing all of your browser's windows and loading phpMyAdmin in a new window thereafter. If the problem still persists, try clearing your cookies too.

[15] 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; you're not required to use it (or any database for that matter).

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

☐ 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 beside **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.



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
$999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999,999.99.00000000000000000000000000000000
in cash, do go with a default of $0.00 (or, more precisely, $0.00000000000000000000000000000000). 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 0.00000000000000000000000000000000 to 10000.00 for each of your users, then click **Go**. Won't they be happy![17]

[17] Of course, they'd be happier with all 65 significant digits.

☐ 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. 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`. (After all, we probably 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 `sprintf` a string like `INSERT INTO users (username, password, cash) VALUES('%s', '%s', 10000.00)` though we leave it to you to decide how much cash your code should give to new users.
iii. Know that `mysql_query` will return `FALSE` if your `INSERT` fails (as can happen if, say, `username` already exists).[18] 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`.[19]
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`.

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

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://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!

In fact, you're even welcome to rename your baby.

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

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

☐ 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.

That's all!

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>
```

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.[20] 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 symbols since those symbols are likely shorter (and thus more efficiently stored) than stocks' actual names.[21] 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.

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

[21] 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.

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.[22] When done defining your table, click **Save**!

☐ Before we let users buy and sell stocks themselves, let's give some freebies to Lord Dark Helmet and friends. Click, in phpMyAdmin's left-hand frame, the link to `users`. Then click the tab labeled **Browse** and remind yourself (assuming your new table's design matches ours) 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`.

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.[23]

```
SELECT * FROM tbl WHERE uid=1
```

Assuming `1` is still Lord Dark Helmet's user ID, that query should return all rows from `tbl` that represent the lord'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=1
```

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

```
SELECT shares FROM tbl WHERE uid=1 AND symbol='HXPN.PK'
```

If you happened to buy the lord some shares of that company, the above should return one row with one column, the number of shares. If you did not think to 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.

[22] 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.
[23] Incidentally, because 1 is a number (just as 10000.00 was earlier), you need not enclose it in quotes like you do strings.

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, Lord Dark Helmet's holdings, one per row in some XHTML table, you can generate rows with code like the below.[24]

```
<?
    $result = mysql_query("SELECT symbol,shares FROM tbl WHERE uid=1");

    while ($row = mysql_fetch_array($result))
    {
        $s = lookup($row["symbol"]);
        print('<tr>');
        print('<td>' . $s->name . '</td>');
        print('<td>' . $row["shares"] . '</td>');
        print('</tr>');
    }
?>
```

Though commonly done, generating XHTML via calls to print isn't terribly elegant. An alternative approach, though still far from ideal, is code more like the below.

```
<? $result = mysql_query("SELECT symbol,shares FROM tbl WHERE uid=1"); ?>

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

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

    <tr>
        <td><? print($s->name); ?></td>
        <td><? print($row["shares"]); ?></td>
    </tr>

<? } ?>
```

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

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

for inspiration or hints. But do not feel obliged to mimic our design. Make this website your own! Incidentally, though we keep using Lord Dark Helmet in examples, your code should work for whichever user is logged in!

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

☐ 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, Lord Dark Helmet) with SQL like the below.

```
DELETE FROM tbl WHERE uid=1 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, the lord).[25]

```
UPDATE users SET cash=cash+500 WHERE uid=1
```

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!

☐ But now it's time to support actual buys. Implement the ability to buy, in, say, `buy.php` and `buy2.php`.[26] The interface with which you provide a user is entirely up to you, though, as before, feel free to look to

```
http://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 "regular expressions." See `http://www.php.net/preg_match` for details. Take care, as always, to apologize to the user if you encounter some problem.

[25] Of course, if the database or webserver happens to die between this `DELETE` and `UPDATE`, Lord Dark Helmet 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 Lord Dark Helmet 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 InnoDB tables and SQL transactions. See `http://dev.mysql.com/doc/refman/5.0/en/innodb.html` for reference.

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

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 `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(1,'HXPN.PK',10)
ON DUPLICATE KEY UPDATE shares=shares+VALUES(shares)
```

As before, be sure to bang on your code!

☐ Do not forget that... Right, bulletin board.

☐ 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.

☐ Phew. Glance back at `index.php` now and, if not there already, be sure 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!

☐ 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!

☐ If you're up for the challenge, why not bite off another feature or two? None of the below is required, and you'll be rewarded only with admiration and praise, but why not see if you can make your site even better? Odds are, you'll learn yet another new trick or two. Feel free to solicit counsel from the course's staff or bulletin board. In fact, you're welcome to collaborate outright with classmates on the features below, inasmuch as none is officially required of you. Interpret each feature below as you will!

- ☐ Empower users to change their passwords.
- ☐ Empower users who've forgotten their password to receive reminders via email. See `http://phpmailer.codeworxtech.com/index.php?pg=tutorial` for reference.
- ☐ Email users "receipts" anytime they buy or sell stocks. See `http://phpmailer.codeworxtech.com/index.php?pg=tutorial` for reference.
- ☐ Empower users to deposit additional funds.
- ☐ Keep users' passwords encrypted in your database. See `http://www.php.net/crypt`, `http://www.php.net/manual/en/ref.mcrypt.php`, and/or `http://dev.mysql.com/doc/refman/5.0/en/encryption-functions.html` for reference.
- ☐ Present users with links to recent articles related to companies for which users have shares. You may find the `stock` class's `items` of interest.
- ☐ Download yet additional fields from Yahoo, updating `helpers.php` and `stock.php` accordingly, and provide users with access to that data.
- ☐ Integrate more sources of data than just MSN and Yahoo.
- ☐ Implement the ability to short stocks!
- ☐ **. . .**

☐ 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://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 Monday, 8 December 2008, who exits this course with the most money in hand.[27]

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

[27] We have yet to pay out.

Submitting Your Work.

☐ Ensure that your home page is in `~/public_html/` and your implementation of C$50 Finance is in `~/public_html/pset7/`.[28] 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.

[28] 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.