Computer Science 50 Week 9 Monday: November 1, 2010
Fall 2010 Andrew Sellergren
Scribe Notes

Contents

1 Announcements and Demos (0:00–10:00)

- This is CS50.
- 1 new handout.
- Many exciting things in store this week, including teaching you how to re-implement the deprecated (with good reason) HTML `blink` tag using JavaScript!
- Movie night hosted by Microsoft will be held this Friday at 7:30 PM at the New England Research and Development (NERD) center. There will be pizza, candy, and a showing of Office Space! RSVP at cs50.net/movie.
- Coming up are CS50 Seminars in which staff members and course affiliates give short lectures on topics of interest in technology, particularly those which might have bearing on final projects. Check out this year's lineup. Be sure to RSVP so that the seminar leader can find an optimal day and time.
- The Final Project Pre-proposal is due this Monday! As the specification details, it need only be a short e-mail to your teaching fellow bouncing ideas off him or her.
- Today in Mather House JCR at 5 PM, the IOP welcomes Alec Ross, the Senior Advisor for Innovation to Hillary Clinton, who heads up the administration's efforts to find practical technological solutions to problems like healthcare and poverty.
- Many of you probably received the same spam message that David did, asking him as a Harvard webmail user to provide his login credentials so that his account wouldn't be shut down. Hopefully you didn't actually provide your credentials, as this was a fairly obvious *phishing attack*. Phishing attacks are attempts to steal login information for legitimate websites by creating phony websites that impersonate them. Based on what we've learned of HTML, we know that the link in the e-mail which appeared to point to `harvard.edu`, could easily point to a phony website like so:

  ```
  <a href="http://123contactform.com/">http://Harvard.edu/Secure/login</a>
  ```

  In most browsers, you can hover over a link to see where it leads and we highly recommend doing so before clicking links from untrusted sources.

2 Problem Set 7 (10:00–19:00)

- In Problem Set 7, you'll be tasked with implementing a stock-trading website which will allow users to register, login, and manage fake stock portfolios based on data from Yahoo! Finance. The transactions that users make, namely buying and selling stocks, will be logged in a MySQL database.

- You should play around with the staff solution before implementing your own. Also click the play the Big Board link to compete with your classmates for the title of best stock trader.

- If we go to Yahoo! Finance and enter in a stock symbol, we can see near-realtime data on its price. How do we go about retrieving this data for our own application's use? We could resort to screen-scraping, meaning we write a script to make an HTTP GET request (notice the stock symbol becomes embedded in the URL when we do a lookup) for the stock lookup page and pull out the data we're interested in. We could even write a `cron` job, a scheduled programmatic task, that does this periodically. However, this technique is tedious and not robust, as the HTML source code is often messy and our screen-scraping program will break as soon as the page's format changes. What's more convenient is the "Download Data" link under Toolbox which actually points to a different URL. This URL actually spits out a CSV (comma-separated values) file. A CSV file is a special kind of plaintext file that stores data with values separated by commas, as its name implies. CSV files are particularly easy to parse with scripting languages like PHP, so when it comes to grabbing Yahoo! Finance data, we'll be better off using this download link.

- When we download this CSV file and double click to open it, we see that it's nicely displayed in column format in Excel. However, we can also open this file in a simple text editor and see that it contains nothing but a single line of strings and numbers delimited by commas.

- For Problem Set 7, we'll actually do much of the heavy lifting for you. We've written a function that will grab the CSV file from Yahoo! Finance, parse it, and return it to you as a PHP variable.

3 From Last Time (19:00–34:00)

- In our first foray into PHP, we were able to create a website where freshmen could provide their registration information which we would then throw away. Eventually, however, we were able to write enough code to e-mail that information to a designated account rather than discarding it.

- Considering where we've come from with the verbosity of C, the amount of PHP code we had to write in order to send an e-mail was comparatively small. We used the dot operator to concatenate the pieces of the user's information and then a function called `mail` to actually send them. Splicing together strings in PHP proves to be much easier than in C because arrays don't have a fixed size. In fact, the variable `$_POST` was a special kind of array that allowed us to store data indexed with a string rather than a number. So `$_POST["name"]` contained the user's name, for example. We call `name` a *key* for this array.

- Question: what happens when you try to access a key in the `$_POST` array that doesn't exist? PHP will return the empty string.

- `$_POST` is a special variable called a superglobal. Whenever a PHP page is accessed via a form submission using the POST method, this variable will be automatically populated with the submitted data. The `action` attribute of the `form` tag specifies the page that will receive the data submitted by our form. The `method` attribute of the `form` tag is either GET or POST, where GET means that the data is appended to the URL and POST means that the data is appended to the HTTP headers. GET is useful for creating URLs that can be bookmarked, but POST is useful for submitting sensitive or lengthy data.

- In our `form` tag, we have `input` tags that have different values for the `type` attribute. In this example, we have `text`, `checkbox`, and `radio` for the name, captain, and gender fields respectively.

- In `register3.php`, we first do a sanity check: if any of the `name`, `gender`, and `captain` keys in the `$_POST` array have not been set, i.e. if the user didn't fill in these inputs in the form, then we'll print out a message telling the user to enter all required fields and provide him with a link to go back. If, however, the user has provided all the required inputs, then we send the e-mail with his information embedded.

- One interesting thing to note is that we can send the e-mail from any address we like just by providing it as an argument to the `mail` function. It's just that easy to spoof an e-mail address because e-mail was never designed with security in mind. If we take the time to inspect the actual headers of the e-mail, we'll find that it originated from the CS50 Cloud's servers, so it is possible to do a bit of spam fighting if we really want to. We will advise you not to actually make use of this e-mail address spoofing, as David got into a bit of trouble doing once because he failed to remove his automatic signature "djm" while impersonating a friend whose initials, needless to say, aren't "djm."

4 More with Forms and PHP (34:00–46:00)

4.1 `froshims4.php`

- `froshims4.php` introduces an interesting technique; it submits to itself:

```
<?
    /*************************************************************************
     * froshims4.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.  Submits to itself.
     **********************************************************************/

    // if form was actually submitted, check for error
    if ($_POST["action"])
    {
        if ($_POST["name"] == "" || $_POST["gender"] == "" || $_POST["dorm"] == "")
            $error = TRUE;
    }
?>

<!DOCTYPE html>

<html>
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    <div style="text-align: center">
      <h1>Register for Frosh IMs</h1>
      <? if ($error): ?>
        <div style="color: red">You must fill out the form!</div>
      <? endif ?>
      <br><br>
      <form action="froshims4.php" method="post">
        <table style="border: 0; margin-left: auto;
             margin-right: auto; text-align: left">
          <tr>
            <td>Name:</td>
            <td><input name="name" type="text"></td>
          </tr>
          <tr>
            <td>Captain:</td>
            <td><input name="captain" type="checkbox"></td>
          </tr>
          <tr>
            <td>Gender:</td>
            <td><input name="gender" type="radio" value="F"> F
                <input name="gender" type="radio" value="M"> M</td>
          </tr>
          <tr>
            <td>Dorm:</td>
            <td>
              <select name="dorm" size="1">
                <option value=""></option>
                <option value="Apley Court">Apley Court</option>
                <option value="Canaday">Canaday</option>
                <option value="Grays">Grays</option>
                <option value="Greenough">Greenough</option>
                <option value="Hollis">Hollis</option>
                <option value="Holworthy">Holworthy</option>
                <option value="Hurlbut">Hurlbut</option>
                <option value="Lionel">Lionel</option>
                <option value="Matthews">Matthews</option>
                <option value="Mower">Mower</option>
                <option value="Pennypacker">Pennypacker</option>
                <option value="Stoughton">Stoughton</option>
                <option value="Straus">Straus</option>
                <option value="Thayer">Thayer</option>
                <option value="Weld">Weld</option>
                <option value="Wigglesworth">Wigglesworth</option>
              </select>
            </td>
          </tr>
        </table>
        <br><br>
        <input name="action" type="submit" value="Register!">
      </form>
    </div>
  </body>
</html>
```

4.2 froshims5.php

- froshims5.php makes use of submitting to itself as a way of error-checking:

```
<?
    /***********************************************************************
     * froshims5.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.  Submits to itself.
     * Pre-populates name field upon error.
     **********************************************************************/

    // if form was actually submitted, check for error
```

```
        if ($_POST["action"])
        {
            if ($_POST["name"] == "" || $_POST["gender"] == "" || $_POST["dorm"] == "")
                $error = TRUE;
        }
    ?>

    <!DOCTYPE html>

    <html>
      <head>
        <title>Frosh IMs</title>
      </head>
      <body>
        <div style="text-align: center">
          <h1>Register for Frosh IMs</h1>
          <? if ($error): ?>
            <div style="color: red">You must fill out the form!</div>
          <? endif ?>
          <br><br>
          <form action="froshims5.php" method="post">
            <table style="border: 0;
                margin-left: auto; margin-right: auto; text-align: left">
              <tr>
                <td>Name:</td>
                <td><input name="name" type="text"
                    value="<?= htmlspecialchars($_POST["name"]) ?>"></td>
              </tr>
              <tr>
                <td>Captain:</td>
                <td><input name="captain" type="checkbox"></td>
              </tr>
              <tr>
                <td>Gender:</td>
                <td><input name="gender" type="radio" value="F"> F
                    <input name="gender" type="radio" value="M"> M</td>
              </tr>
              <tr>
                <td>Dorm:</td>
                <td>
                  <select name="dorm" size="1">
                    <option value=""></option>
                    <option value="Apley Court">Apley Court</option>
                    <option value="Canaday">Canaday</option>
                    <option value="Grays">Grays</option>
                    <option value="Greenough">Greenough</option>
                    <option value="Hollis">Hollis</option>
                    <option value="Holworthy">Holworthy</option>
                    <option value="Hurlbut">Hurlbut</option>
                    <option value="Lionel">Lionel</option>
                    <option value="Matthews">Matthews</option>
                    <option value="Mower">Mower</option>
                    <option value="Pennypacker">Pennypacker</option>
                    <option value="Stoughton">Stoughton</option>
                    <option value="Straus">Straus</option>
                    <option value="Thayer">Thayer</option>
                    <option value="Weld">Weld</option>
                    <option value="Wigglesworth">Wigglesworth</option>
                  </select>
                </td>
              </tr>
            </table>
            <br><br>
            <input name="action" type="submit" value="Register!">
          </form>
        </div>
      </body>
    </html>
```

The value in submitting a form to itself, as we've done by specifying `froshims5.php` as the value of the `action` attribute here, is that we can keep the form's inputs populated if the user makes a mistake in entering them. Because the form submits to itself, we can move the logic that checks for correct inputs onto the same page as the form itself. Then, if the user submits faulty data, we can display an error message on the form page and not lose the user's inputs in moving from one page to another.

- More specifically, if there's an error in the user's form, we set a variable named `$error` to `TRUE`. Later in our code, we check the value of that variable:

```
<? if ($error): ?>
    <div style="color: red">You must fill out the form!</div>
<? endif ?>
```

Notice that we can commingle PHP and HTML code relatively seamlessly. In this case, if `$error` evaluates to true, we'll display an error message in red.

4.3 `froshims6.php`

- In `froshims6.php`, we take a stab at using PHP to help automatically generate some of our HTML form:

```
<?
    /***********************************************************************
     * froshims6.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.  Submits to itself.
     * Generates list of dorms via an array.
     **********************************************************************/

    // array of dorms
    $DORMS = array(
     "Apley Court",
     "Canaday",
     "Grays",
     "Greenough",
     "Hollis",
     "Holworthy",
     "Hurlbut",
     "Lionel",
     "Matthews",
     "Mower",
     "Pennypacker",
     "Stoughton",
     "Straus",
     "Thayer",
     "Weld",
     "Wigglesworth"
    );

    // if form was actually submitted, check for error
    if ($_POST["action"])
    {
        if ($_POST["name"] == "" || $_POST["gender"] == "" || $_POST["dorm"] == "")
            $error = TRUE;
    }
?>


<!DOCTYPE html>

<html>
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    <div style="text-algin: center">
      <h1>Register for Frosh IMs</h1>
      <? if ($error): ?>
        <div style="color: red;">You must fill out the form!</div>
      <? endif ?>
      <br><br>
      <form action="froshims6.php" method="post">
        <table style="border: 0;
            margin-left: auto; margin-right: auto; text-align: left">
          <tr>
            <td>Name:</td>
            <td><input name="name" type="text"
                value="<?= htmlspecialchars($_POST["name"]) ?>"></td>
          </tr>
          <tr>
            <td>Captain:</td>
            <td><input name="captain" type="checkbox"></td>
          </tr>
          <tr>
            <td>Gender:</td>
            <td><input name="gender" type="radio" value="F"> F
                <input name="gender" type="radio" value="M"> M</td>
          </tr>
          <tr>
            <td>Dorm:</td>
            <td>
              <select name="dorm" size="1">
                <option value=""></option>
                <? foreach ($DORMS as $dorm): ?>
                  <option value="<?= $dorm ?>"><?= $dorm ?></option>
                <? endforeach ?>
              </select>
            </td>
          </tr>
        </table>
        <br><br>
        <input name="action" type="submit" value="Register!">
      </form>
    </div>
  </body>
</html>
```

The `$DORMS` variable is declared using the `array` function and is initialized with a series of strings denoting the names of the freshman dorms on Harvard's campus. Using this array, we can create our `select` element without having to hardcode all of the `option` tags. As before, we write the open tag for the `select` element and specify the first `option` as blank so that it will be the default:

```
<select name="dorm" size="1">
 <option value=""></option>
```

After that, though, we enter PHP mode in what looks to be a loop:

```
<? foreach ($DORMS as $dorm): ?>
   <option value="<?= $dorm ?>"><?= $dorm ?></option>
<? endforeach ?>
```

Using the `foreach` syntax, we loop through and grab each value of the `$DORMS` array and assign it to the temporary variable `$dorm`. Then we print out `$dorm` as both the `value` attribute and the display name of the `option` itself. You can imagine that if HarvardFML is implemented in PHP, the `foreach` loop could be used to print out all of the posts for a particular date.

- The actual HTML output of `froshims6.php` is actually the same as our previous versions, but the back-end source code is a little cleaner thanks to PHP.

5 Database-driven Websites (46:00–55:00)

- Before we examine the most compelling version of our freshman intramurals registration page, let's talk about the database behind it. A database is a storage platform that organizes persistent data into tables. You can think of a database and its tables as an Excel file and the spreadsheets within it, respectively.

- Database engines abound. There are Microsoft Access, MySQL, Oracle, and Microsoft SQL Server just to name a few. We'll be using MySQL because it's free and open-source (i.e. the source code is publicly available and modifiable). Facebook is one of MySQL's largest proponents and has shown that MySQL can be scalable enough to run enterprise applications. MySQL is generally very popular and is widely supported by domain hosts.

- To interact with our database server, we'll be using phpMyAdmin, another excellent piece of free, open-source software. We've set up an instance of it at cs50.net/phpmyadmin.

- Intuitively, we can organize our freshman registration information into a single table called "registrants" that contains columns for name, gender, dorm, and captainship. For performance reasons, MySQL requires strict data typing, so we need to decide ahead of time what data types these four columns will have. Let's go with `VARCHAR` (a type of string which can have variable length) for name, with a maximum length of 255, `BOOLEAN` for captain, `VARCHAR` for gender (although there are many different data types that could work here), and `VARCHAR` for dorm, with a maximum length of 255. When we click the Save button, the table will be created, but we'll also see the actual SQL statement which was executed. SQL, by the way, stands for *structured query language*.

5.1 `register8.php`

- Let's take a look at how we will interact with our newly created database using PHP in `register8.php`:

```
<?
    /***********************************************************************
     * register8.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.  Records registration
     * in database.  Redirects user to froshims8.php upon error.
     **********************************************************************/

    // validate submission
    if ($_POST["name"] == "" || $_POST["gender"] == "" || $_POST["dorm"] == "")
    {
        header("Location: http://cloud.cs50.net/" .
               "~cs50/lectures/8/src/froshims/froshims8.php");
        exit;
    }

    // connect to database
    mysql_connect("localhost", "malan", "12345");
    mysql_select_db("malan_lecture");

    // scrub inputs
    $name = mysql_real_escape_string($_POST["name"]);
    if ($_POST["captain"])
        $captain = 1;
    else
        $captain = 0;
    $gender = mysql_real_escape_string($_POST["gender"]);
    $dorm = mysql_real_escape_string($_POST["dorm"]);

    // prepare query
    $sql = "INSERT INTO registrants (name, captain, gender, dorm)
     VALUES('$name', $captain, '$gender', '$dorm')";

    // execute query
    mysql_query($sql);
?>

<!DOCTYPE html>

<html>
  <head>
    <title>Frosh IMs</title>
  </head>
  <body>
    You are registered!  (Really.)
  </body>
</html>
```

First, we connect to the database server by providing our login credentials to the `mysql_connect` function. Second, because there will be multiple databases on this server, we must select ours by calling the `mysql_select_db` function. Third, we need to scrub the data that the user has provided. That is, because there are a lot of ways that malicious users can sabotage our database if we blindly pass their inputs to the database server, we need to go through a process of cleaning those inputs. We do this with the `mysql_real_escape_string` function. Fourth, we will create a SQL query and execute it using the `mysql_query` function. Assuming all of these steps are successful, the user's data will be inserted into our MySQL database.

- If we navigate to `froshims8.php`, which is nearly identical to earlier versions of the form, and enter in our data, we see a message indicating that we've successfully registered. The real test comes, however, when we check the contents of our database using phpMyAdmin. And ta-da, it's there!

6 More on Problem Set 7 (55:00–58:00)

- David has fixed the Big Board so that it properly displays the portfolio rankings and the profit percentages! If you're feeling particularly enterprising, know that you can make a whole slew of money by exploiting two weaknesses of our system: first, our stock price data is delayed by 15 minutes and second, your purchases won't affect the stock prices. May the best hacker win!

7 PHP and MySQL (58:00–72:00)

- SQL, or structured query language, isn't exactly a programming language. Rather, it is a language that enables us to get at the data that is stored in databases. For our purposes, SQL boils down to four basic commands:
  - `INSERT`
  - `SELECT`
  - `UPDATE`
  - `DELETE`

  We've actually already used the `INSERT` command in `register8.php`. We'll peel back the layers on all of these commands shortly.

- Interfacing with databases via SQL is a capability of almost all modern programming languages, not just PHP.

7.1 `register8.php` (cont'd)

- In `register8.php`, we connected to our MySQL database using a function called `mysql_connect` which took three arguments: the IP address of the server, the username, and the password.[1]

- `mysql_real_escape_string` is a function that we called in order to sanitize the user's inputs before inserting them into our database. This is to prevent a so-called *SQL injection attack*,[2] whereby a malicious user could actually provide as a form input a SQL command that would then be executed and would sabotage our database.

- To set the `$captain` variable, we treat `$_POST["captain"]` as a boolean value. Recall that if the checkbox for captainship was checked, this value would be passed as "on". If it wasn't checked, the value wouldn't be passed at all. So by testing if `$_POST["captain"]` is set, we're finding out if the checkbox was checked and if so, setting the `$captain` variable to 1, or true.

- As an aside, notice that we haven't had to declare variables in PHP. We can create new variables on the fly simply by assigning them a value and not even specifying their type.

- We prepare our SQL statement as follows:

```
// prepare query
$sql = "INSERT INTO registrants (name, captain, gender, dorm)
VALUES('$name', $captain, '$gender', '$dorm')";
```

[1] By the way, `mysql.localdomain` is a made-up IP address that is accessible only internally on the Cloud since we don't want outsiders to be able to connect to it.

[2] Everyone could learn a lesson from Little Bobby Tables's mom.

After specifying what table to insert the data into (`registrants`), we give a list of column names in parentheses separated by commas. This list of columns isn't entirely necessary, but it allows us to omit certain columns in the table if we don't want to insert any data into them. In this case, we're being explicit by naming all of the columns.

- The `VALUES` string is then populated with the user's santized inputs. In PHP, variables that we specify inside double quotes will automatically be replaced with their values at runtime. SQL also requires that string values be surrounded by single quotes, so this is why only `$captain` is not.

- The final value of `$sql` will look something like this:

  ```
  INSERT INTO registrants
  (name, captain, gender, dorm)
  VALUES('David Malan', 1, 'M', 'Matthews');
  ```

  We can actually copy and paste this query into phpMyAdmin's SQL tab and execute it directly against our database to verify that it works.

7.2 `registrants.php`

- We have one more script to look at which allows us to examine the data in our database:

  ```
  <?
      // connect to database
      mysql_connect("mysql.localdomain", "malan", "12345");
      mysql_select_db("malan_lecture");

      // prepare query
      $sql = "SELECT * FROM registrants";

      // execute query
      $result = mysql_query($sql);


  ?>

  <!DOCTYPE html>

  <html>
    <head>
      <title>Frosh IMs</title>
    </head>
    <body>
      <ul>
  ```

```
      <?
          // iterate over results
          while ($row = mysql_fetch_array($result))
          {
              print("<li>");
              print(htmlspecialchars($row["name"]));
              print("</li>");
          }
      ?>
    </ul>
  </body>
</html>
```

The while loop is iteratively grabbing one row at a time from our SQL query result until there are no rows left, in which case `mysql_fetch_assoc()` will return false.