Computer Science 50 Week 9 Wednesday: November 2, 2011
Fall 2011 Andrew Sellergren
Scribe Notes

Contents

1 Announcements and Demos (0:00–4:00)

- Another CS50 Lunch this Friday! RSVP here. This week we'll be joined by Eugene Chung of NEA and Andrew McCollum, formerly of Facebook.
- Don't forget to check out the CS50 Seminars. Microsoft has now kindly contributed some Windows smartphones to supplement the BlackBerry phones contributed by RIMM.
- Mark Zuckerberg will be visiting campus on Monday! Space in the panel session on Monday night will be limited, so follow this link and do a keyword search for 31911 to throw your name in the hat. And, sorry, don't get your hopes up: Mark will be visiting MIT during lecture on Monday, so don't come to Sanders expecting to see anyone more famous than David.
- A special shout out to Jason Hirschhorn who taught section on Monday while dressed as a giant pumpkin (and wearing nothing underneath).

2 From Last Time (4:00–10:00)

2.1 `froshims5.php`

- We looked at creating dynamic websites using PHP, a programming language, and MySQL, a database language. Recall `froshims5.php`:

```
<?
    /***********************************************************************
     * 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 (isset($_POST["action"]))
    {
        if (empty($_POST["name"]) || empty($_POST["gender"]) || empty($_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">
                <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>
```

One nice feature of this version of our Frosh IMs registration form was that if the user failed to provide all the necessary inputs, he would be shown an error message and his name would be prepopulated in the name field. Notice how easy it was to accomplish this and also how many websites still don't do it. User interface design is a highly underappreciated aspect of web programming, to be sure.

- `htmlspecialchars` was a function we used to "scrub" the user's input and prevent HTML or JavaScript from being injected into our website. This is the first of many instances in which we will distrust user input. Consider that HTML forms which use the GET method can be easy linked to from e-mails. Many phishing attacks rely on this to cause user's to accidentally submit forms simply by being tricked into clicking on a link.
- Although `froshims5.php` prepopulated the user's name, it did not prepopulate the user's dorm upon invalid submission. The dorm field is implemented as a `select` tag, so we have to handle it slightly differently than the name field. In order to prepopulate the dorm field, we'll need to output an extra attribute `selected` for the specific `option` tag that was chosen by the user.

3 More with Frosh IMs (10:00–38:00)

3.1 `froshims6.php`

- `froshims6.php` takes a slightly more elegant approach to creating the options for the dorm dropdown menu:

```
<?
    /*********************************************************************
     * 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 (isset($_POST["action"]))
    {
        if (empty($_POST["name"]) || empty($_POST["gender"]) || empty($_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">
                  <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>
```

Because we have to output some similar HTML for each of the dorms in the dropdown, we can abstract away the names of the dorms into an array named `$DORMS`. Unfortunately, PHP requires the `array` function to be called when initializing an array rather than just writing open and close square brackets. Using the `foreach` construct, we loop through all of the elements of the `$DORMS` array and print the dorm name as both the `value` attribute and the inner HTML of the `option` tag. Note that the colon at the end of the `foreach` construct signifies that we're still inside the loop even though we're exiting PHP mode.

- Why aren't we escaping the name of the dorm? Well, we're the ones who created the `$DORMS` array, so presumably we haven't attempted to exploit our own code.
- When we visit `froshims6.php` and right click to View Source, we see that the HTML for the dorm dropdown menu is identical to `froshims5.php`, with the exception of whitespace.

3.2 `froshims7.php`

- `froshims7.php` takes on the challenge of prepopulating the dorm field:

```
<?
    /***********************************************************************
     * froshims7.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a registration form for Frosh IMs.  Submits to itself.
     * Generates list of dorms via an array.  Pre-populates name and
     * dorm fields upon error.
     **********************************************************************/

    // 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 (isset($_POST["action"]))
    {
        if (empty($_POST["name"]) || empty($_POST["gender"]) || empty($_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 ($error): ?>
        <div style="color: red;">You must fill out the form!</div>
      <? endif ?>
      <br><br>
      <form action="froshims7.php" method="post">
        <table style="border: 0; margin-left: auto;
          margin-right: auto; text-align: left">
          <tr>
            <td>Name:</td>
            <td><input name="name" type="text"
              value="<?= htmlspecialchars($_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">
                <option value=""></option>
                <?
                    foreach ($DORMS as $dorm)
                    {
                        if ($_POST["dorm"] == $dorm)
                            echo "<option selected='selected'" .
                                "value='$dorm'>$dorm</option>";
                        else
                            echo "<option value='$dorm'>$dorm</option>";
                    }
                ?>
              </select>
            </td>
          </tr>
        </table>
        <br><br>
        <input name="action" type="submit" value="Register!">
      </form>
    </div>
  </body>
</html>
```

We have the same `foreach` construct that appeared in `froshims6.php` but this time, if the dorm name is what the user selected previously (i.e. is the value for the `dorm` key in the `$_POST` array), then we output the extra `selected` attribute as well.

- Question: what does `echo` do? It's synonymous with `print`.
- Question: where is the `$dorm` variable introduced? It's defined implicitly by the `foreach` construct.

3.3 `register8.php`

- Recall that along with the destination IP address, a server request is accompanied by a port number so that the server knows what type of request it is and how to handle it. For example, HTTP requests are most often tied to ports 80 and 443 (the latter for SSL), SSH requests are tied to port 22, FTP requests are tied to port 21, and so on. MySQL requests are tied to port 3306, by default.
- In `register8.php`, we use a function `mysql_connect` to connect to our MySQL server:

```
<?
    /*********************************************************************
     * 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 (empty($_POST["name"]) || empty($_POST["gender"]) || empty($_POST["dorm"]))
    {
        header("Location: http://localhost/~jharvard/froshims/froshims8.php");
        exit;
    }

    // connect to database
    mysql_connect("localhost", "jharvard", "crimson");
    mysql_select_db("jharvard_week9");

    // 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>
```

We use `localhost` as the hostname (the first argument to `mysql_connect`) because the web server and the database server are both running on the same machine, namely the Appliance. In general, it's not a good idea to connect to a database server on a different hostname than the web server since MySQL traffic is not encrypted by default. We use the `mysql_select_db` function to choose the `jharvard_week9` database, much like choosing a specific spreadsheet within an Excel file. After we're connected, we scrub the user's input using the `mysql_real_escape_string` function. Just as we distrust user's input when it comes to prepopulating form fields, we distrust user's input when it comes to inserting into our database. By scrubbing user's input, we avoid accidentally deleting or exposing data in our database.

- On a somewhat related note, one of the top complaints from the Problem Set 5 survey was getting menu information from `dining.harvard.edu`. If this was one of your complaints, then you should take heart knowing that CS50 has an API for accessing this menu information. Using this API, you can create your own more intuitive way of displaying and accessing Harvard menu information. There are other APIs available for accessing campus events, tweets, maps, news, and more. The HarvardFood API is implemented as a screen scraper, that is, a program that acts as a web browser, downloads the menu web page, and parses the HTML to find the actual menu items.

- The actual SQL statement that inserts data into our database contains two comma-separated lists of columns and values and not much more. To execute this statement, we pass it to the `mysql_query` function.

3.4 `registrants.php`

- `registrants.php` is a script that connects to our database and pulls all existing entries in our `registrants` table:

```
<?
    // connect to database
    mysql_connect("localhost", "jharvard", "crimson");
    mysql_select_db("jharvard_week9");

    // 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 * is the wildcard operator, meaning select all fields. SQL, by the way, is case-insensitive (except for field, table, and database names), but as a matter of style, it's good practice to capitalize keywords like `SELECT`.

- What do you actually get back from a database when you execute a `SELECT` statement? `mysql_query` will return you an array of rows. To loop over this array of rows, we use the `mysql_fetch_array` function which returns the next row in the result set as an associative array or false if there are no more rows left.

- When we navigate to this web page, we see that all of the registrants' names are displayed as bullet points.

- In the `users` table we provide you for Problem Set 7, there are fields for username, password hash, and user ID. This user ID is unique and auto-incrementing, meaning that every time a user is added to the table, the maximum user ID increases by 1. Facebook operates similarly: the lower your user ID (which might be visible in the URL of your profile), the earlier you joined Facebook. But if we ensure that usernames are unique, why bother assigning an extra unique identifier? Looking up users by a number rather than a string should be much faster given that integer comparisons are less expensive than string comparisons. Using integers as unique identifiers also optimizes other tables that we might add later. If we want to implement a table that stores relationships between users, we need only store two integers, one for each user, rather than two strings, which will most likely require more bytes.

4 Sessions (38:00–57:00)

4.1 `counter.php`

- We mentioned last time that HTTP is a stateless protocol, meaning that as soon as a request is served with a response, the connection to the server is severed and the server more or less "forgets" about the client. This isn't very convenient for sites that implement things like shopping carts, however, since the web server needs a way to remember which items a user has selected for purchase. This is where sessions come in.
- `counter.php` demonstrates a very simple use of sessions:

```
<?
    /***************************************************************
     * counter.php
     *
     * Computer Science 50
     * David J. Malan
     *
     * Implements a counter.  Demonstrates sessions.
     **************************************************************/

    // enable sessions
    session_start();

    // check counter
    if (isset($_SESSION["counter"]))
        $counter = $_SESSION["counter"];
    else
        $counter = 0;

    // increment counter
    $_SESSION["counter"] = $counter + 1;
?>

<!DOCTYPE html>

<html>
  <head>
    <title>Counter</title>
  </head>
  <body>
    You have visited this site <?= $counter ?> time(s).
  </body>
</html>
```

To begin working with sessions, we call `session_start`, a function which tells PHP to give our program access to a special superglobal variable named `$_SESSION`. This is a variable in which we can store anything we want to related to the particular user that is currently interacting with our site. By storing this information in a server-side variable, we allow a user to be remembered by the server even after he or she closes our site.

- In this simple script, we check to see if there is a `counter` index set in `$_SESSION`. If there is, we set a local variable `$counter` to the value stored in `$_SESSION`. If this index isn't set, we initialize `$counter` to 0. Finally, we store in `$_SESSION` the value of `$counter + 1`.

- The effect of this simple script is to report how many times the user has visited this page. As we hit refresh over and over, the number increases. For the purposes of a site like Facebook or Amazon, this counter variable might be replaced by a simple flag indicating whether the user is logged in or not. That way, the user doesn't have to be prompted to login on every single page.

- To dig more into this concept of sessions, let's turn to our old friend Firebug. When we click on the Net tab in Firebug and enable it, we get a view of HTTP requests that are made by the browser much like we did using Live HTTP Headers. When we visit `counter.php`, we see that a single GET request is made to Remote IP 127.0.0.1:80, the numeric equivalent of localhost, on port 80, the default port for HTTP. The size of the request was 141 bytes. When we expand the request, we can see the actual headers that were sent back and forth between client and server. Among other things, the client sends its user agent and the encodings it accepts (for example, gzip indicates that the browser supports a type of compression that helps expedite requests and responses). In its response, the server sends the date and time, its operating system and PHP version information (which can be a security concern given that a malicious user might know of exploits in older versions of software that have been fixed in later versions), and, in the Set-Cookie header, a unique session identifier that might look like the following:

  ```
  PHPSESSID=qcm25jiui0bvjasf05o2t509a0
  ```

  This pseudorandom string is a so-called *cookie* which can be used to identify a particular user. To cooperate with the server, the client will store this cookie locally and send it back to the server whenever it makes another request of it. You can think of cookies as handstamps that tell the server: this user has been here before, look up anything we know about her.

- Sessions and cookies work great in practice, but they are also prone to abuse. By default, cookies are sent in plaintext from client to server. If a malicious user places himself between client and server and intercepts a request that contains a cookie, he can use that cookie to impersonate another user. The Firefox plugin Firesheep demonstrates just how easy it is to steal cookies in this manner. One way of addressing this vulnerability is to encrypt traffic sent between client and server, as is the case for requests made to URLs beginning with `https`. Still, most websites, including Facebook, do not use `https` by default because it is more expensive. If this concerns you,[1] you might consider downloading the Force-TLS or HTTPS Everywhere Firefox plugins which attempt to use `https` whenever it is supported by a website. Another way of addressing this vulnerability is using wireless network encryption such as WPA2. You may have (or should have) enabled this on your home wireless network so that your traffic is not easily readable by those who shouldn't be reading it. If you're on a public wireless network that doesn't support encryption, you can login to a VPN (virtual private network) such as Harvard's which will encrypt your traffic and send it to an intermediate server which you presumably trust before sending it to the web server that will actually fulfill your request.

[1]It probably should, even if you don't wear a tinfoil hat.