6.470 IAP 2014
PHP Exercises

Here are some exercises on PHP. The first two are simply to get you used to the syntax of PHP, and the last two go over many server-side programming issues that you will encounter frequently.

In the first two exercises, you are asked to write a function first. You can check yourself by calling the function in the script after it is declared. If the file is named filename.php, simply type ‘php filename.php’ on an Athena or Mac OSX terminal, and observe the output. For example:

Afterwards, you will need to use the function, in conjunction with other code and HTML, to get the desired output on a webpage.

Exercise 1: Charlie bit my finger!

Part 1: Charlie will bite your finger exactly 50% of the time. First, write a function `isBitten()` that returns TRUE with 50% probability, and FALSE otherwise.
Hint: You may find the `rand()` *function useful.*

Part 2: Below the function, write code to generate a webpage that displays "Charlie bit your finger!" or "Charlie did not bite your finger!" using the `isBitten()` function.

Exercise 2: Counting words (after lecture 6)

Part 1: Write a function `countWords($str)` that takes any string of characters and finds the number of times each word occurs. You should ignore the distinction between capital and lowercase letters, and do not have to worry about dealing with characters that are not letters.
Hint: Create an associative array mapping word keys to the number of times they occur. You will need to look at PHP's string functions to split a sentence into words.
Hint 2: The `print_r($array_name)` *function is useful for examining the contents of an array.*

Part 2: In the same file, write a form consisting of a single text field and a submit button. The "action" attribute to the form should be the same page that the form is on (don't hard code, use `$_SERVER['PHP_SELF']`). The form should send the contents of the text field via GET.

Upon submitting the form, you should be redirected to the same page, but the URL should contain the string from the text field as a GET request normally behaves.

Part 3: Check for the existence of a GET request. If a GET request exists with the name of the input field that you made in Part 2, run the contents of the request through your `countWords($str)` function. Take the output array of the function, and display an HTML table of words and the number of times they occur. Make the table sorted by decreasing number of occurrences.

Exercise 3: Register and login

In this exercise, you will implement a basic registration and login system, without worrying about any styling issues. You will get practice with forms, sending requests, and working with a MySQL database.

In parts 2-3, the behavior of your page depends on whether a POST request exists. Basically, you're sending the form to the same PHP file that originally generated the form. Thus, your PHP files need to detect the existence of values in the $_POST array, and respond accordingly.

Part 1: Make the database connection

Write a PHP file that can be added to other PHP files using the `include` or `require` functions. This file should:

- Make a connection to a MySQL database, and log in with valid credentials. The connection resource should be stored in a variable with an appropriate name.
- Create a database 6470login if it does not exist. (On scripts.mit.edu, you might need to do this through their interface)
- Select the 6470login database.
- Create a table users if it does not exist with the following fields:
    - USERNAME VARCHAR(20)
    - PASSWORD VARCHAR (20)
    - PHONE VARCHAR(10)
- The USERNAME field should be designated as UNIQUE.
- If any of these operations cause an error, stop execution and print the error message

Part 2: Write the registration form

Note that all of this part should be done in the *same* PHP file. The script should respond differently depending on the situation (whether a POST request exists, whether the username is already taken, etc.).

Write a PHP file that will output a form containing 3 fields: username, password, and phone number. These fields should be sent via POST to the same file, which should take care of inserting them into a database named 6470 and table named 6470exerciseusers (as shown above), and then confirm the registration by displaying the username and phone number back to the browser.

If the username already exists, your INSERT query should fail if you designated the USERNAME field as unique. You should query the database before attempting the insert, and if the username exists already, display an error message and a blank registration form again.

Remember to properly escape user input before making the database query.

Part 3: Write the login form

Write a PHP file that will output a form containing 2 fields: username and password. Upon submission of the form, the code should check against the database to see whether the username-password pair was correct. If so, display a welcome message. If not, display the message "Invalid username or password" followed by the same login form.

Once again, there should only be one PHP file, and you should redirect to the same place after submitting. The output should be one of three options:

1. The login form.
2. The welcome message, if successful login.
3. The invalid message and the login form, if failed login.

Since we haven't implemented cookies or sessions, if you successfully log in, visiting the page again (or refreshing without resending POST data) should put you back to the login form, as if you are not logged in. This is fine for now. We will fix this when we deal with session variables.

Exercise 4: Remembering things

This exercise is a continuation of the previous two, and deals specifically with sessions and cookies. Recall the differences between the two, including where the information is stored and how long the information persists. To refresh yourself on these topics, take a look at lecture 9, the slides used in the presentations, or a tutorial on these topics.

In this exercise, start with your code from Exercise 3. We will build upon it.

Part 1: Keep me logged in!

Starting with the login form in Exercise 4, add the following:

- A successful login should set some session variable so that the server knows that the user is logged in. For example, set `$_SESSION['user']` to be TRUE.
- When the page is loaded, check the session variable. If the user is logged in, display the welcome message instead of the login form.
- Add a "Log Out" button to the welcome message that, when clicked, removes the session variable so that the user is logged out. Clicking the button should redirect the user to the same page, which now shows a login form.

This behavior should be consistent with the lifetime of sessions. That is, the user should stay logged in despite refreshing and opening the same page in multiple tabs, but should be logged out once the browser is closed.

Part 2: Remember me!

Add a "Remember me!" checkbox to the login form. If the box is checked and the login is successful, save a cookie that identifies the user to the server. On further visits to the page, the user should appear logged in, even if the browser has been closed.

You may choose a reasonable expiration time for the cookie. Remember also that if the user manually logs out by clicking the "Log Out" button that the cookie should be deleted (set the expiration to be some time in the past).

There are many (wrong) insecure ways to do this, such as simply saving a cookie with the username that will achieve the desired behavior. If you are looking to make the system more secure, I'll refer you to Charles Miller's "Persistent Login Cookie Best Practice" guide here: http://fishbowl.pastiche.org/2004/01/19/persistent_login_cookie_best_practice/.