Web Programming Step by Step
PHP Assignment: NerdLuv

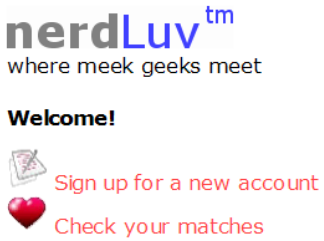
Due March 20th 23:59:59PM.

This assignment is about making a simple multi-page "online dating" site that processes HTML forms with PHP. **Online dating** has become mainstream with popular sites such as eHarmony, Match.com, OkCupid, Chemistry, and Plenty of Fish. Your task for this assignment is to write HTML and PHP code for a fictional online dating site for desperate single geeks, called **NerdLuv**. Turn in the following files:

- signup.php, a page with a form that the user can use to sign up for a new account
- signup-submit.php, the page that receives data submitted by signup.php and signs up the new user
- matches.php, a page with a form for existing users to log in and check their dating matches
- matches-submit.php, the page that receives data submitted by matches.php and show's the user's matches

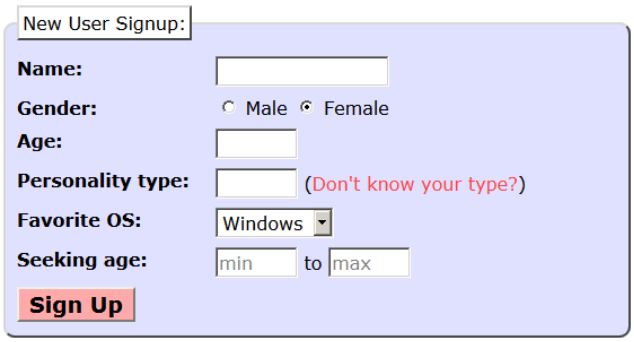
There are some **provided files** on the web site. The first is a complete version of the site's front page, index.php. This front page simply links to your other pages. The other complete provided files are top.html and bottom.html, which contain common header/footer HTML code that you should include in your other pages. We also provide a complete CSS file nerdluv.css with all of the page styles. You should link to this CSS file from all of your pages and use its styles in your code. You should be able to fully style all pages using the styles in nerdluv.css only.

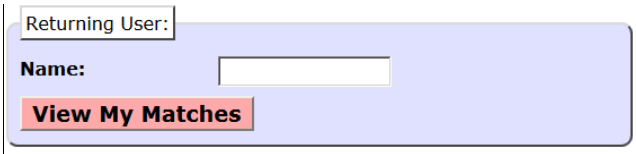
Index Page (index.php) and Overall Site Navigation:



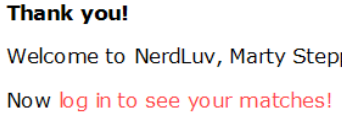
The provided index.php page has a header logo, links to signup.php and matches.php, and footer notes/images. You do not need to modify this file, but you should put it in the same folder with your other files and upload it with your files.

The "Sign Up" link leads to signup.php (left below), and "Check matches" to matches.php (right below):

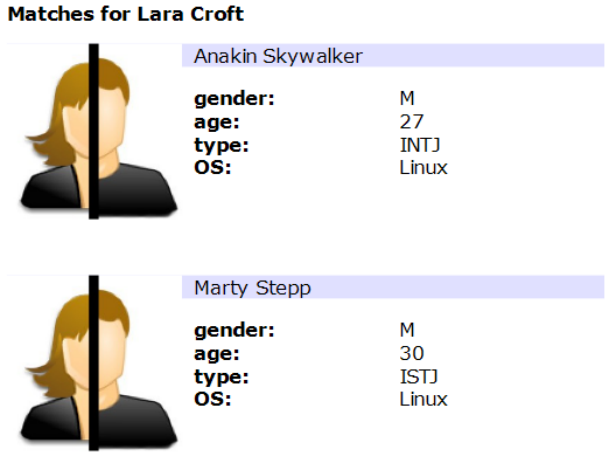




When submitted, the Signup page looks like this:



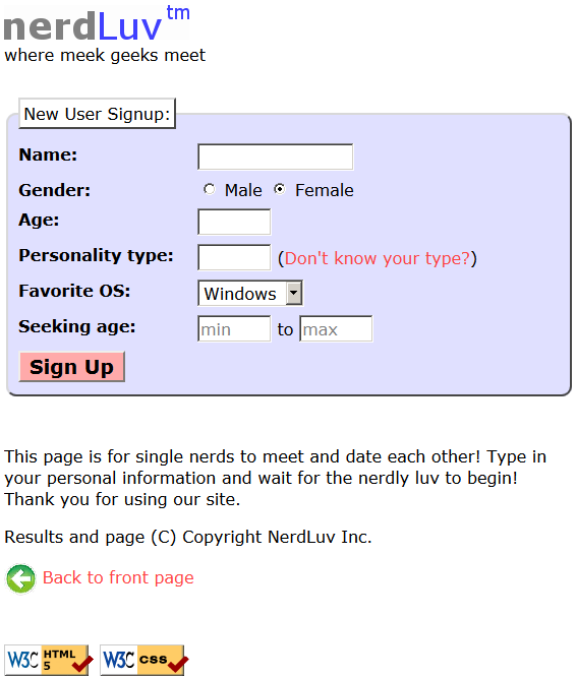
When submitted, the View Matches form looks like this:



The details about each page's contents and behavior are described on the following pages.
Screenshots in this document are from Windows in Firefox, which may differ from your system.

Sign-Up Page (signup.php):

The signup.php page has a header logo, a **form** to create a new account, and footer notes/images. You must write the HTML code for the form. The form contains the following labeled fields:



- **Name:** A 16-character box for the user to type a name. A name should only have alphabetic letters with the first letter of each world capitalized.
- **Gender:** Radio buttons for the user to select a gender of Male or Female. When the user clicks the text next to a radio button, that button should become checked. Initially Female is checked.
- **Age:** A 6-letter-wide text input box for the user to type his/her age in years. The box should allow typing up to 2 characters. The characters must be digits only.
- **Personality type:** A 6-character-wide text box allowing the user to type a Keirsey personality type, such as ISTJ or ENFP. The box should let the user type up to 4 characters. The label has a link to http://www.humanmetrics.com/cgi-win/JTypes2.asp . The input must be one of the 16 types.
- **Favorite OS:** A drop-down select box allowing the user to select a favorite operating system. The choices are Windows, Mac OS X, and Linux. Initially "Windows" is selected.
- **Seeking age:** Two 6-character-wide text boxes for the user to specify the range of acceptable ages of partners. The box should allow the user to type up to 2 characters in each box. Initially both are empty and have placeholder text of "min" and "max" respectively. When the user starts typing, this placeholder text disappears. The input must be digits only.
- **Sign Up:** When pressed, submits the form for processing as described below.

Submitting the Sign-Up Form (signup-submit.php):

When the user presses "Sign Up," the form should **submit** its data as a POST to signup-submit.php. (The exact names and values of the query parameter(s) are up to you.) your PHP code should read the data from the query parameters and store it as described below. The resulting page has the usual header and footer and text thanking the user. The text "log in to see your matches!" links to matches.php.

Your site's user data is stored in a file singles.txt, placed in the same folder as your PHP files. We will provide you an initial version of this file. The file contains data records as lines in *exactly* the following format, with the user's name, gender (M or F), age, personality type, operating system, and min/max seeking age, separated by commas:

Angry Video Game Nerd,M,29,ISTJ,Mac OS X,1,99
Lara Croft,F,23,ENTP,Linux,18,30
Seven of Nine,F,40,ISTJ,Windows,12,50

Your signup-submit.php code should create a line representing the new user's information and add it to the end of the file. See the PHP file_put_contents function in book Chapter 5 or the lecture slides.

In all pages, **validate data** for form submissions. Make sure no field is left blank and all the input conforms to the required format. If there is an error, show the errors on the page by replacing the three lines of text starting at "Thank you!" in the example above. There is no need to check duplicate names.

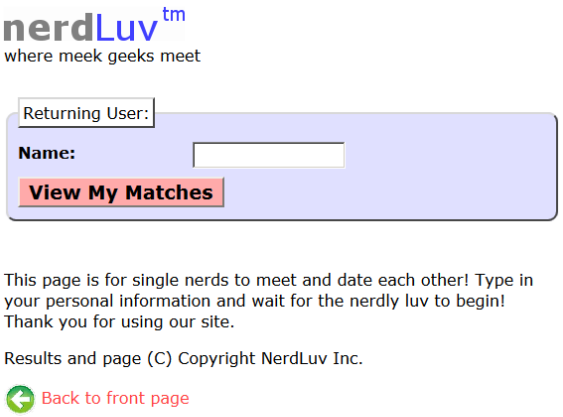
View Matches Page (matches.php):

The matches.php page has a header logo, a **form** to log in and view the user's matches, and footer notes/images. You must write the HTML for the form. The form has one field:

- **Name:** A label and 16-letter box for the user to type a name. Initially empty. Submit to the server as a query parameter name.

When the user presses "View My Matches," the form **submits** its data as a GET request to matches-submit.php. The name of the query parameter sent should be name, and its value should be the encoded text typed by the user. For example, when the user views matches for Rosie O Donnell, the URL should be:

- matches-submit.php?name=Rosie+O+Donnell



Viewing Matches (matches-submit.php):

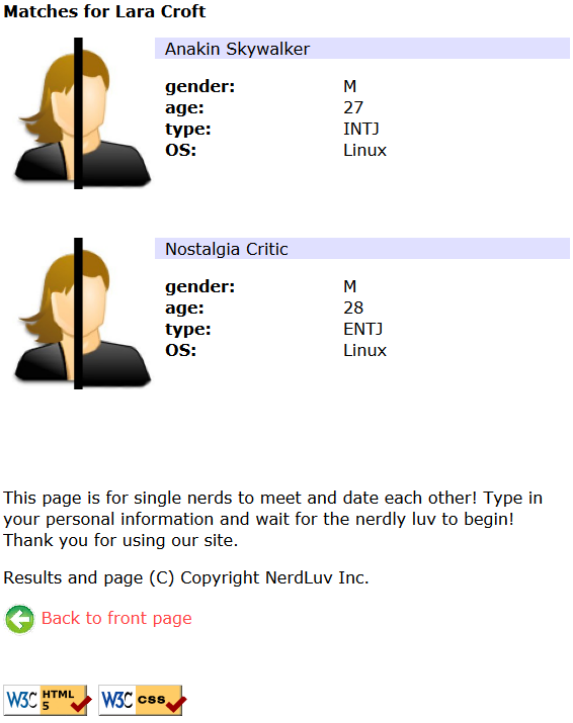
When viewing matches for a given user, matches-submit.php should show a central area displaying each match. Write PHP code that reads the name from the page's name query parameter and finds which other singles match the given user. The existing singles to match against are records found in the file singles.txt as described previously. You may assume that the name parameter is passed and will be found in the file.

Below the banner should be a heading of "Matches for (name)". Below this is a list of singles that match the user. A "match" is a person with **all** of the following qualities:

- The **opposite gender** of the given user;
- Of **compatible ages**; that is, each person is between the other's minimum and maximum ages, inclusive;
- Has the **same favorite operating system** as this user;
- Shares **at least one personality type letter in common** at the same index in each string.
  For example, ISTP and ESFP have 2 in common (S, P).

As you find each match, output the HTML to display the matches, in the order they were originally found in the file. Each match has the image user.jpg, the person's name, and an unordered list with their gender, age, personality type, and OS.

If no match is found, display a message of "No match is found.".



Styling:

The styles you need are already given to you in nerdluv.css, but you still need to use proper tags and class attributes to make sure they are applied. Be mindful of the styles on forms and form controls. In this document several screenshots of the various pages have been shown. Make sure that your form has the same style as in these examples. If you choose the right tags to represent your form, it should match. Make sure that form fields line up in **columns** by using a strong tag or column class so that each text label floats to the left and is 11em wide.

In matches-submit.php, the matches are displayed in a div with class of match. First is a paragraph containing an image of the match, shown with a width of 150px, and the person's name to the right. The paragraph has a light blue background color. The section with the match's gender, age, etc. must be represented as an unordered list (ul).

Uploading and Testing:

Upload all files to your Linux virtual machine to test them. You must change **permissions** on singles.txt so that PHP can write to it. It should be writable by everyone.

Suggested Development Strategy and Hints:

- Based on index.php, write matches.php and matches-submit.php to work properly for existing users.
- Write an **initial version** that outputs *every* person, even ones who aren't compatible "matches." This way you can debug your file I/O, styles, etc. Then add checks like gender, age, and OS. Focus on the PHP code and behavior first, as opposed to style details (CSS is not an emphasis of this assignment).
- Write signup.php and signup-submit.php. If you finish the match page you'll understand forms, making the signup page easier. This is tough; there are more parameters to manage, and you must write to a file.

Use **debug print and print_r statements** to track down bugs. For example, you can print_r($_GET); or $_POST to see the query parameters submitted. Use **Firebug** and also **View Source** to find HTML output problems.

Recall that form controls must have name attributes. Sometimes you must also add a value to affect how data is sent.

Implementation and Grading:

Your HTML output for all pages must pass the W3C **HTML validator**. (Not the PHP source code itself, but the HTML output it generates.) Do not use HTML tables. Since we are using HTML **forms**, choose proper form controls and set their attributes accordingly. Properly choose between GET and POST requests for sending data.

Your PHP code should not cause errors or warnings. Minimize use of the global keyword, use indentation/spacing, and avoid lines over 100 characters. Use material from the first four weeks of class and the first six book chapters.

Some HTML sections are shared redundantly between your PHP pages, found in the provided files top.html and bottom.html. Include these files as appropriate in your other pages using the PHP include function.

A major grading focus is **redundancy**. Use **functions, parameters/return, included files/code**, loops, variables, etc. to avoid redundancy. If you have PHP code you want to share between multiple pages, you may turn in an optional file named common.php containing this code. You can include your common.php in your other pages.

For full credit, reduce the amount of large chunks of PHP code in the middle of HTML code. Replace such chunks with **functions** declared at the top or bottom of your file. You will also lose points if you use PHP print or echo statements. Insert dynamic content into the page using PHP **expression blocks**, <?= ... ?> , as taught in class.

Another grading focus is PHP **commenting**. Put a descriptive comment header at the top of each file, **each function**, and each section of PHP code.

Format your HTML and PHP code similarly to the examples from class. Properly use whitespace and indentation. Do not place more than one block element on a line or begin a block element past the 100th character.

Part of your grade will also come from successfully uploading your files to the web. You should place your files into a hosting server. Some are free, such as https://www.000webhost.com and https://x10hosting.com (use your gmail address to sign up). Your page should be accessible from anywhere.

Submit your assignment online at http://www.networks.howard.edu/lij/courses/submit_hw.html. The following files should be submitted after zipped into a single archive:

- All the HTML and PHP files, CSS files and images with their directory structure preserved.
- A readme.txt file that has
  - The URL of your web page on the hosting server.
  - The URL and summary of five most significant commits
- Any notes that can help you to avoid credit deductions or to earn more credits