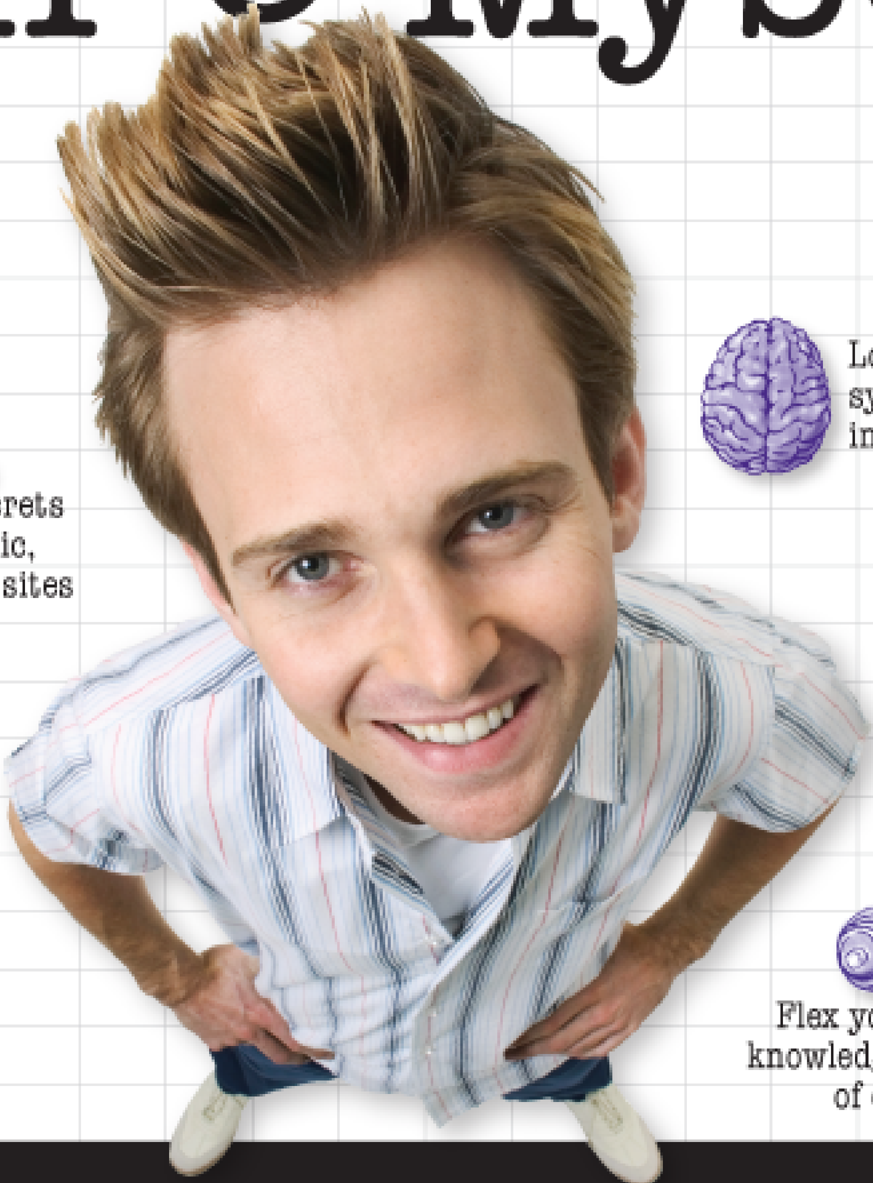
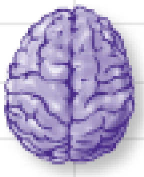
A Brain-Friendly Guide

Head First PHP & MySQL



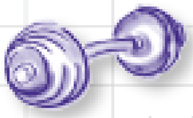
Discover the secrets behind dynamic, database-driven sites



Load all the key syntax directly into your brain

Hook up your PHP and MySQL code

Avoid embarrassing mishaps with web forms



Flex your scripting knowledge with dozens of exercises

O'REILLY®

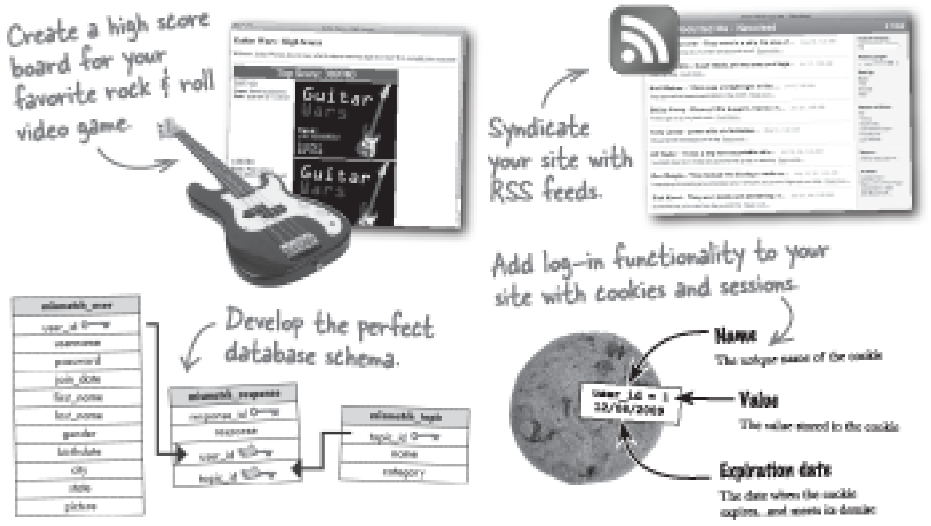
Lynn Beighley & Michael Morrison

Head First PHP & MySQL

Web Programming/PHP

What will you learn from this book?

Ready to take your static HTML web pages to the next level, and build database-driven sites using PHP and MySQL? Then *Head First PHP & MySQL* is your hands-on guide to getting dynamic sites running, fast. Get your hands dirty building real applications, ranging from a video game high-score message board to an online dating site. By the time you're through, you'll be validating forms, working with session IDs and cookies, performing database queries and joins, handling file I/O operations, and more.


Create a high score board for your favorite rock & roll video game.
Guitar Wars
Syndicate your site with RSS feeds.
Add log-in functionality to your site with cookies and sessions.
Develop the perfect database schema.
mismatch_user
mismatch_response
mismatch_topic
Name
The unique name of the cookie
Value
The value stored in the cookie
Expiration date
The date when the cookie expires...and meets its demise
user_id = 1
12/08/2009

Why does this book look so different?

We think your time is too valuable to spend struggling with new concepts. Using the latest research in cognitive science and learning theory to craft a multi-sensory learning experience, *Head First PHP & MySQL* uses a visually rich format designed for the way your brain works, not a text-heavy approach that puts you to sleep.

"PHP and MySQL are two of today's most popular web development technologies, and this book shows readers why. Building a site without them is now as unthinkable as doing web design without CSS. This book is a great introduction and is laugh-out-loud funny. It's the book I wish I had learned from."

—*Harvey Quamen, Associate Professor of English and Humanities Computing, University of Alberta*

"Reading Head First PHP & MySQL is like taking a class from the 'cool' teacher. It makes you look forward to learning."

—*Stephanie Liese, Web Developer*

US $44.99 CAN $44.99
ISBN: 978-0-596-00630-3

54499
9 780596 006303

O'REILLY®
www.oreilly.com
www.headfirstlabs.com

Head First PHP & MySQL

by Lynn Beighley and Michael Morrison

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Series Creators:	Kathy Sierra, Bert Bates
Series Editor:	Brett D. McLaughlin
Editor:	Sanders Kleinfeld
Design Editor:	Louise Barr
Cover Designers:	Louise Barr, Steve Fehler
Production Editor:	Brittany Smith
Proofreader:	Colleen Gorman
Indexer:	Julie Hawks
Page Viewers:	Julien and Drew

Michael's nephew Julien generously lent his Superman powers to help get this book finished.

Printing History:

December 2008: First Edition.

Drew is, at this very moment, installing a new kitchen in Lynn's new old house.

No hardwood floors, UFOs, Elvis look-alikes, or virtual guitars were harmed in the making of this book. But a few broken hearts were mended thanks to some careful mismatching!

ISBN: 978-0-596-00630-3

[M]

7 building personalized web apps

Remember me?

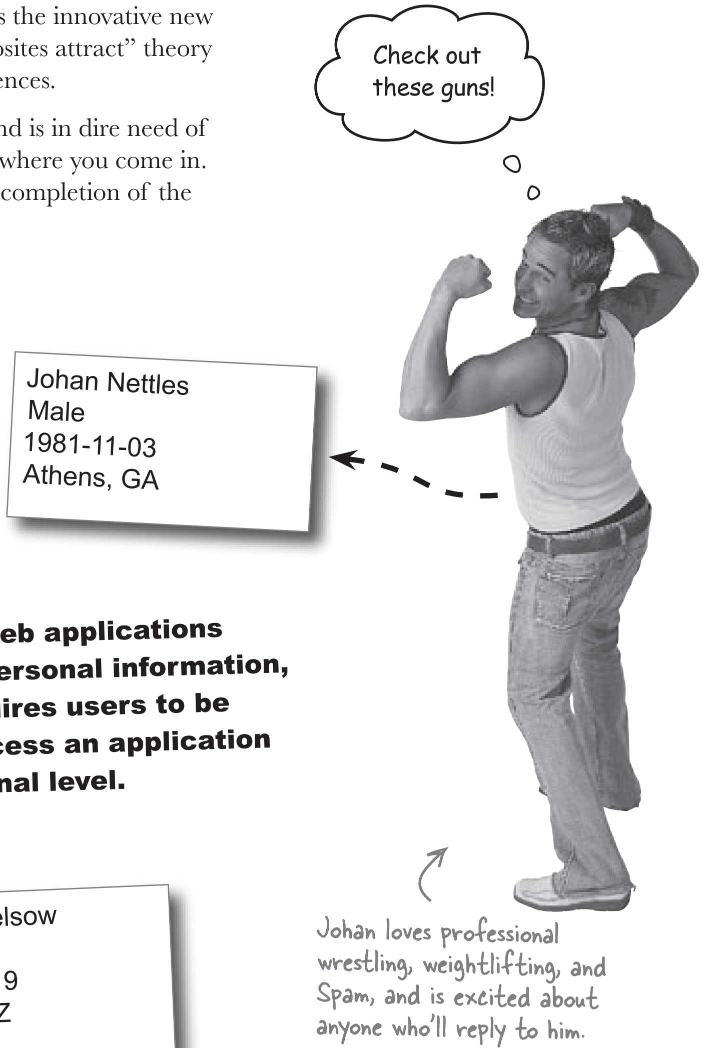
What's your name again? Johnson, right. Well, I'm showing no record of you, Mr. Jackson. Are you sure you signed up for a warranty on your cryogenic storage cell? Oh, I see, so you're calling from inside your cell right now. And what was your name again?

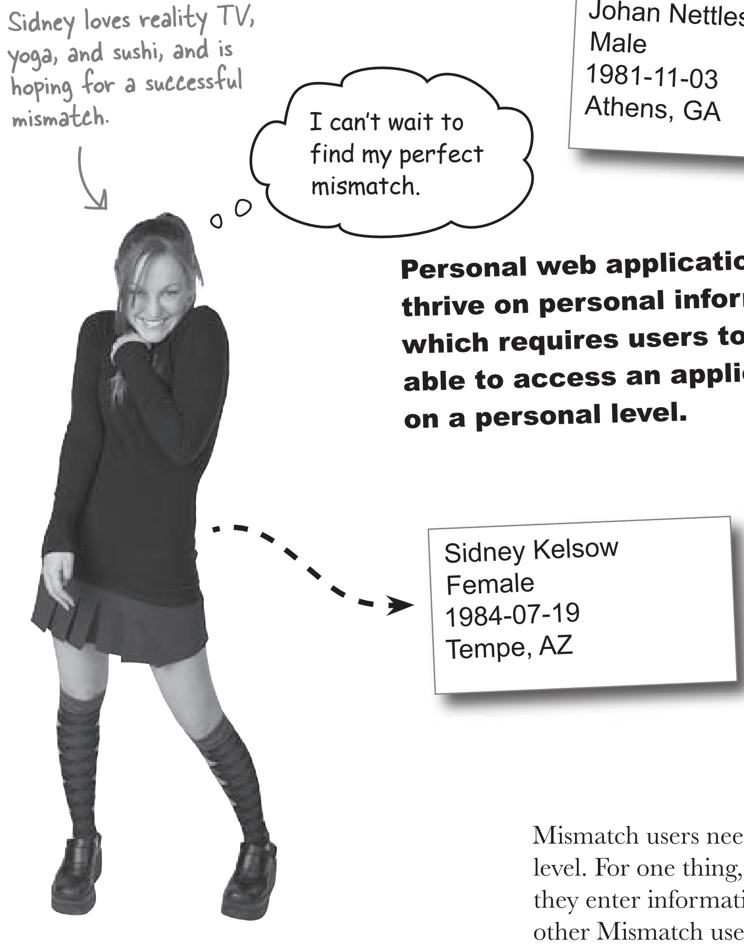
No one likes to be forgotten, especially users of web applications. If an application has any sense of "membership," meaning that users somehow interact with the application in a personal way, then the application needs to remember the users. You'd hate to have to reintroduce yourself to your family every time you walk through the door at home. You don't have to because they have this wonderful thing called **memory**. But *web applications don't remember people automatically*—it's up to a savvy web developer to use the tools at their disposal (PHP and MySQL, maybe?) to **build personalized web apps that can actually remember users**.

They say opposites attract

It's an age-old story: boy meets girl, girl thinks boy is completely nuts, boy thinks girl has issues, but their differences become the attraction, and they end up living happily ever after. This story drives the innovative new dating site, Mis-match.net. Mismatch takes the "opposites attract" theory to heart by mismatching people based on their differences.

Problem is, Mismatch has yet to get off the ground and is in dire need of a web developer to finish building the system. That's where you come in. Millions of lonely hearts are anxiously awaiting your completion of the application... don't let them down!


Check out these guns!
Johan Nettles
Male
1981-11-03
Athens, GA
Johan loves professional wrestling, weightlifting, and Spam, and is excited about anyone who'll reply to him.

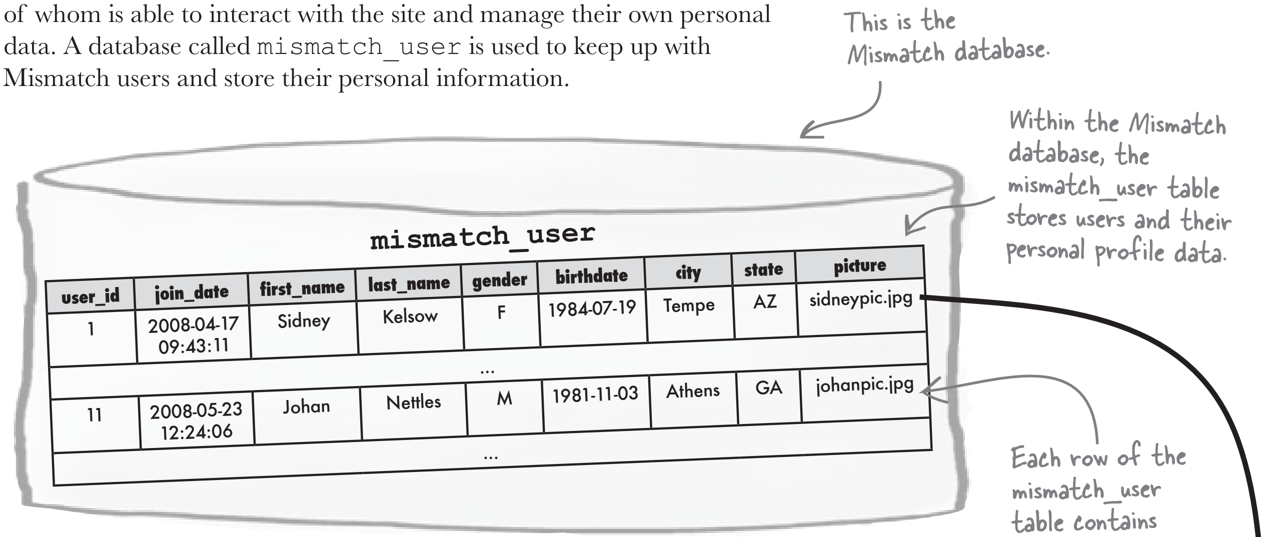

Sidney loves reality TV, yoga, and sushi, and is hoping for a successful mismatch.
I can't wait to find my perfect mismatch.
Sidney Kelsow
Female
1984-07-19
Tempe, AZ

Personal web applications thrive on personal information, which requires users to be able to access an application on a personal level.

Mismatch users need to be able to interact with the site on a personal level. For one thing, this means they need personal profiles where they enter information about themselves that they can share with other Mismatch users, such as their gender, birthdate, and location.

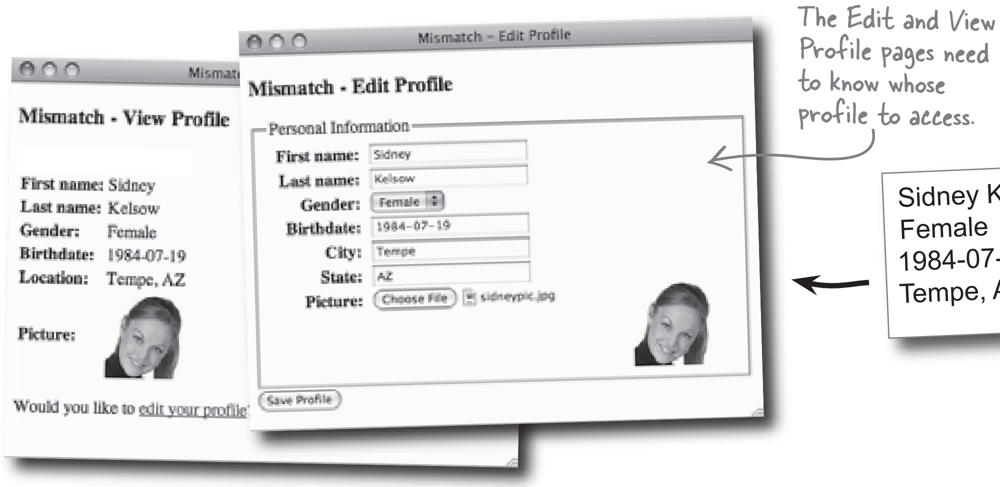
Mismatch is all about personal data

So Mismatch is all about establishing connections through personal data. These connections must take place within a **community of users**, each of whom is able to interact with the site and manage their own personal data. A database called `mismatch_user` is used to keep up with Mismatch users and store their personal information.


This is the Mismatch database.
Within the Mismatch database, the mismatch_user table stores users and their personal profile data.
Each row of the mismatch_user table contains personal data for a single user.

mismatch_user

user_id	join_date	first_name	last_name	gender	birthdate	city	state	picture
1	2008-04-17 09:43:11	Sidney	Kelsow	F	1984-07-19	Tempe	AZ	sidneypic.jpg
...								
11	2008-05-23 12:24:06	Johan	Nettles	M	1981-11-03	Athens	GA	johanpic.jpg
...								


The Edit and View Profile pages need to know whose profile to access.

Mismatch - View Profile

First name: Sidney
Last name: Kelsow
Gender: Female
Birthdate: 1984-07-19
Location: Tempe, AZ
Picture:

Would you like to edit your profile?

Mismatch – Edit Profile

Mismatch - Edit Profile

Personal Information

First name: Sidney
Last name: Kelsow
Gender: Female
Birthdate: 1984-07-19
City: Tempe
State: AZ
Picture: Choose File sidneypic.jpg

Save Profile

Sidney Kelsow
Female
1984-07-19
Tempe, AZ

In addition to viewing a user profile, Mismatch users can edit their own personal profiles using the Edit Profile page. But there's a problem in that the application needs to know which user's profile to edit. The Edit Profile page somehow needs to keep track of the user who is accessing the page.

BRAIN POWER

How can Mismatch customize the Edit Profile page for each different user?

Mismatch needs user log-ins

The solution to the Mismatch personal data access problem involves user log-ins, meaning that users need to be able to log into the application. This gives Mismatch the ability to provide access to information that is custom-tailored to each different user. For example, a logged-in user would only have the ability to edit their own profile data, although they might also be able to view other users' profiles. User log-ins provide the key to personalization for the Mismatch application.

User log-ins allow web applications to get personal with users.

A user log-in typically involves two pieces of information, a username and a password.

Username

The job of the username is to provide each user with a unique name that can be used to identify the user within the system. Users can potentially access and otherwise communicate with each other through their usernames.

`jnettles` `sidneyk`

Usernames typically consist of alphanumeric characters and are entirely up to the user.

Password

The password is responsible for providing a degree of security when logging in users, which helps to safeguard their personal data. To log in, a user must enter both a username and password.

`********` `******`

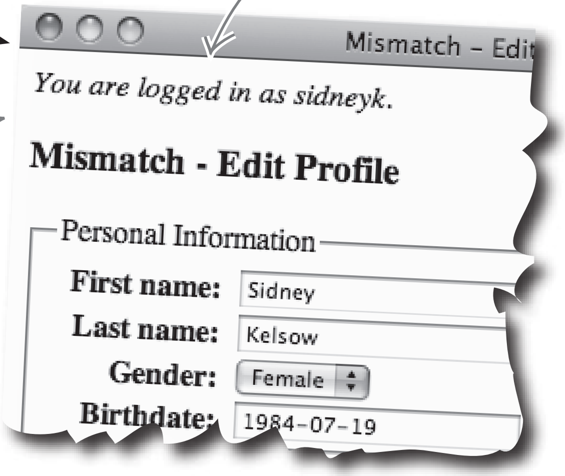
Passwords are extremely sensitive pieces of data and should never be made visible within an application, even inside the database.

A username and password allows a user to log in to the Mismatch application and access personal data, such as editing their profile.

The user's username and password are all that is required to let the application know who they are.

When a user logs in, the application is able to remember the user and provide a personalized experience.

The Edit Profile page now indicates that the user is logged in.


Mismatch – Edit

You are logged in as sidneyk.

Mismatch - Edit Profile

Personal Information

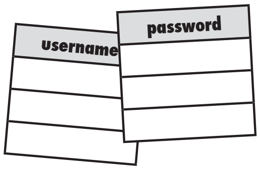
First name:	Sidney
Last name:	Kelsow
Gender:	Female
Birthdate:	1984-07-19

Come up with a user log-in gameplan

Adding user log-in support to Mismatch is no small feat, and it's important to work out exactly what is involved before writing code and running database queries. We know there is an existing table that stores users, so the first thing is to alter it to store log-in data. We'll also need a way for users to enter their log-in data, and this somehow needs to integrate with the rest of the Mismatch application so that pages such as the Edit Profile page are only accessible after a successful log-in. Here are the log-in development steps we've worked out so far:

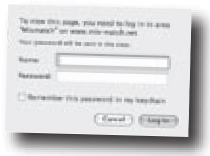
1. **Use `ALTER` to add `username` and `password` columns to the table.**

   The database needs new columns for storing the log-in data for each user. This consists of a username and password.



2. **Build a new Log-In script that prompts the user to enter their username and password.**

   The Log In form is what will ultimately protect personalized pages in that it prompts for a valid username and password. This information must be entered properly before Mismatch can display user-specific data. So the script must limit access to personalized pages so that they can't be viewed without a valid log-in.



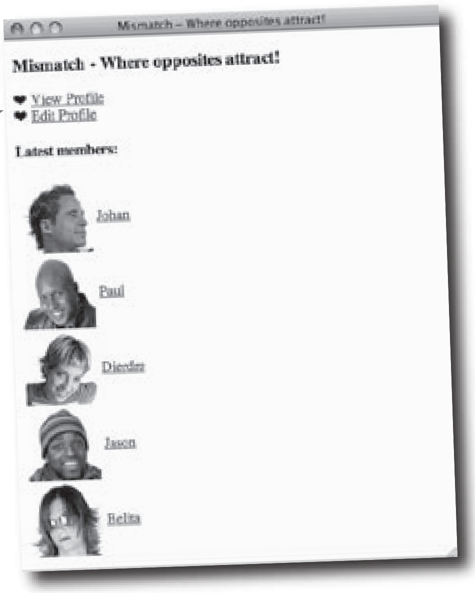
3. **Connect the Log-In script to the rest of the Mismatch application.**

   The Edit Profile and View Profile pages of the Mismatch application should only be accessible to logged in users. So we need to make sure users log in via the Log In script before being allowed to access these pages.

Before going any further, take a moment to tinker with the Mismatch application and get a feel for how it works.

Download all of the code for the Mismatch application from the Head First Labs web site at `www.headfirstlabs.com/books/hfphp`. Post all of the code to your web server except for the `.sql` files, which contain SQL statements that build the necessary Mismatch tables. Make sure to run the statement in each of the `.sql` files in a MySQL tool so that you have the initial Mismatch tables to get started with.

When all that's done, navigate to the `index.php` page in your web browser, and check out the application. Keep in mind that the View Profile and Edit Profile pages are initially broken since they are entirely dependent upon user log-ins, which we're in the midst of building.



Mismatch – Where opposites attract!

Mismatch - Where opposites attract!

❤ View Profile
❤ Edit Profile

Latest members:

Johan
Paul
Dierdre
Jason
Belita

These two links lead into the personalized parts of the application.

The main Mismatch page allows you to see the name and picture of the latest users, but not much else without being logged in.

Download It!

The complete source code for the Mismatch application is available for download from the Head First Labs web site:

`www.headfirstlabs.com/books/hfphp`

Prepping the database for log-ins

OK, back to the construction. The `mismatch_user` table already does a good job of holding profile information for each user, but it's lacking when it comes to user log-in information. More specifically, the table is missing columns for storing a username and password for each user.

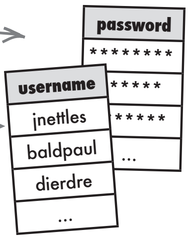
The mismatch_user table needs columns for username and password in order to store user log-in data.

mismatch_user

user_id	join_date	first_name	last_name	gender	birthdate	city	state	picture

Username and password data both consist of pure text, so it's possible to use the familiar `VARCHAR` MySQL data type for the new `username` and `password` columns. However, unlike some other user profile data, the username and password shouldn't ever be allowed to remain empty (`NULL`).

The username and password columns contain simple text data but should never be allowed to go empty.



there are no Dumb Questions

Q: Why can't you just use `user_id` instead of `username` for uniquely identifying a user?

A: You can if you want. In fact, the purpose of `user_id` is to provide an efficient means of uniquely identifying user rows. However, numeric IDs tend to be difficult to remember, and users really like being able to make up their own usernames for accessing personalized web applications. So it's more of a usability decision to allow Johan to be able to log in as "jnettles" instead of "11". No one wants to be relegated to just being a number!

Sharpen your pencil

Finish writing an SQL statement to add the `username` and `password` columns to the table positioned as shown, with `username` able to hold 32 characters, `password` able to hold 16 characters, and neither of them allowing `NULL` data.

Few people would want to try and remember a password longer than 16 characters!

mismatch_user

user_id	username	password	join_date	first_name	last_name	gender	birthdate	city	state	picture

Sharpen your pencil Solution

Finish writing an SQL statement to add the `username` and `password` columns to the table positioned as shown, with `username` able to hold 32 characters, `password` able to hold 16 characters, and neither of them allowing `NULL` data.

ALTER TABLE is used to add new columns to an existing table.

```
ALTER TABLE mismatch_user ADD username VARCHAR(32) NOT NULL AFTER user_id,
ADD password VARCHAR(16) NOT NULL AFTER username
```

The username column is added first, so it's OK to reference it here.

The AFTER statement controls where in the table new columns are added.

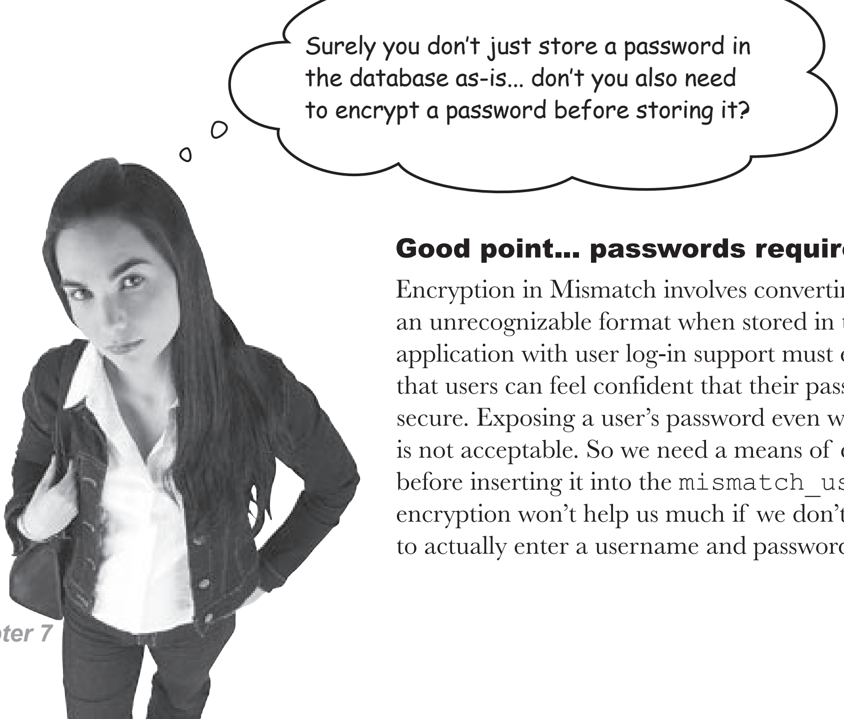
mismatch_user

user_id	username	password	join_date	first_name	last_name	gender	birthdate	city	state	picture

The position of columns in a table doesn't necessarily matter, although it can serve an organizational purpose in terms of positioning the most important columns first.

DONE

1 Use `ALTER` to add `username` and `password` columns to the table.


Surely you don't just store a password in the database as-is... don't you also need to encrypt a password before storing it?

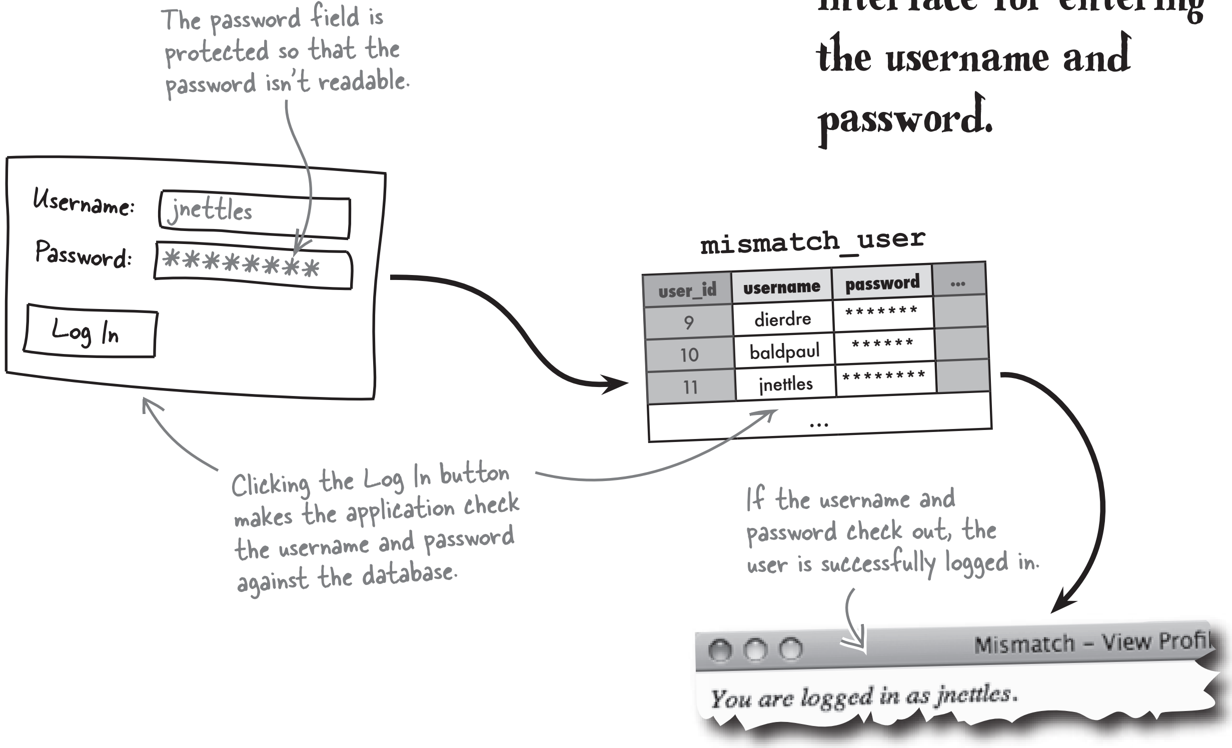
Good point... passwords require encryption.

Encryption in Mismatch involves converting a password into an unrecognizable format when stored in the database. Any application with user log-in support must encrypt passwords so that users can feel confident that their passwords are safe and secure. Exposing a user's password even within the database itself is not acceptable. So we need a means of encrypting a password before inserting it into the `mismatch_user` table. Problem is, encryption won't help us much if we don't have a way for users to actually enter a username and password to log in...

Constructing a log-in user interface

With the database altered to hold user log-in data, we still need a way for users to enter the data and actually log in to the application. This log-in user interface needs to consist of text edit fields for the username and password, as well as a button for carrying out the log-in.

An application log-in requires a user interface for entering the username and password.


The password field is protected so that the password isn't readable.
Username: jnettles
Password: ********
Log In
mismatch_user
user_id | username | password | ...
9 | dierdre | *******
10 | baldpaul | ******
11 | jnettles | ********
...
Clicking the Log In button makes the application check the username and password against the database.
If the username and password check out, the user is successfully logged in.
Mismatch - View Profil
You are logged in as jnettles.

there are no Dumb Questions

Q: So asterisks aren't actually stored in the database, right?

A: That's correct. The asterisks displayed in a password form field simply provide **visual security**, preventing someone from looking over your shoulder as you enter the password. When the form is submitted, the password itself is submitted, not the asterisks. That's why it's important for the password to be encrypted before inserting it into the database.

Relax

If you're worried about how users will be able to log in when we haven't assigned them user names and passwords yet... don't sweat it.

We'll get to creating user names and passwords for users in just a bit. For now it's important to lay the groundwork for log-ins, even if we still have more tasks ahead before it all comes together.

Encrypt passwords with SHA()

The MySQL SHA() function encrypts a piece of text into a unique 40-character code.

The log-in user interface is pretty straightforward, but we didn't address the need to encrypt the log-in password. MySQL offers a function called `SHA()` that applies an encryption algorithm to a string of text. The result is an encrypted string that is exactly 40 hexadecimal characters long, regardless of the original password length. So the function actually generates a 40-character code that uniquely represents the password.

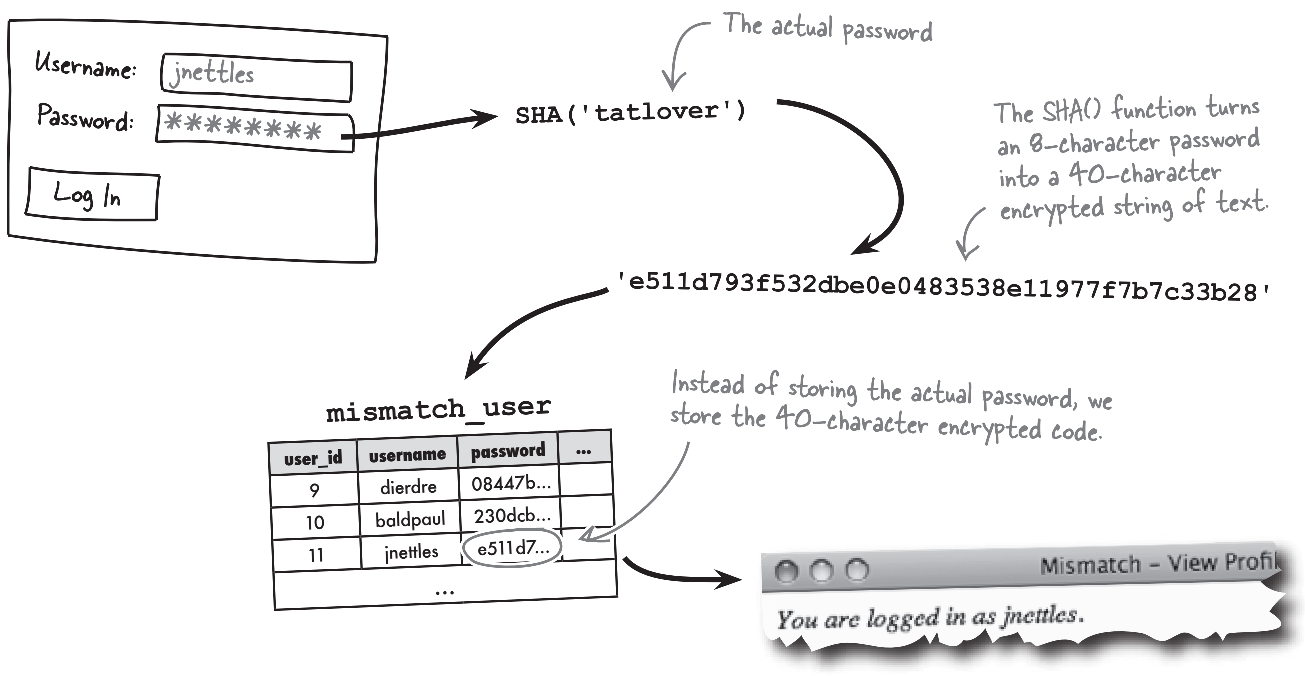
Since `SHA()` is a MySQL function, not a PHP function, you call it as part of the query that inserts a password into a table. For example, this code inserts a new user into the `mismatch_user` table, making sure to encrypt the password with `SHA()` along the way.

```
INSERT INTO mismatch_user
  (username, password, join_date) VALUES ('jnettles', SHA('tatlover'), NOW())
```

The SHA() function encrypts the password into a 40-character hexadecimal code that gets stored in the password column of the mismatch_user table.

This is the actual password as entered into the password form field.

The same `SHA()` function works on the other end of the log-in equation by checking to see that the password entered by the user matches up with the encrypted password stored in the database.



Username: jnettles
Password: ********
Log In
The actual password
SHA('tatlover')
The SHA() function turns an 8-character password into a 40-character encrypted string of text.
'e511d793f532dbe0e0483538e11977f7b7c33b28'
Instead of storing the actual password, we store the 40-character encrypted code.
Mismatch - View Profil

mismatch_user

user_id	username	password	...
9	dierdre	08447b...	
10	baldpaul	230dcb...	
11	jnettles	e511d7...	
...			

You are logged in as jnettles.

Comparing ~~Decrypting~~ passwords

Once you've encrypted a piece of information, the natural instinct is to think in terms of decrypting it at some point. But the `SHA()` function is a one-way encryption with no way back. This is to preserve the security of the encrypted data—even if someone hacked into your database and stole all the passwords, they wouldn't be able to decrypt them. So how is it possible to log in a user if you can't decrypt their password?

You don't need to know a user's original password to know if they've entered the password correctly at log-in. This is because `SHA()` generates the same 40-character code as long as you provide it with the same string of text. So you can just encrypt the log-in password entered by the user and compare it to the value in the `password` column of the `mismatch_user` table. This can be accomplished with a single SQL query that attempts to select a matching user row based on a password.

The SHA() function provides one-way encryption—you can't decrypt data that has been encrypted.

This is the password entered by the user in order to log in.

```
SELECT * FROM mismatch_user
  WHERE password = SHA('tatlover')
```

The SHA() function is called to encrypt the password so that it can appear in the WHERE clause.

This `SELECT` query selects all rows in the `mismatch_user` table whose `password` column matches the entered password, `'tatlover'` in this case. Since we're comparing encrypted versions of the password, it isn't necessary to know the original password. A query to actually log in a user would use `SHA()`, but it would also need to `SELECT` on the user ID, as we see in just a moment.

Making room for the encrypted password

The `SHA()` function presents a problem for Mismatch since encrypted passwords end up being 40 characters long, but our newly created `password` column is only 16 characters long. An `ALTER` is in order to expand the `password` column for storing encrypted passwords.

```
ALTER TABLE mismatch_user
  CHANGE password password VARCHAR(40) NOT NULL
```

The size of the password column is changed to 40 so that encrypted passwords will fit.

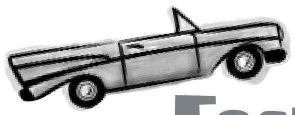
there are no Dumb Questions

Q: What does SHA() stand for?

A: The `SHA()` function stands for Secure Hash Algorithm. A "hash" is a programming term that refers to a unique, fixed-length string that uniquely represents a string of text. In the case of `SHA()`, the hash is the 40-character hexadecimal encrypted string of text, which uniquely represents the original password.

Q: Are there any other ways to encrypt passwords?

A: Yes. MySQL offers another function similar to `SHA()` called `MD5()` that carries out a similar type of encryption. But the `SHA()` algorithm is considered a little more secure than `MD5()`, so it's better to use `SHA()` instead. PHP also offers equivalent functions (`sha1()` and `md5()`) if you need to do any encryption in PHP code, as opposed to within an SQL query.



Test Drive

Add the `username` and `password` columns to the `mismatch_user` table, and then try them out.

Using a MySQL tool, execute the ALTER statement to add the username and password columns to the mismatch_user table.

```
ALTER TABLE mismatch_user ADD username VARCHAR(32) NOT NULL AFTER user_id,
  ADD password VARCHAR(16) NOT NULL AFTER username
```

But our password column actually needs to be able to hold a 40-character encrypted string, so ALTER the table once more to make room for the larger password data.

```
ALTER TABLE mismatch_user
  CHANGE password password VARCHAR(40) NOT NULL
```

Now, to test out the new columns, let's do an INSERT for a new user.

Don't forget to encrypt the password by calling the SHA() function.

```
INSERT INTO mismatch_user
  (username, password, join_date) VALUES ('jimi', SHA('heyjoe'), NOW())
```

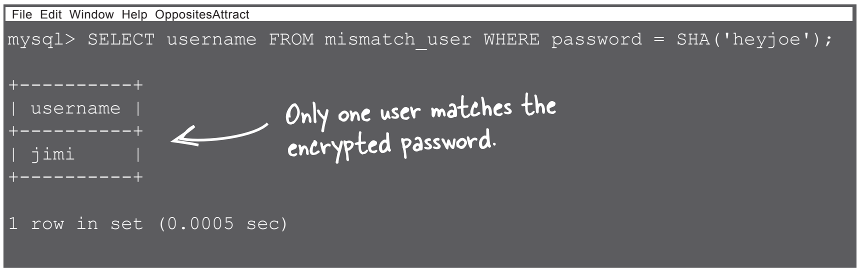
To double-check that the password was indeed encrypted in the database, take a look at it by running a SELECT on the new user.

```
SELECT password FROM mismatch_user WHERE username = 'jimi'
```

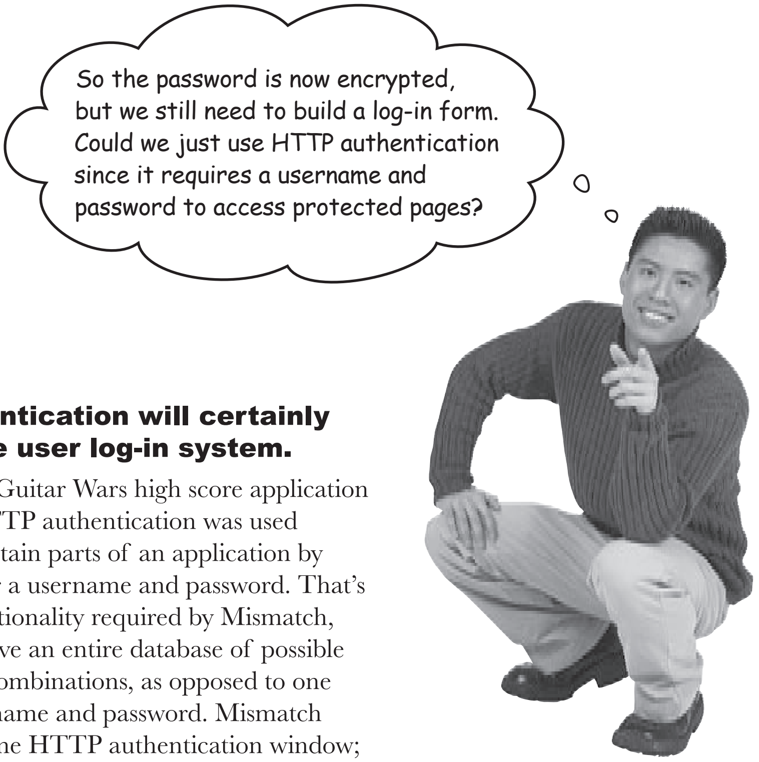
And finally, you can simulate a log-in check by doing a SELECT on the username and using the SHA() function with the password in a WHERE clause.

For a successful log-in, this must be the same password used when inserting the row.

```
SELECT username FROM mismatch_user WHERE password = SHA('heyjoe')
```

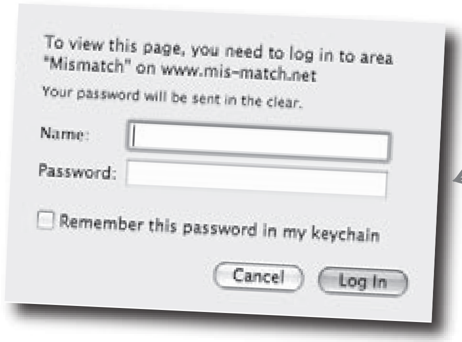

File Edit Window Help OppositesAttract
Only one user matches the encrypted password.

```
mysql> SELECT username FROM mismatch_user WHERE password = SHA('heyjoe');

+----------+
| username |
+----------+
| jimi     |
+----------+

1 row in set (0.0005 sec)
```


So the password is now encrypted, but we still need to build a log-in form. Could we just use HTTP authentication since it requires a username and password to access protected pages?

Yes! HTTP authentication will certainly work as a simple user log-in system.

If you recall from the Guitar Wars high score application in the last chapter, HTTP authentication was used to restrict access to certain parts of an application by prompting the user for a username and password. That's roughly the same functionality required by Mismatch, except that now we have an entire database of possible username/password combinations, as opposed to one application-wide username and password. Mismatch users could use the same HTTP authentication window; however, they'll just be entering their own personal username and password.


To view this page, you need to log in to area "Mismatch" on www.mis-match.net

Your password will be sent in the clear.

Name:

Password:

The standard HTTP authentication window, which is browser-specific, can serve as a simple log-in user interface.

Authorizing users with HTTP

As Guitar Wars illustrated, two headers must be sent in order to restrict access to a page via an HTTP authentication window. These headers result in the user being prompted for a username and password in order to gain access to the Admin page of Guitar Wars.

```
HTTP/1.1 401 Unauthorized
```

```
WWW-Authenticate: Basic realm="Guitar Wars"
```

These two headers must be sent in order to restrict access to a page via HTTP authentication.

Sending the headers for HTTP authentication amounts to two lines of PHP code—a call to the `header()` function for each header being sent.

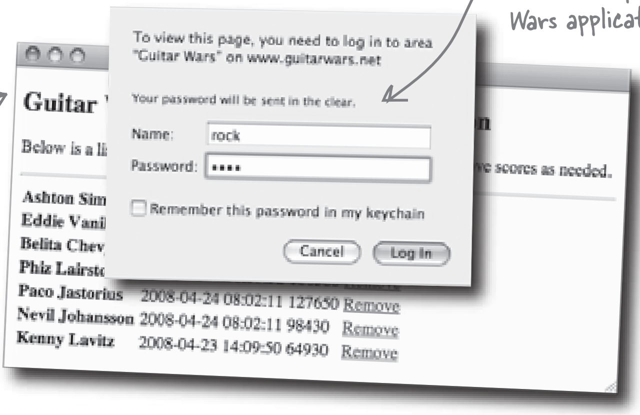
```
header('HTTP/1.1 401 Unauthorized');
header('WWW-Authenticate: Basic realm="Mismatch"');
```

HTTP authentication requires us to send two headers.

This is the realm for the authentication, which applies to the entire application.

Unless a user enters the correct username and password, they cannot see or use this page.

A username and password are required in order to access restricted pages in the Guitar Wars application.


To view this page, you need to log in to area "Guitar Wars" on www.guitarwars.net

Your password will be sent in the clear.

Name: rock

Password: ••••

☐ Remember this password in my keychain

Cancel | Log In

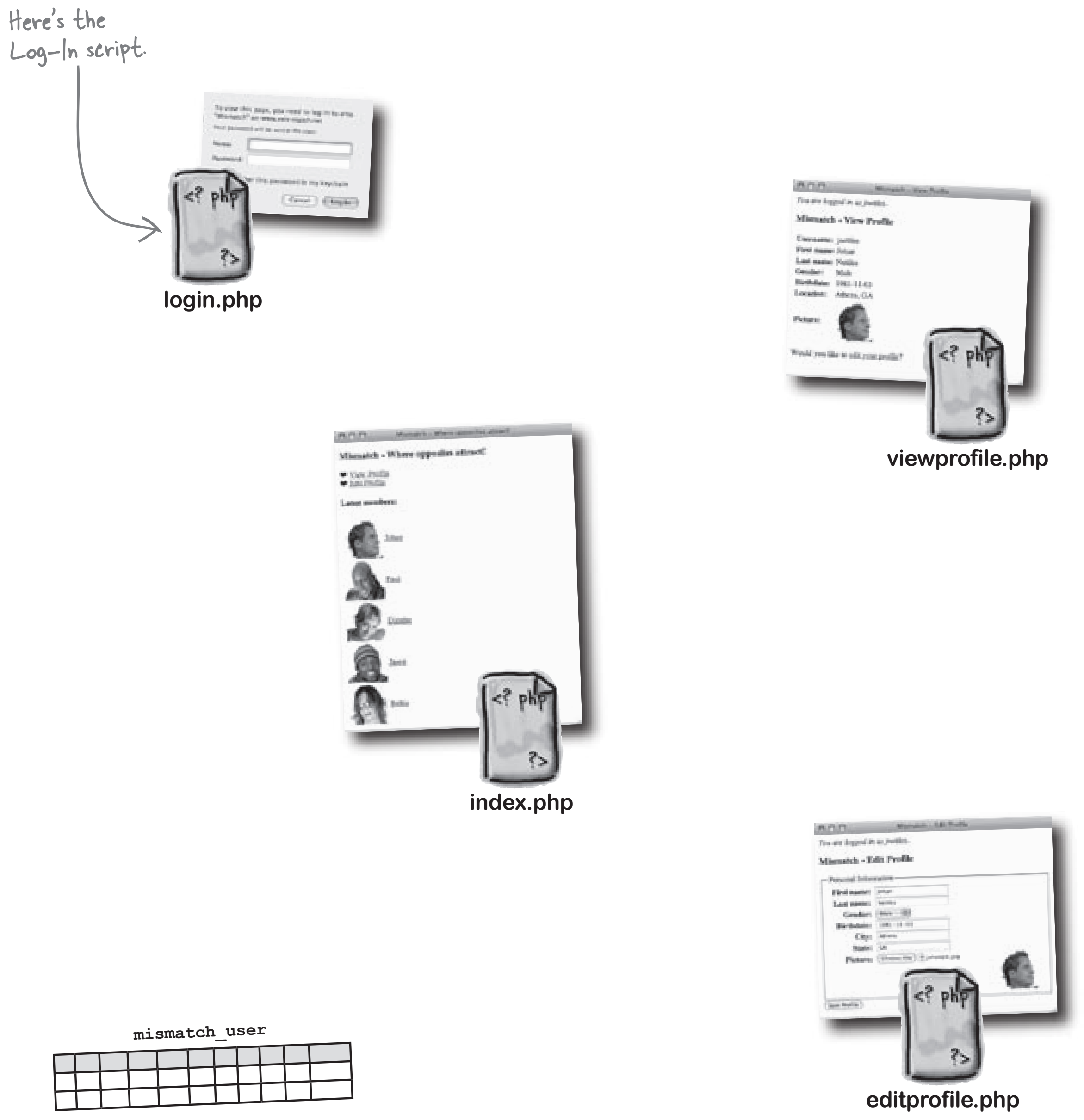
Guitar

Below is a li … ve scores as needed.

Ashton Sim
Eddie Vanil
Belita Chev
Phiz Lairsto
Paco Jastorius 2008-04-24 08:02:11 127650 Remove
Nevil Johansson 2008-04-24 08:02:11 98430 Remove
Kenny Lavitz 2008-04-23 14:09:50 64930 Remove

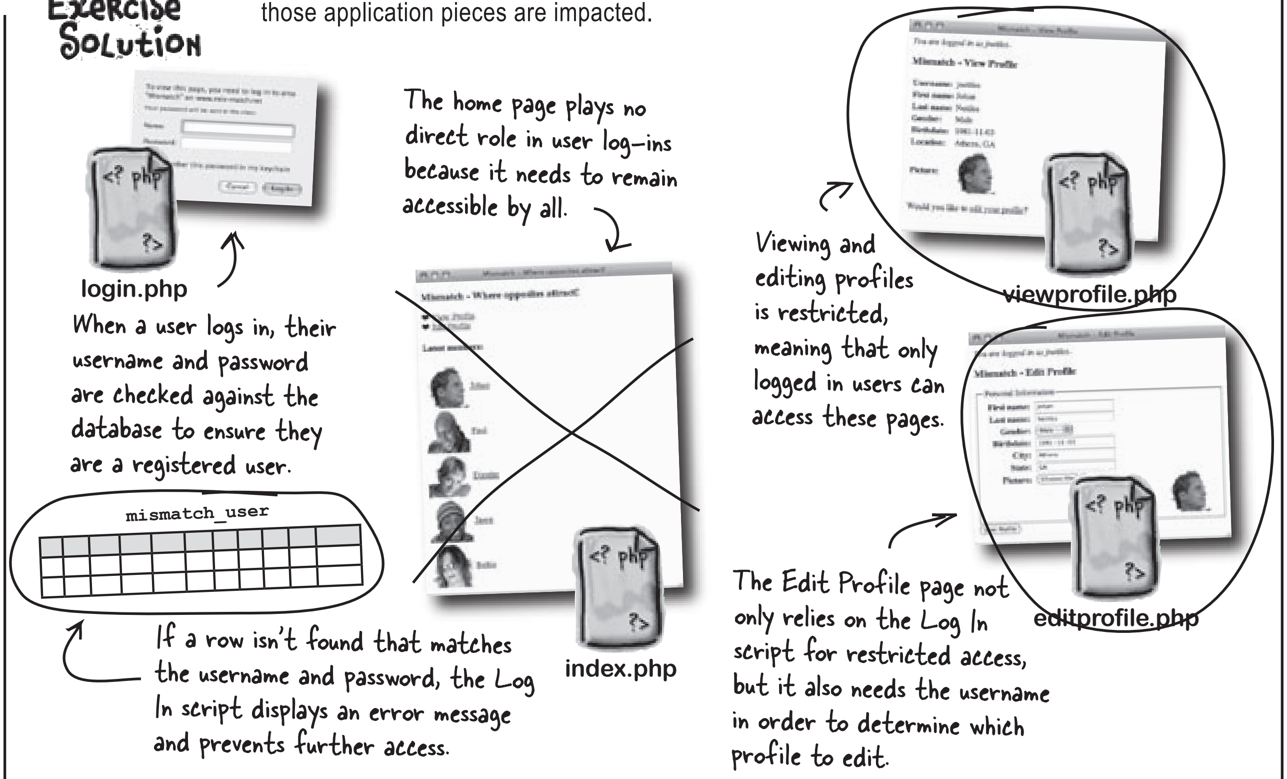
Exercise

Circle the different parts of the Mismatch application that are impacted by the Log-In script (`login.php`) and its usage of HTTP authentication to control access. Then annotate how those application pieces are impacted.



Exercise Solution

Circle the different parts of the Mismatch application that are impacted by the Log-In script (`login.php`) and its usage of HTTP authentication to control access. Then annotate how those application pieces are impacted.


login.php
When a user logs in, their username and password are checked against the database to ensure they are a registered user.
mismatch_user
If a row isn't found that matches the username and password, the Log In script displays an error message and prevents further access.
The home page plays no direct role in user log-ins because it needs to remain accessible by all.
index.php
Viewing and editing profiles is restricted, meaning that only logged in users can access these pages.
viewprofile.php
editprofile.php
The Edit Profile page not only relies on the Log In script for restricted access, but it also needs the username in order to determine which profile to edit.

there are no Dumb Questions

Q: Why isn't it necessary to include the home page when requiring user log-ins?

A: Because the home page is the first place a user lands when visiting the site, and it's important to let visitors glimpse the site before requiring a log-in. So the home page serves as both a teaser and a starting point—a teaser for visitors and a starting point for existing users who must log in to go any deeper into the application.

Q: Can logged-in users view anyone's profile?

A: Yes. The idea is that profiles are visible to all users who log in, but remain private to guests. In other words, you have to be a member of Mismatch in order to view another user's profile.

Q: How does password encryption affect HTTP authentication?

A: There are two different issues here: transmitting a password and storing a password. The `SHA()` MySQL function focuses on securely **storing** a password in a database in an encrypted form. The database doesn't care how you transmitted the password initially, so this form of encryption has no impact on HTTP authentication. However, an argument could be made that encryption should also take place during the **transmission** of the password when the HTTP authentication window submits it to the server. This kind of encryption is outside the scope of this chapter and, ultimately, only necessary when dealing with highly sensitive data.

Logging In Users with HTTP Authentication

The Log-In script (`login.php`) is responsible for requesting a username and password from the user using HTTP authentication headers, grabbing the username and password values from the `$_SERVER` superglobal, and then checking them against the `mismatch_user` database before providing access to a restricted page.

```
<?php
  require_once('connectvars.php');

  if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])) {
    // The username/password weren't entered so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h3>Mismatch</h3>Sorry, you must enter your username and password to log in and access ' .
      'this page.');
  }

  // Connect to the database
  $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

  // Grab the user-entered log-in data
  $user_username = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_USER']));
  $user_password = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_PW']));

  // Look up the username and password in the database
  $query = "SELECT user_id, username FROM mismatch_user WHERE username = '$user_username' AND " .
    "password = SHA('$user_password')";
  $data = mysqli_query($dbc, $query);

  if (mysqli_num_rows($data) == 1) {
    // The log-in is OK so set the user ID and username variables
    $row = mysqli_fetch_array($data);
    $user_id = $row['user_id'];
    $username = $row['username'];
  }
  else {
    // The username/password are incorrect so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h2>Mismatch</h2>Sorry, you must enter a valid username and password to log in and ' .
      'access this page.');
  }

  // Confirm the successful log-in
  echo('<p class="login">You are logged in as ' . $username . '.</p>');
?>
```

If the username and password haven't been entered, send the authentication headers to prompt the user.

Grab the username and password entered by the user.

Perform a query to see if any rows match the username and encrypted password.

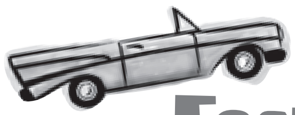
If a row matches, it means the log-in is OK, and we can set the $user_id and $username variables.

If no database row matches the username and password, send the authentication headers again to re-prompt the user.

All is well at this point, so confirm the successful log-in.

DONE

2 Build a new Log-In script that prompts the user to enter their username and password.



Test Drive

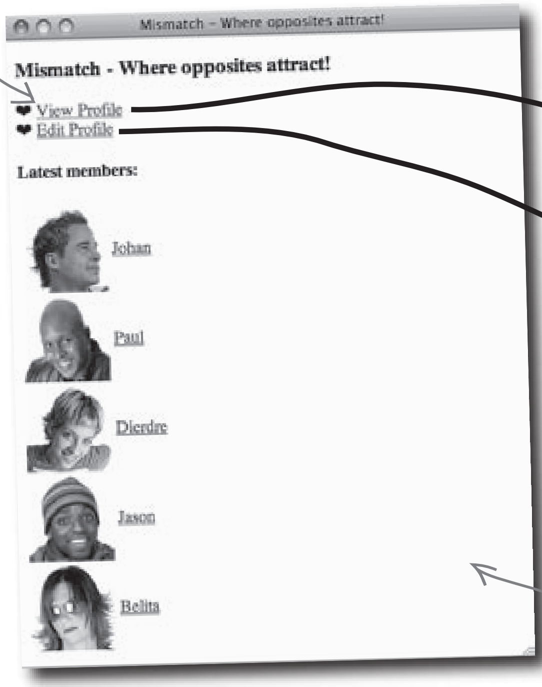
Create the new Log-In script, and include it in the View Profile and Edit Profile scripts.

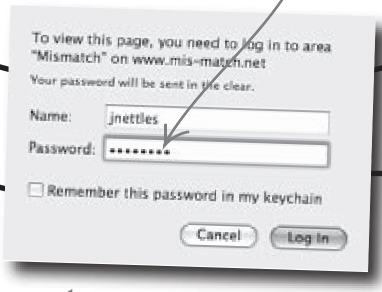
Create a new text file named `login.php`, and enter the code for the Log-In script in it (or download the script from the Head First Labs site at `www.headfirstlabs.com/books/hfphp`). Then add PHP code to the top of the `viewprofile.php` and `editprofile.php` scripts to include the new Log-In script.

Upload all of the scripts to your web server, and then open the main Mismatch page in a web browser. Click the View Profile or Edit Profile link to log in and access the personalized pages. Of course, this will only work if you've already added a user with a username and password to the database.

These two links lead to the protected pages, which invoke the Log-In script if a user isn't logged in.

This password is SHA() encrypted and compared with the password in the database to determine if the log-in is allowed.


Mismatch - Where opposites attract!
Mismatch - Where opposites attract!
View Profile
Edit Profile
Latest members:
Johan
Paul
Dierdre
Jason
Belita



To view this page, you need to log in to area "Mismatch" on www.mis-match.net

Your password will be sent in the clear.

Name: jnettles

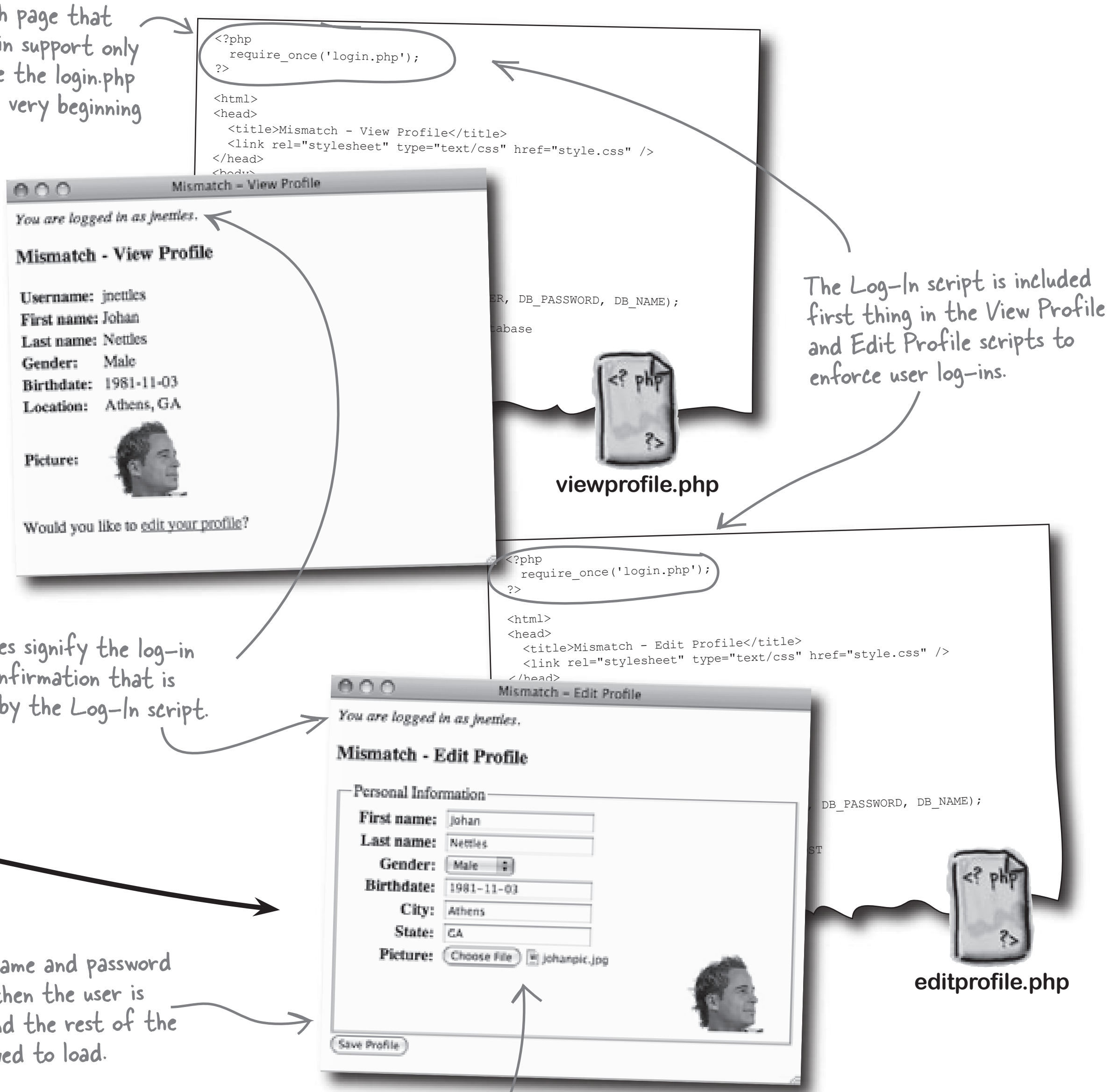
Password: ••••••••

☐ Remember this password in my keychain

The Log-In script uses HTTP authentication to prevent unauthorized access to the View Profile and Edit Profile pages.

The home page is not protected by the Log-In script, but it does serve as the starting point for navigating deeper into the application.

Any Mismatch page that requires log-in support only has to include the login.php script at the very beginning of its code.



```
<?php
  require_once('login.php');
?>

<html>
<head>
  <title>Mismatch - View Profile</title>
  <link rel="stylesheet" type="text/css" href="style.css" />
</head>
```

```
ER, DB_PASSWORD, DB_NAME);

tabase
```

viewprofile.php

Mismatch - View Profile

You are logged in as jnettles.

Mismatch - View Profile

Username:	jnettles
First name:	Johan
Last name:	Nettles
Gender:	Male
Birthdate:	1981-11-03
Location:	Athens, GA
Picture:	

Would you like to edit your profile?

The Log-In script is included first thing in the View Profile and Edit Profile scripts to enforce user log-ins.

```
<?php
  require_once('login.php');
?>

<html>
<head>
  <title>Mismatch - Edit Profile</title>
  <link rel="stylesheet" type="text/css" href="style.css" />
</head>
```

```
, DB_PASSWORD, DB_NAME);

ST
```

editprofile.php

Both pages signify the log-in with a confirmation that is provided by the Log-In script.

Mismatch - Edit Profile

You are logged in as jnettles.

Mismatch - Edit Profile

Personal Information

First name:	Johan
Last name:	Nettles
Gender:	Male
Birthdate:	1981-11-03
City:	Athens
State:	GA
Picture:	Choose File johanpic.jpg

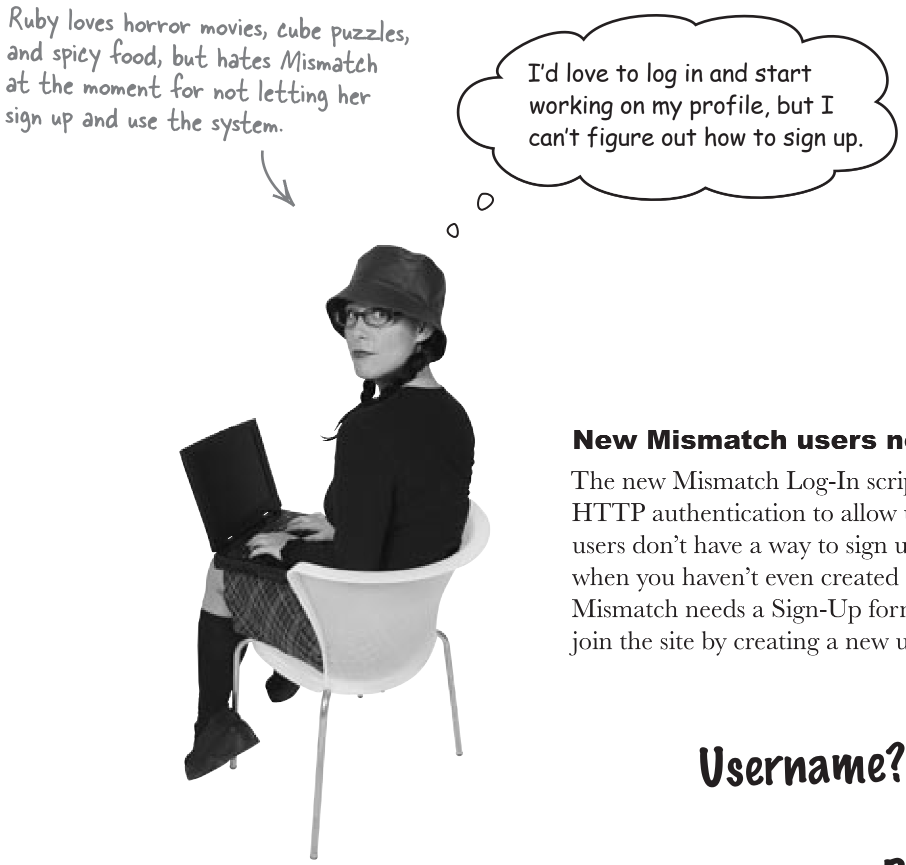
Save Profile

If the username and password check out, then the user is logged in, and the rest of the page is allowed to load.

Each user is now presented with their very own customized Mismatch experience.

DONE

3 **Connect the Log-In script to the rest of the Mismatch application.**


Ruby loves horror movies, cube puzzles, and spicy food, but hates Mismatch at the moment for not letting her sign up and use the system.
I'd love to log in and start working on my profile, but I can't figure out how to sign up.

New Mismatch users need a way to sign up.

The new Mismatch Log-In script does a good job of using HTTP authentication to allow users to log in. Problem is, users don't have a way to sign up—logging in is a problem when you haven't even created a username or password yet. Mismatch needs a Sign-Up form that allows new users to join the site by creating a new username and password.

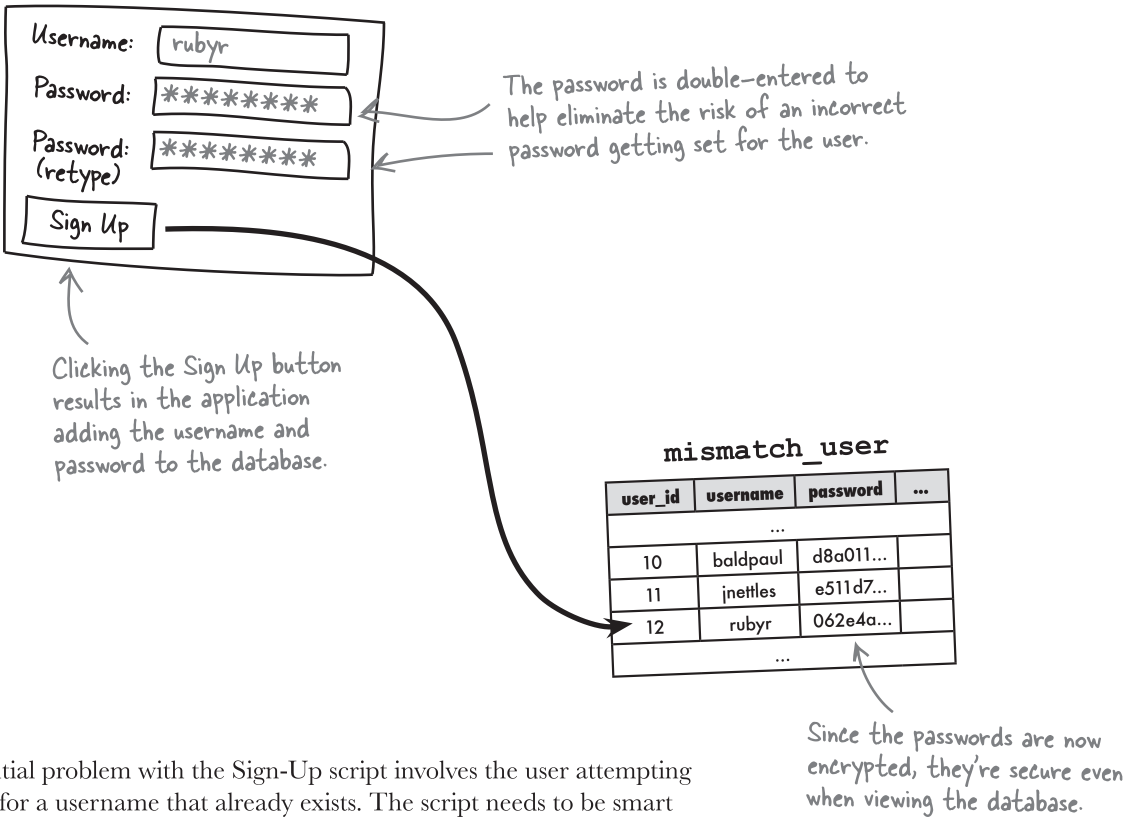
Username?

Password?

A form for signing up new users

What does this new Sign-Up form look like? We know it needs to allow the user to enter their desired username and password... anything else? Since the user is establishing their password with the new Sign-Up form, and passwords in web forms are typically masked with asterisks for security purposes, it's a good idea to have two password form fields. So the user enters the password twice, just to make sure there wasn't a typo.

So the job of the Sign-Up page is to retrieve the username and password from the user, make sure the username isn't already used by someone else, and then add the new user to the `mismatch_user` database.


Username: rubyr
Password: ********
Password: (retype) ********
Sign Up
The password is double-entered to help eliminate the risk of an incorrect password getting set for the user.
Clicking the Sign Up button results in the application adding the username and password to the database.
mismatch_user
Since the passwords are now encrypted, they're secure even when viewing the database.

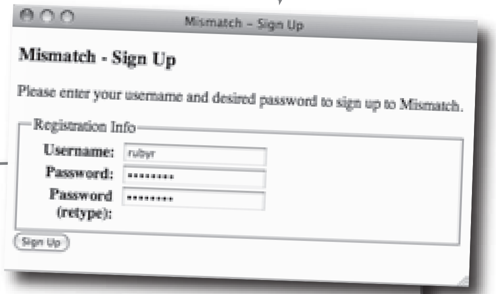
user_id	username	password	...
...			
10	baldpaul	d8a011...	
11	jnettles	e511d7...	
12	rubyr	062e4a...	
...			

One potential problem with the Sign-Up script involves the user attempting to sign up for a username that already exists. The script needs to be smart enough to catch this problem and force the user to try a different username.
So the job of the Sign-Up page is to retrieve the username and password from the user, make sure the username isn't already used by someone else, and then add the new user to the `mismatch_user` database.

PHP & MySQL Magnets

The Mismatch Sign-Up script uses a custom form to prompt the user for their desired username and password. Problem is, the script code is incomplete. Use the magnets below to finish up the script so new users can sign up and join the Mismatch community.

Here's the Sign-Up form.



Mismatch - Sign Up

Please enter your username and desired password to sign up to Mismatch.

Registration Info

Username:	rubyr
Password:	••••••••
Password (retype):	••••••••

```
<?php
 require_once('appvars.php');
 require_once('connectvars.php');

 // Connect to the database
 $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

 if (isset($_POST['submit'])) {
  // Grab the profile data from the POST
 .................... = mysqli_real_escape_string($dbc, trim($_POST['....................']));
 .................... = mysqli_real_escape_string($dbc, trim($_POST['....................']));
 .................... = mysqli_real_escape_string($dbc, trim($_POST['....................']));

   if (!empty($username) && !empty($password1) && !empty($password2) &&
    (.................... == ....................)) {
    // Make sure someone isn't already registered using this username
    $query = "SELECT * FROM mismatch_user WHERE username = '....................'";
    $data = mysqli_query($dbc, $query);
    if (mysqli_num_rows($data) == 0) {
     // The username is unique, so insert the data into the database
     $query = "INSERT INTO mismatch_user (username, password, join_date) VALUES " .
      "('....................', SHA('....................'), NOW())";
     mysqli_query($dbc, $query);

     // Confirm success with the user
     echo '<p>Your new account has been successfully created. You\'re now ready to log in and ' .
      '<a href="editprofile.php">edit your profile</a>.</p>';

     mysqli_close($dbc);
     exit();
    }
```

Don't forget, you have to escape an apostrophe if it appears inside of single quotes.

```
  else {
    // An account already exists for this username, so display an error message
    echo '<p class="error">An account already exists for this username. Please use a different ' .
      'address.</p>';

      ................... = "";

    }
  }
  else {
    echo '<p class="error">You must enter all of the sign-up data, including the desired password ' .
      'twice.</p>';
  }
}

mysqli_close($dbc);
?>

<p>Please enter your username and desired password to sign up to Mismatch.</p>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
  <fieldset>
    <legend>Registration Info</legend>
    <label for="username">Username:</label>

    <input type="text" id="..................." name="..................."

      value="<?php if (!empty(...................)) echo ...................; ?>" /><br />

    <label for="...................">Password:</label>

    <input type="..................." id="..................." name="..................." /><br />

    <label for="...................">Password (retype):</label>

    <input type="..................." id="..................." name="..................." /><br />

  </fieldset>
  <input type="submit" value="Sign Up" name="submit" />
</form>
```

signup.php

password
password

$password1
$password1
$password1

$username
$username
$username
$username
$username
$username

username
username
username

password2
password2
password2
password2

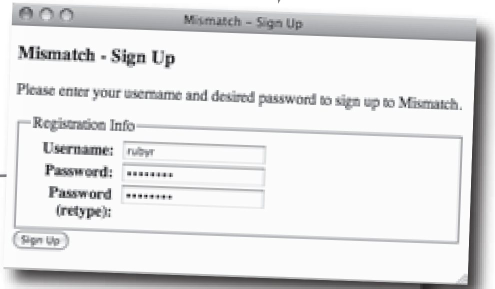
password1
password1
password1
password1

$password2
$password2

PHP & MySQL Magnets Solution

The Mismatch Sign-Up script uses a custom form to prompt the user for their desired username and password. Problem is, the script code is incomplete. Use the magnets below to finish up the script so new users can sign up and join the Mismatch community.

Here's the Sign-Up form.


Mismatch - Sign Up

Please enter your username and desired password to sign up to Mismatch.

Registration Info

Username: rubyr

Password: ••••••••

Password (retype): ••••••••

```
<?php
 require_once('appvars.php');
 require_once('connectvars.php');

 // Connect to the database
 $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

 if (isset($_POST['submit'])) {
  // Grab the profile data from the POST
  $username = mysqli_real_escape_string($dbc, trim($_POST['username']));
  $password1 = mysqli_real_escape_string($dbc, trim($_POST['password1']));
  $password2 = mysqli_real_escape_string($dbc, trim($_POST['password2']));

  if (!empty($username) && !empty($password1) && !empty($password2) &&
   ($password1 == $password2)) {
   // Make sure someone isn't already registered using this username
   $query = "SELECT * FROM mismatch_user WHERE username = '$username'";
   $data = mysqli_query($dbc, $query);
   if (mysqli_num_rows($data) == 0) {
    // The username is unique, so insert the data into the database
    $query = "INSERT INTO mismatch_user (username, password, join_date) VALUES " .
     "('$username', SHA('$password1'), NOW())";
    mysqli_query($dbc, $query);

    // Confirm success with the user
    echo '<p>Your new account has been successfully created. You\'re now ready to log in and ' .
     '<a href="editprofile.php">edit your profile</a>.</p>';

    mysqli_close($dbc);
    exit();
   }
```

Grab all of the user-entered data, making sure to clean it up first.

Check to make sure that none of the form fields are empty and that both passwords match.

Perform a query to see if any existing rows match the username entered.

If no match is found, the username is unique, so we can carry out the INSERT.

Either password could be used here since they must be equal to get to this point.

Confirm the successful sign-up with the user, and exit the script.

```
    else {
      // An account already exists for this username, so display an error message
      echo '<p class="error">An account already exists for this username. Please use a different ' .
        'address.</p>';

        $username = "";
    }
  }
  else {
    echo '<p class="error">You must enter all of the sign-up data, including the desired password ' .
      'twice.</p>';
  }
}

mysqli_close($dbc);
?>

<p>Please enter your username and desired password to sign up to Mismatch.</p>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
  <fieldset>
    <legend>Registration Info</legend>
    <label for="username">Username:</label>
    <input type="text" id="username" name="username"
      value="<?php if (!empty($username)) echo $username; ?>" /><br />
    <label for="password1">Password:</label>
    <input type="password" id="password1" name="password1" /><br />
    <label for="password2">Password (retype):</label>
    <input type="password" id="password2" name="password2" /><br />
  </fieldset>
  <input type="submit" value="Sign Up" name="submit" />
</form>
```

The username is not unique, so display an error message.

Clear the $username variable so that the form field is cleared.

One or more of the form fields are empty, so display an error message.

signup.php

there are no Dumb Questions

Q: Why couldn't you just use HTTP authentication for signing up new users?

A: Because the purpose of the Sign-Up script isn't to restrict access to pages. The Sign-Up script's job is to allow the user to enter a unique username and password, and then add them to the user database. Sure, it's possible to use the HTTP authentication window as an input form for the username and password, but the authentication functionality is overkill for just signing up a new user. It's better to create a custom form for sign-ups—then you get the benefit of double-checking the password for data entry errors.

Q: So does the Sign-Up script log in users after they sign up?

A: No. And the reason primarily has to do with the fact that the Log-In script already handles the task of logging in a user, and there's no need to duplicate the code in the Sign-Up script. The Sign-Up script instead presents a link to the Edit Profile page, which is presumably where the user would want to go after signing in. And since they aren't logged in yet, they are presented with the Log-In window as part of attempting to access the Edit Profile page. So the Sign-Up script leads the user to the Log-In window via the Edit Profile page, as opposed to logging them in automatically.

Give users a chance to sign up

We have a Sign-Up script, but how do users get to it? We need to let users know how to sign up. One option is to put a "Sign Up" link on the main Mismatch page. That's not a bad idea, but we would ideally need to be able to turn it on and off based on whether a user is logged in. Another possibility is to just show a "Sign Up" link as part of the Log-In script.

When a new user clicks the "View Profile" or "Edit Profile" links on the main page, for example, they'll be prompted for a username and password by the Log-In script. Since they don't yet have a username or password, they will likely click Cancel to bail out of the log-in. That's our chance to display a link to the Sign-Up script by tweaking the log-in failure message displayed by the Log-In script so that it provides a link to `signup.php`.

Here's the original log-in failure code:

This code just shows a log-in error message with no mention of how to sign up for Mismatch.

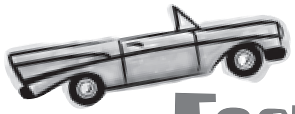
```
exit('<h3>Mismatch</h3>Sorry, you must enter your username and password to log in and access ' .
  'this page.');
```

This code actually appears in two different places in the Log-In script: when no username or password are entered and when they are entered incorrectly. It's probably a good idea to go ahead and provide a "Sign Up" link in both places. Here's what the new code might look like:

This code is much more helpful since it generates a link to the Sign-Up script so that the user can sign up.

```
exit('<h2>Mismatch</h2>Sorry, you must enter a valid username and password to log in and ' .
  'access this page. If you aren\'t a registered member, please <a href="signup.php">sign up</a>.');
```

Nothing fancy here, just a normal HTML link to the signup.php script.

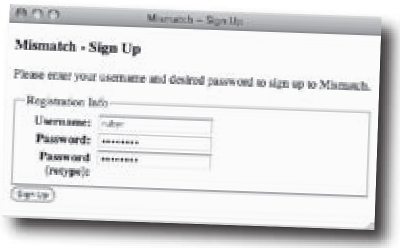


Test Drive

Add Sign-Up functionality to Mismatch.

Create a new text file named `signup.php`, and enter the code for the Sign-Up script in it (or download the script from the Head First Labs site at `www.headfirstlabs.com/books/hfphp`). Then modify the `login.php` script to add links to the Sign-Up script for users who can't log in.

Upload the scripts to your web server, and then open the Sign-Up page in a web browser. Sign up as a new user and then log in. Then edit your profile and view your profile to confirm that the sign-up and log-in worked correctly. The application now has that personalized touch that's been missing.

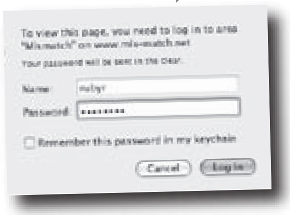


Mismatch - Sign Up

Please enter your username and desired password to sign up to Mismatch.

Registration Info

Username:	rubyr
Password:	•••••••••
Password (retype):	•••••••••



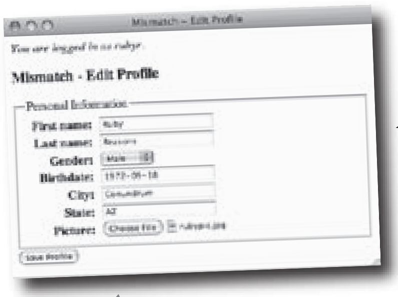
To view this page, you need to log in to area "Mismatch" on www.mis-match.net

Your password will be sent in the clear.

Name:	rubyr
Password:	•••••••••

☐ Remember this password in my keychain

HTTP authentication is used to log in Ruby based on her sign-up information.



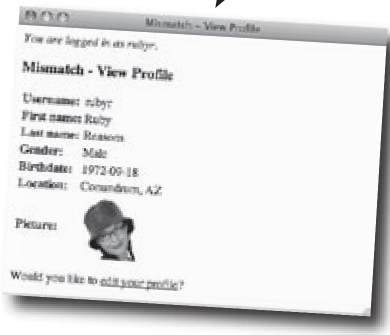
You are logged in as rubyr.

Mismatch - Edit Profile

Personal Information

First name:	Ruby
Last name:	Reasons
Gender:	Male
Birthdate:	1972-09-18
City:	Conundrum
State:	AZ
Picture:	Choose File rubyr.jpg

Sign-ups and log-ins turn an impersonal application into a community of interested users.



You are logged in as rubyr.

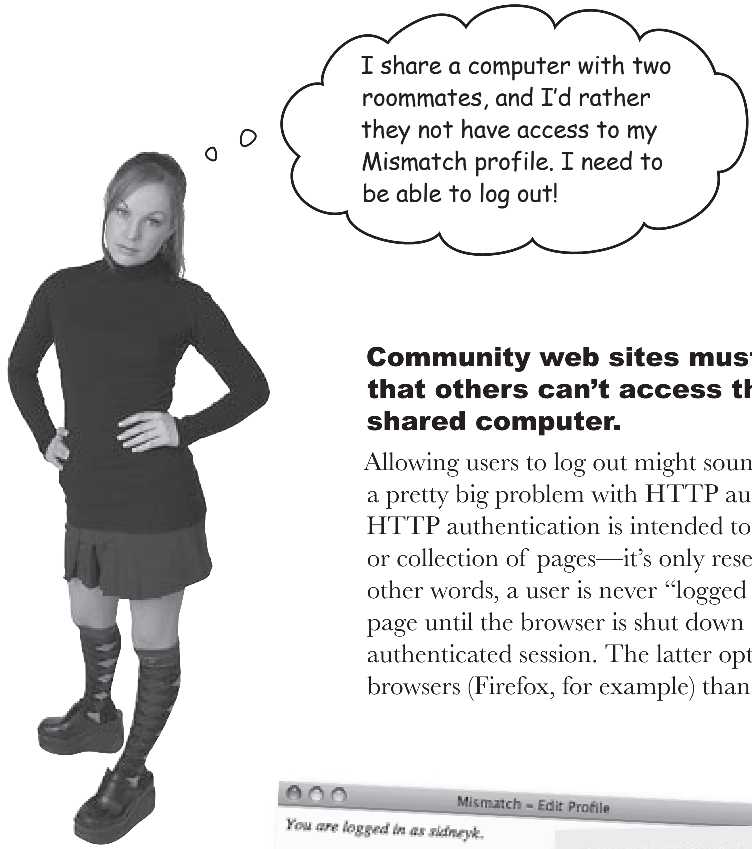
Mismatch - View Profile

Username:	rubyr
First name:	Ruby
Last name:	Reasons
Gender:	Male
Birthdate:	1972-09-18
Location:	Conundrum, AZ
Picture:	

Would you like to edit your profile?

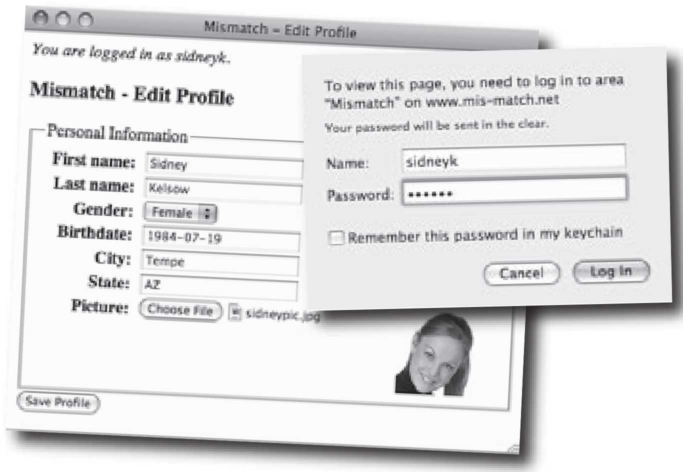
Ruby's profile is only accessible after logging in.

Cool! I can log in to Mismatch and then edit and view my personal profile.


I share a computer with two roommates, and I'd rather they not have access to my Mismatch profile. I need to be able to log out!

Community web sites must allow users to log out so that others can't access their personal data from a shared computer.

Allowing users to log out might sound simple enough, but it presents a pretty big problem with HTTP authentication. The problem is that HTTP authentication is intended to be carried out once for a given page or collection of pages—it's only reset when the browser is shut down. In other words, a user is never "logged out" of an HTTP authenticated web page until the browser is shut down or the user manually clears the HTTP authenticated session. The latter option is easier to carry out in some browsers (Firefox, for example) than others (Safari).



You are logged in as sidneyk.

Mismatch - Edit Profile

Personal Information

First name:	Sidney
Last name:	Kelsow
Gender:	Female
Birthdate:	1984-07-19
City:	Tempe
State:	AZ
Picture:	sidneypic.jpg

To view this page, you need to log in to area "Mismatch" on www.mis-match.net

Your password will be sent in the clear.

Name: sidneyk

Password: ••••••

Remember this password in my keychain

Once you log in, you stay in until you close the browser.

A log-out feature would allow Sidney to carefully control access to her personal profile.

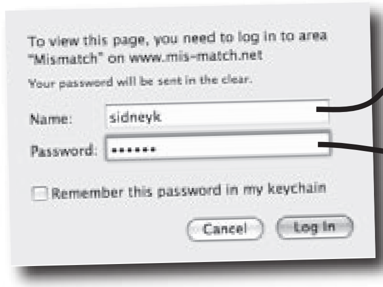
Even though HTTP authentication presents a handy and simple way to support user log-ins in the Mismatch application, it doesn't provide any control over logging a user out. We need to be able to both remember users and also allow them to log out whenever they want.

Wouldn't it be dreamy if we could remember the user without keeping them logged in forever. Am I just a hopeless PHP romantic?

Sometimes you just need a cookie

The problem originally solved by HTTP authentication is twofold: there is the issue of limiting access to certain pages, and there is the issue of remembering that the user entered information about themselves. The second problem is the tricky one because it involves an application remembering who the user is across multiple pages (scripts). Mismatch accomplishes this feat by checking the username and password stored in the `$_SERVER` superglobal. So we took advantage of the fact that PHP stores away the HTTP authentication username and password in a superglobal that persists across multiple pages.

HTTP authentication stores data persistently on the client but doesn't allow you to delete it when you're done.



To view this page, you need to log in to area "Mismatch" on www.mis-match.net
Your password will be sent in the clear.
Name: sidneyk
Password: ••••••
Remember this password in my keychain
Cancel | Log In

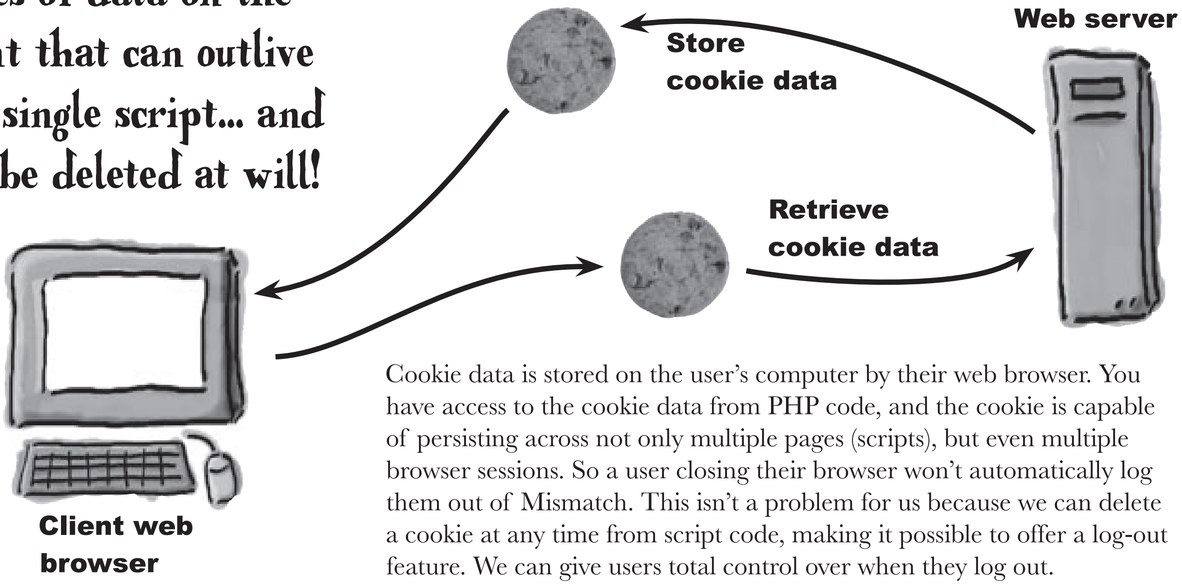
`$_SERVER['PHP_AUTH_USER']`

`$_SERVER['PHP_AUTH_PW']`

The $_SERVER superglobal stores the username and password persistently.

But we don't have the luxury of HTTP authentication anymore because it can't support log-outs. So we need to look elsewhere for user persistence across multiple pages. A possible solution lies in **cookies**, which are pieces of data stored by the browser on the user's computer. Cookies are a lot like PHP variables except that cookies hang around after you close the browser, turn off your computer, etc. More importantly, cookies can be deleted, meaning that you can eliminate them when you're finished storing data, such as when a user indicates they want to log out.

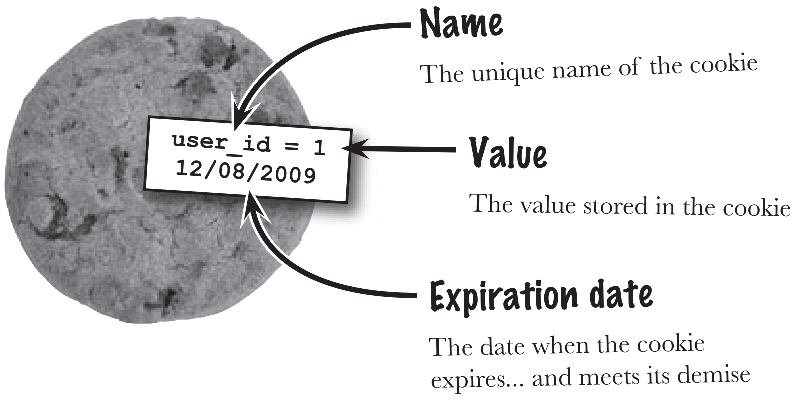
Cookies allow you to persistently store small pieces of data on the client that can outlive any single script... and can be deleted at will!



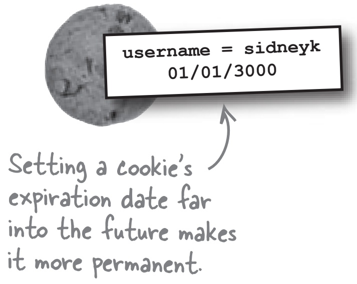
Cookie data is stored on the user's computer by their web browser. You have access to the cookie data from PHP code, and the cookie is capable of persisting across not only multiple pages (scripts), but even multiple browser sessions. So a user closing their browser won't automatically log them out of Mismatch. This isn't a problem for us because we can delete a cookie at any time from script code, making it possible to offer a log-out feature. We can give users total control over when they log out.

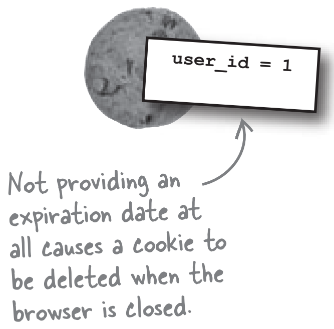
What's in a cookie?

A cookie stores **a single piece of data** under a unique name, much like a variable in PHP. Unlike a variable, a cookie can have an expiration date. When this expiration date arrives, the cookie is destroyed. So cookies aren't exactly immortal—they just live longer than PHP variables. You can create a cookie without an expiration date, in which case it acts just like a PHP variable—it gets destroyed when the browser closes.


user_id = 1
12/08/2009
Name
The unique name of the cookie
Value
The value stored in the cookie
Expiration date
The date when the cookie expires... and meets its demise

Cookies allow you to store a string of text under a certain name, kind of like a PHP text variable. It's the fact that cookies outlive normal script data that makes them so powerful, especially in situations where an application consists of multiple pages that need to remember a few pieces of data, such as log-in information.


username = sidneyk
01/01/3000
Setting a cookie's expiration date far into the future makes it more permanent.


user_id = 1
Not providing an expiration date at all causes a cookie to be deleted when the browser is closed.

So Mismatch can mimic the persistence provided by the `$_SERVER` superglobal by setting two cookies—one for the username and one for the password. Although we really don't need to keep the password around, it might be more helpful to store away the user ID instead.

there are no Dumb Questions

Q: What's the big deal about cookies being persistent? Isn't data stored in a MySQL database persistent too?

A: Yes, database data is most certainly persistent. In fact, it's technically much more persistent than a cookie because there is no expiration date involved—if you stick data in a database, it stays there until you explicitly remove it. The real issue in regard to cookies and persistence is convenience. We don't need to store the current user's ID or username for all eternity just to allow them to access their profile; we just need a quick way to know who they are. What we really need is **temporary persistence**, which might seem like an oxymoron until you consider the fact that we need data to hang around longer than a page (persistent), but not forever.

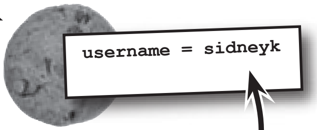
Use ~~Bake~~ cookies with PHP

PHP provides access to cookies through a function called `setcookie()` and a superglobal called `$_COOKIE`. The `setcookie()` function is used to set the value and optional expiration date of a cookie, and the `$_COOKIE` superglobal is used to retrieve the value of a cookie.

```
setcookie('username', 'sidneyk');
```

The first argument to setcookie() is the name of the cookie.

The value to be stored in the cookie is passed as the second argument.



```
echo('<p class="login">You are logged in as ' . $_COOKIE['username'] . '.</p>');
```

The name of the cookie is used to reference the cookie value in the $_COOKIE superglobal.

The power of setting a cookie is that the cookie data persists across multiple scripts, so we can remember the username without having to prompt the user to log in every time they move from one page to another within the application. But don't forget, we also need to store away the user's ID in a cookie since it serves as a primary key for database queries.

The PHP setcookie() function allows you to store data in cookies.

```
setcookie('user_id', '1');
```

Cookies are always stored as text, so even though the user ID is a number, we store it in a cookie as the string '1'.

The `setcookie()` function also accepts an optional third argument that sets the expiration date of the cookie, which is the date upon which the cookie is automatically deleted. If you don't specify an expiration date, as in the above example, the cookie automatically expires when the browser is closed.

Sharpen your pencil

Switching Mismatch to use cookies involves more than just writing a new Log-Out script. We must first revisit the Log-In script and change it to use cookies instead of HTTP authentication. Circle and annotate the parts of the Log-In code that you think need to change to accommodate cookies.

```
<?php
  require_once('connectvars.php');

  if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])) {
    // The username/password weren't entered so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h3>Mismatch</h3>Sorry, you must enter your username and password to ' .
      'log in and access this page. If you aren\'t a registered member, please ' .
      '<a href="signup.php">sign up</a>.');
  }

  // Connect to the database
  $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

  // Grab the user-entered log-in data
  $user_username = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_USER']));
  $user_password = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_PW']));

  // Look up the username and password in the database
  $query = "SELECT user_id, username FROM mismatch_user WHERE username = " .
    "'$user_username' AND password = SHA('$user_password')";
  $data = mysqli_query($dbc, $query);

  if (mysqli_num_rows($data) == 1) {
    // The log-in is OK so set the user ID and username variables
    $row = mysqli_fetch_array($data);
    $user_id = $row['user_id'];
    $username = $row['username'];
  }
  else {
    // The username/password are incorrect so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h2>Mismatch</h2>Sorry, you must enter a valid username and password ' .
      'to log in and access this page. If you aren\'t a registered member, ' .
      'please <a href="signup.php">sign up</a>.');
  }

  // Confirm the successful log-in
  echo('<p class="login">You are logged in as ' . $username . '.</p>');
?>
```

login.php

Sharpen your pencil Solution

Switching Mismatch to use cookies involves more than just writing a new Log-Out script. We must first revisit the Log-In script and change it to use cookies instead of HTTP authentication. Circle and annotate the parts of the Log-In code that you think need to change to accommodate cookies.

We need to check for the existence of a cookie to see if the user is logged in or not.

Instead of getting the username and password from an authentication window, we need to use a form with POST data.

We no longer need to send HTTP authentication headers.

```
<?php
  require_once('connectvars.php');

  if (!isset($_SERVER['PHP_AUTH_USER']) || !isset($_SERVER['PHP_AUTH_PW'])) {
    // The username/password weren't entered so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h3>Mismatch</h3>Sorry, you must enter your username and password to ' .
      'log in and access this page. If you aren\'t a registered member, please ' .
      '<a href="signup.php">sign up</a>.');
  }

  // Connect to the database
  $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

  // Grab the user-entered log-in data
  $user_username = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_USER']));
  $user_password = mysqli_real_escape_string($dbc, trim($_SERVER['PHP_AUTH_PW']));

  // Look up the username and password in the database
  $query = "SELECT user_id, username FROM mismatch_user WHERE username = " .
    "'$user_username' AND password = SHA('$user_password')";
  $data = mysqli_query($dbc, $query);

  if (mysqli_num_rows($data) == 1) {
    // The log-in is OK so set the user ID and username variables
    $row = mysqli_fetch_array($data);
    $user_id = $row['user_id'];
    $username = $row['username'];
  }
  else {
    // The username/password are incorrect so send the authentication headers
    header('HTTP/1.1 401 Unauthorized');
    header('WWW-Authenticate: Basic realm="Mismatch"');
    exit('<h2>Mismatch</h2>Sorry, you must enter a valid username and password ' .
      'to log in and access this page. If you aren\'t a registered member, ' .
      'please <a href="signup.php">sign up</a>.');
  }

  // Confirm the successful log-in
  echo('<p class="login">You are logged in as ' . $username . '.</p>');
?>
```

The query doesn't have to change at all!

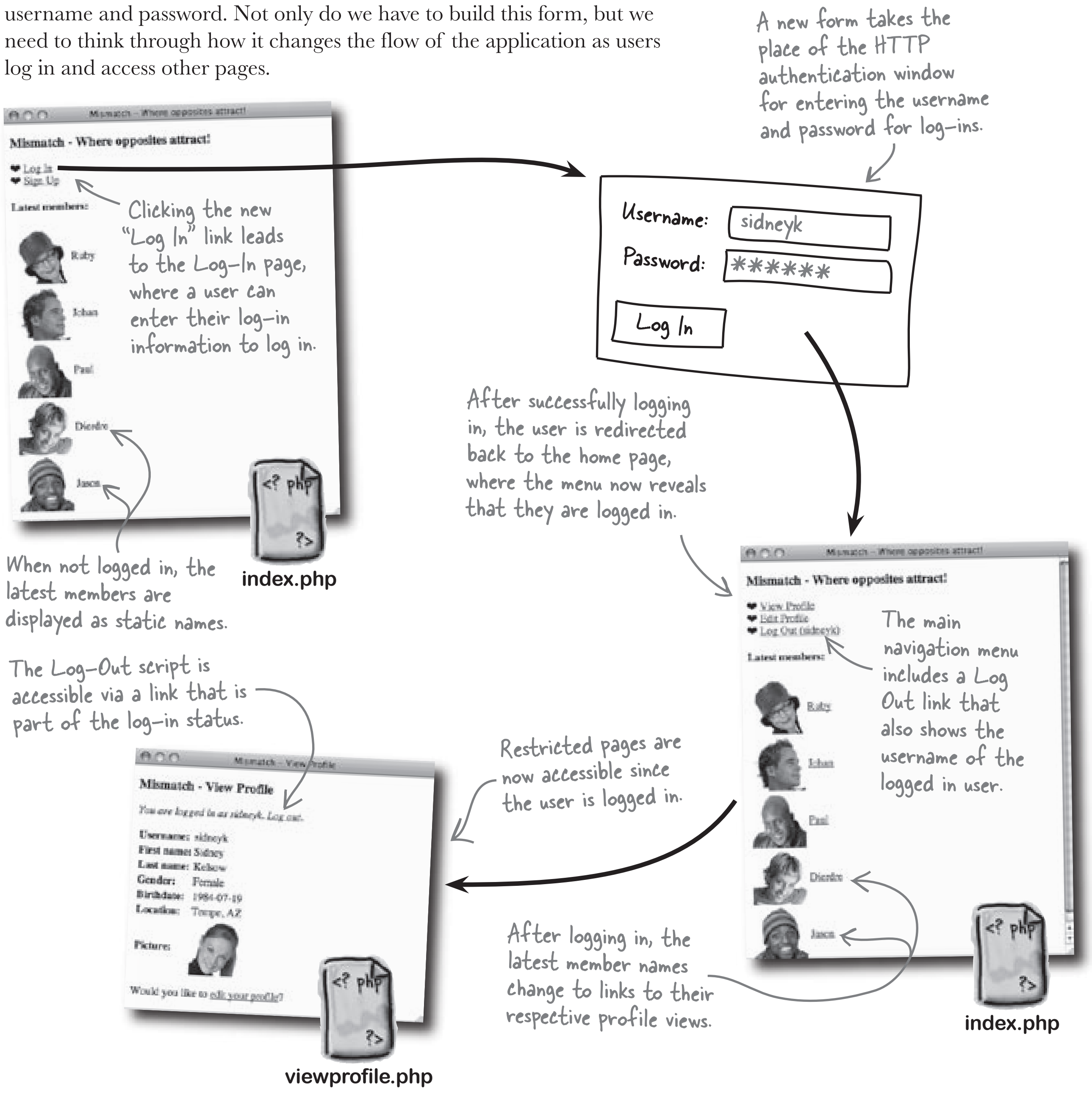
Here we need to set two cookies instead of setting script variables.

Since we can't rely on the HTTP authentication window for entering the username and password, we need to create an HTML Log-In form for entering them.

login.php

Rethinking the flow of log-ins

Using cookies instead of HTTP authentication for Mismatch log-ins involves more than just rethinking the storage of user data. What about the log-in user interface? The cookie-powered log-in must provide its own form since it can't rely on the authentication window for entering a username and password. Not only do we have to build this form, but we need to think through how it changes the flow of the application as users log in and access other pages.


Mismatch - Where opposites attract!
Mismatch - Where opposites attract!
Log In
Sign Up
Latest members:
Ruby
Johan
Paul
Dierdre
Jason
Clicking the new "Log In" link leads to the Log-In page, where a user can enter their log-in information to log in.
When not logged in, the latest members are displayed as static names.
index.php
A new form takes the place of the HTTP authentication window for entering the username and password for log-ins.
Username: sidneyk
Password: ******
Log In
After successfully logging in, the user is redirected back to the home page, where the menu now reveals that they are logged in.
Mismatch - Where opposites attract!
View Profile
Edit Profile
Log Out (sidneyk)
Latest members:
Ruby
Johan
Paul
Dierdre
Jason
The main navigation menu includes a Log Out link that also shows the username of the logged in user.
After logging in, the latest member names change to links to their respective profile views.
index.php
The Log-Out script is accessible via a link that is part of the log-in status.
Restricted pages are now accessible since the user is logged in.
Mismatch - View Profile
viewprofile.php

Mismatch - View Profile

You are logged in as sidneyk. Log out.

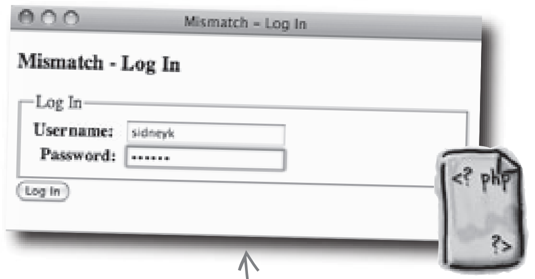
Username: sidneyk
First name: Sidney
Last name: Kelsow
Gender: Female
Birthdate: 1984-07-19
Location: Tempe, AZ

Picture:

Would you like to edit your profile?

A cookie-powered log-in

The new version of the Log-In script that relies on cookies for log-in persistence is a bit more complex than its predecessor since it must provide its own form for entering the username and password. But it's more powerful in that it provides log-out functionality.



Mismatch - Log In

Username: sidneyk

Password: ••••••

Here's the new Log-In form.

```
<?php
  require_once('connectvars.php');

  // Clear the error message
  $error_msg = "";

  // If the user isn't logged in, try to log them in
  if (!isset($_COOKIE['user_id'])) {
    if (isset($_POST['submit'])) {
      // Connect to the database
      $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

      // Grab the user-entered log-in data
      $user_username = mysqli_real_escape_string($dbc, trim($_POST['username']));
      $user_password = mysqli_real_escape_string($dbc, trim($_POST['password']));

      if (!empty($user_username) && !empty($user_password)) {
        // Look up the username and password in the database
        $query = "SELECT user_id, username FROM mismatch_user WHERE username = '$user_username' AND " .
          "password = SHA('$user_password')";
        $data = mysqli_query($dbc, $query);

        if (mysqli_num_rows($data) == 1) {
          // The log-in is OK so set the user ID and username cookies, and redirect to the home page
          $row = mysqli_fetch_array($data);
          setcookie('user_id', $row['user_id']);
          setcookie('username', $row['username']);
          $home_url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
          header('Location: ' . $home_url);
        }
        else {
          // The username/password are incorrect so set an error message
          $error_msg = 'Sorry, you must enter a valid username and password to log in.';
        }
      }
      else {
        // The username/password weren't entered so set an error message
        $error_msg = 'Sorry, you must enter your username and password to log in.';
      }
    }
  }
?>

<html>
<head>
  <title>Mismatch - Log In</title>
  <link rel="stylesheet" type="text/css" href="style.css" />
</head>
<body>
  <h3>Mismatch - Log In</h3>
```

Error messages are now stored in a variable and displayed, if necessary, later in the script.

Check the user_id cookie to see if the user is logged in.

If the user isn't logged in, see if they've submitted log-in data.

The user-entered data now comes from form POST data instead of an authentication window.

Log in the user by setting user_id and username cookies.

Redirect the user to the Mismatch home page upon a successful log-in.

Set the error message variable if anything is wrong with the log-in data.

The Log-In script is now a full web page, so it requires all the standard HTML elements.

continues on the facing page...

```
<?php
  // If the cookie is empty, show any error message and the log-in form; otherwise confirm the log-in
  if (empty($_COOKIE['user_id'])) {
    echo '<p class="error">' . $error_msg . '</p>';
?>

  <form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
    <fieldset>
      <legend>Log In</legend>
      <label for="username">Username:</label>
      <input type="text" id="username" name="username"
        value="<?php if (!empty($user_username)) echo $user_username; ?>" /><br />
      <label for="password">Password:</label>
      <input type="password" id="password" name="password" />
    </fieldset>
    <input type="submit" value="Log In" name="submit" />
  </form>

<?php
  }
  else {
    // Confirm the successful log in
    echo('<p class="login">You are logged in as ' . $_COOKIE['username'] . '.</p>');
  }
?>

</body>
</html>
```

If the user still isn't logged in at this point, go ahead and show the error message.

These two form fields are used to enter the username and password for logging in.

Everything prior to this curly brace is still part of the first if clause.

If the user is logged in at this point, just tell them so.

Finish the HTML code to complete the Log-In web page.

there are no Dumb Questions

Q: Why is it necessary to store both the user ID and username in cookies?

A: Since both pieces of information uniquely identify a user within the Mismatch user database, you could use either one for the purpose of keeping up with the current user. However, `user_id` is a better (more efficient) user reference with respect to the database because it is a numeric primary key. On the other hand, `user_id` is fairly cryptic and doesn't have any meaning to the user, so username comes in handy for letting the user know they are logged in, such as displaying their name on the page. Since multiple people sometimes share the same computer, it is important to not just let the user know they are logged in, but also who they are logged in *as*.

Q: Then why not also store the password in a cookie as part of the log-in data?

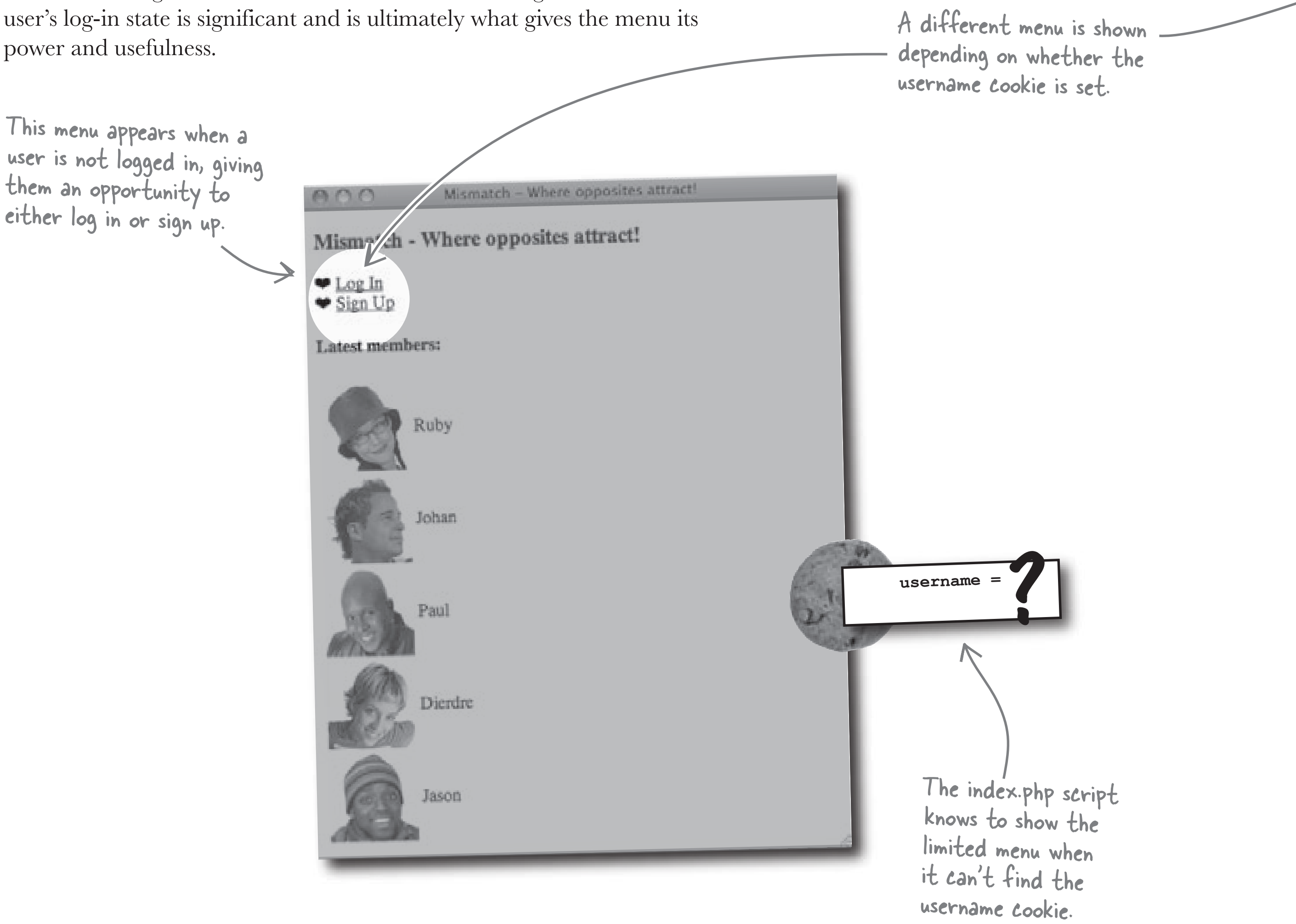
A: The password is only important for initially verifying that a user is who they claim to be. Once the password is verified as part of the log-in process, there is no reason to keep it around. Besides, passwords are very sensitive data, so it's a good idea to avoid storing them temporarily if at all possible.

Q: It looks as if the form in the Log-In script is actually inside the `if` statement? Is that possible?

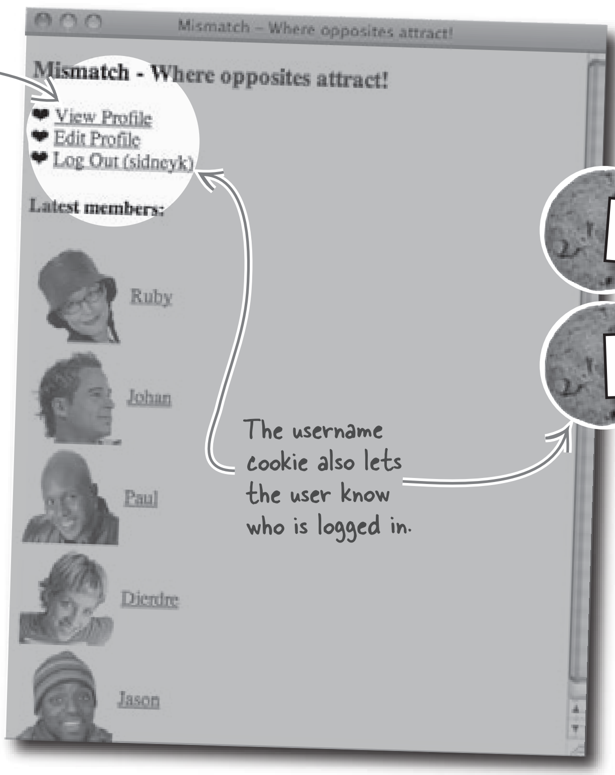
A: Yes. In fact it's quite common for PHP code to be "broken up" around HTML code, as is the case with the Log-In script. Just because you close a section of PHP code with `?>`, doesn't mean the logic of the code is closed. When you open another section of PHP code with `<?php`, the logic continues right where it left off. In the Log-In script, the HTML form is contained within the first `if` branch, while the `else` branch picks up after the form code. Breaking out of PHP code into HTML code like this keeps you from having to generate the form with a bunch of messy `echo` statements.

Navigating the Mismatch application

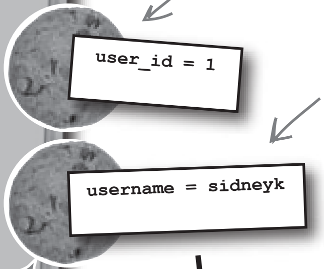
The new Log-In script changes the flow of the Mismatch application, requiring a simple menu that appears on the home page (index.php). This menu is important because it provides access to the different major parts of the application, currently the View Profile and Edit Profile pages, as well as the ability for users to log in, sign up, and log out depending on their current log-in state. The fact that the menu changes based on the user's log-in state is significant and is ultimately what gives the menu its power and usefulness.


A different menu is shown depending on whether the username cookie is set.
This menu appears when a user is not logged in, giving them an opportunity to either log in or sign up.
Mismatch – Where opposites attract!
Mismatch - Where opposites attract!
Log In
Sign Up
Latest members:
Ruby
Johan
Paul
Dierdre
Jason
username = ?
The index.php script knows to show the limited menu when it can't find the username cookie.

The menu is generated by PHP code within the index.php script, and this code uses the $_COOKIE superglobal to look up the username cookie and see if the user is logged in or not. The user ID cookie could have also been used, but the username is actually displayed in the menu, so it makes more sense to check for it instead.



The user_id cookie isn't used for the different menus but is still important for Mismatch user persistence.



The username cookie determines which menu is displayed

The username cookie also lets the user know who is logged in.

Menu for logged in users

```
// Generate the navigation menu
if (isset($_COOKIE['username'])) {
  echo '&#10084; <a href="viewprofile.php">View Profile</a><br />';
  echo '&#10084; <a href="editprofile.php">Edit Profile</a><br />';
  echo '&#10084; <a href="logout.php">Log Out (' . $_COOKIE['username'] . ')</a>';
}
else {
  echo '&#10084; <a href="login.php">Log In</a><br />';
  echo '&#10084; <a href="signup.php">Sign Up</a>';
}
```

Menu for visitors (users who aren't logged in)

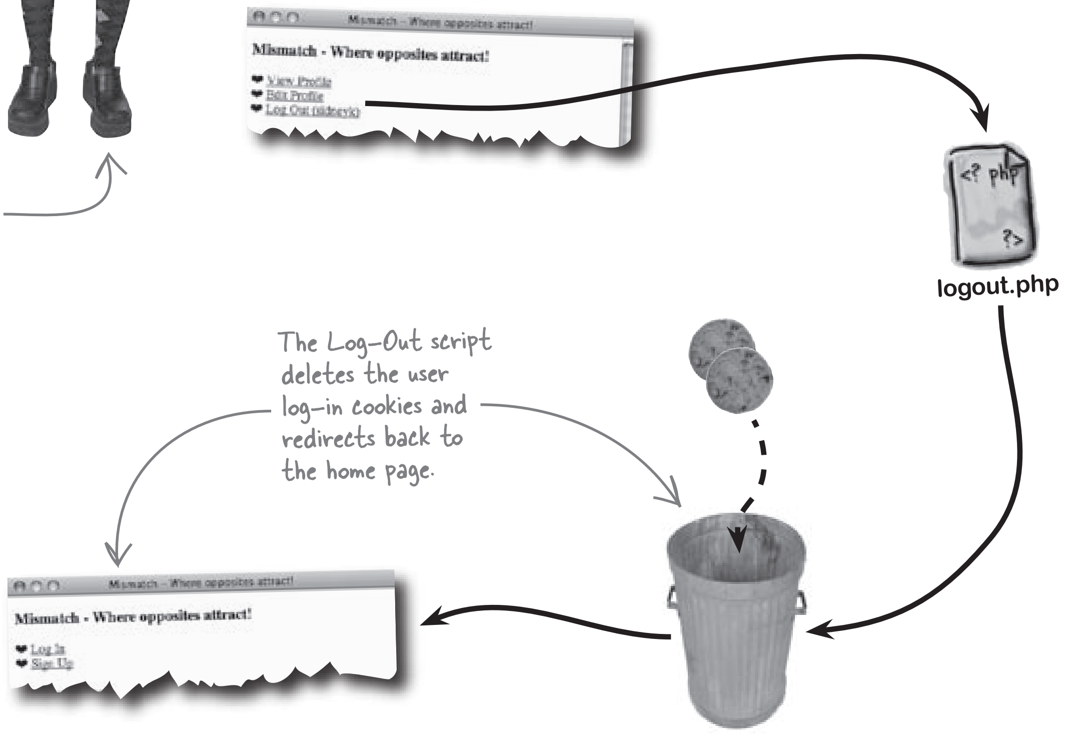
The little heart symbols next to each menu item are made possible by this HTML entity, which is supported on most browsers.

Hello, remember me? I still really, really need to log out.
Sidney is still waiting to log out...

We really need to let users log out.

Cookies have made logging into Mismatch and navigating the site a bit cleaner, but the whole point of switching from HTTP authentication to cookies was to allow users to log out. We need a new Log-Out script that deletes the two cookies (user ID and username) so that the user no longer has access to the application. This will prevent someone from getting on the same computer later and accessing a user's private profile data.

Since there is no user interface component involved in actually logging out a user, it's sufficient to just redirect them back to the home page after logging them out.


Mismatch - Where opposites attract!
Mismatch - Where opposites attract!
View Profile
Edit Profile
Log Out (sidneyk)
<? php ?>
logout.php
The Log-Out script deletes the user log-in cookies and redirects back to the home page.
Mismatch - Where opposites attract!
Mismatch - Where opposites attract!
Log In
Sign Up

Logging out means deleting cookies

Logging out a user involves deleting the two cookies that keep track of the user. This is done by calling the `setcookie()` function, and passing an expiration date that causes the cookies to get deleted at that time.

Minutes

The current time

Seconds

Hours

```
setcookie('username', 'sidneyk', time() + (60 * 60 * 8));
```

Together, this expression sets an expiration date that is 8 hours from the current time.

This code sets an expiration date 8 hours into the future, which means the cookie will be automatically deleted in 8 hours. But we want to delete a cookie immediately, which requires setting the expiration date to a time in the past. The amount of time into the past isn't terribly important—just pick an arbitrary amount of time, such as an hour, and subtract it from the current time.

```
setcookie('username', 'sidneyk', time() - 3600);
```

60 seconds * 60 minutes = 3600 seconds, which is 1 hour into the past.

To delete a cookie, just set its expiration date to a time in the past.

The Log-Out script for Mismatch is missing a few pieces of code. Write the missing code, making sure that the log-in cookies get deleted before the Log-Out page is redirected to the home page.

```
<?php
  // If the user is logged in, delete the cookie to log them out
  if (............................................) {
    // Delete the user ID and username cookies by setting their expirations to an hour ago (3600)
     ....................................................
     ....................................................
  }

  // Redirect to the home page
  $home_url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '.....................';
  header('Location: ' . $home_url);
?>
```

The Log-Out script for Mismatch is missing a few pieces of code. Write the missing code, making sure that the log-in cookies get deleted before the Log-Out page is redirected to the home page.

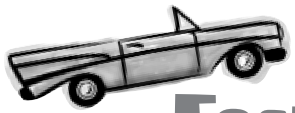
```
<?php
  // If the user is logged in, delete the cookie to log them out
  if ( isset($_COOKIE['user_id']) ) {
    // Delete the user ID and username cookies by setting their expirations to an hour ago (3600)
    setcookie('user_id', '', time() - 3600);
    setcookie('username', '', time() - 3600);
  }

  // Redirect to the home page
  $home_url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . ' /index.php ';
  header('Location: ' . $home_url);
?>
```

Only log out a user if they are already logged in.

Set each cookie to an hour in the past so that they are deleted by the system.

Redirect to the Mismatch home page, which is constructed as an absolute URL.

A location header results in the browser redirecting to another page.

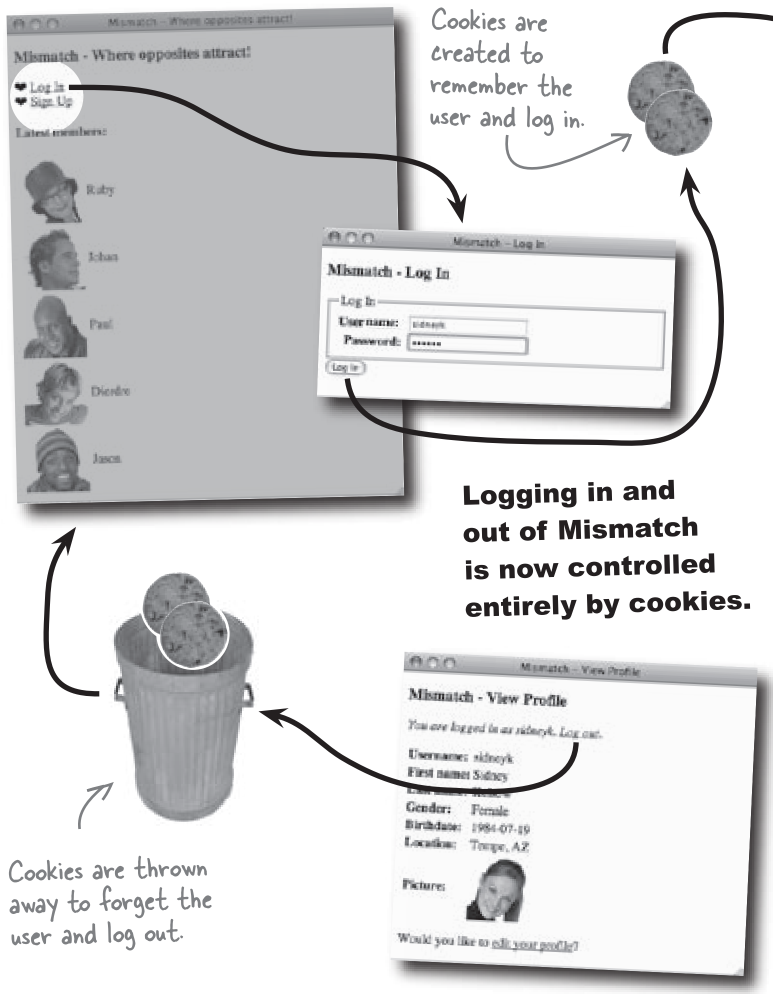


Test Drive

Use cookies to add Log-Out functionality to Mismatch.

Modify the Mismatch scripts so that they use cookies to allows users to log in and out (or download the scripts from the Head First Labs site at `www.headfirstlabs.com/books/hfphp`. The cookie modifications involve changes to the `index.php`, `login.php`, `logout.php`, `editprofile.php`, and `viewprofile.php` scripts. The changes to the latter two scripts are fairly minor, and primarily involve changing `$user_id` and `$username` global variable references so that they use the `$_COOKIE` superglobal instead.

Upload the scripts to your web server, and then open the main Mismatch page (`index.php`) in a web browser. Take note of the navigation menu, and then click the “Log In” link and log in. Notice how the Log-In script leads you back to the main page, while the menu changes to reflect your logged in status. Now click “Log Out” to blitz the cookies and log out.



Logging in and out of Mismatch is now controlled entirely by cookies.

I'm outta here for now. See ya!
Sidney is pleased that she can log out and know that her Mismatch profile can't be edited by anyone while she's away.

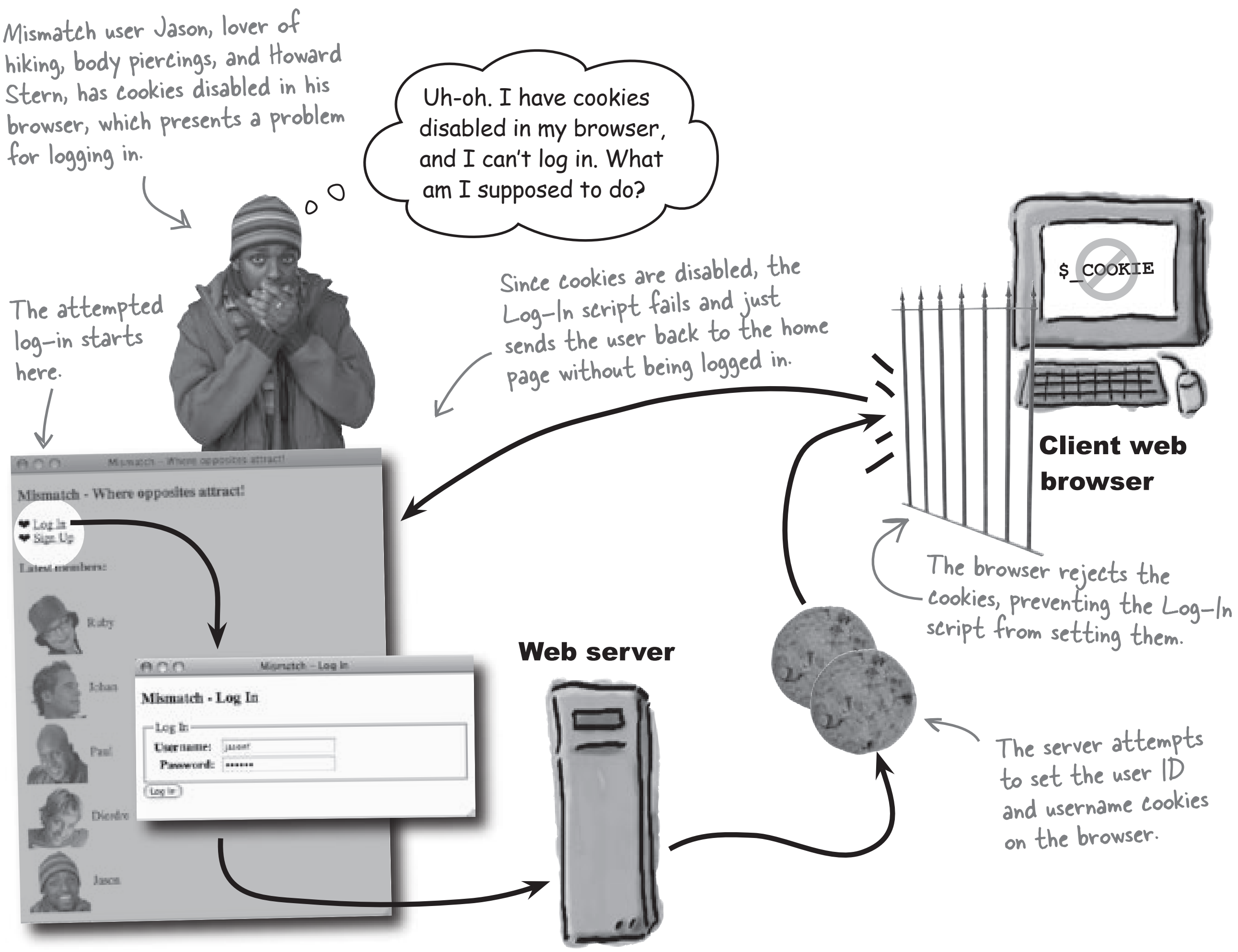
there are no Dumb Questions

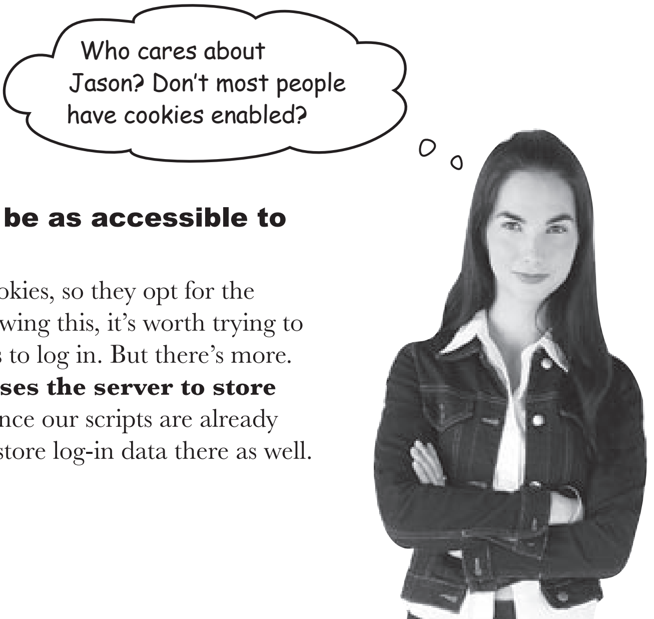
Q: So simply deleting the cookies is all that is required to log out?

A: Yes. Cookies are responsible for storing all of the log-in information for Mismatch (user ID and username), so deleting them results in a complete log-out.

Q: Why are the cookies set to an hour in the past in order to be deleted? Is there something significant about an hour?

A: No. A cookie is automatically deleted by the web browser once its expiration date/time passes. So deleting a cookie involves setting the expiration to any time in the past. An hour (3600 seconds) is just an arbitrary amount of time chosen to consistently indicate that we're deleting a cookie.


Mismatch user Jason, lover of hiking, body piercings, and Howard Stern, has cookies disabled in his browser, which presents a problem for logging in.
Uh-oh. I have cookies disabled in my browser, and I can't log in. What am I supposed to do?
The attempted log-in starts here.
Since cookies are disabled, the Log-In script fails and just sends the user back to the home page without being logged in.
$_COOKIE
Client web browser
Mismatch - Where opposites attract!
Log In
Sign Up
Latest members:
Ruby
Johan
Paul
Dierdre
Jason
Mismatch - Log In
Log In
Username: jason
Password: ••••••
Log In
Web server
The browser rejects the cookies, preventing the Log-In script from setting them.
The server attempts to set the user ID and username cookies on the browser.


Who cares about Jason? Don't most people have cookies enabled?

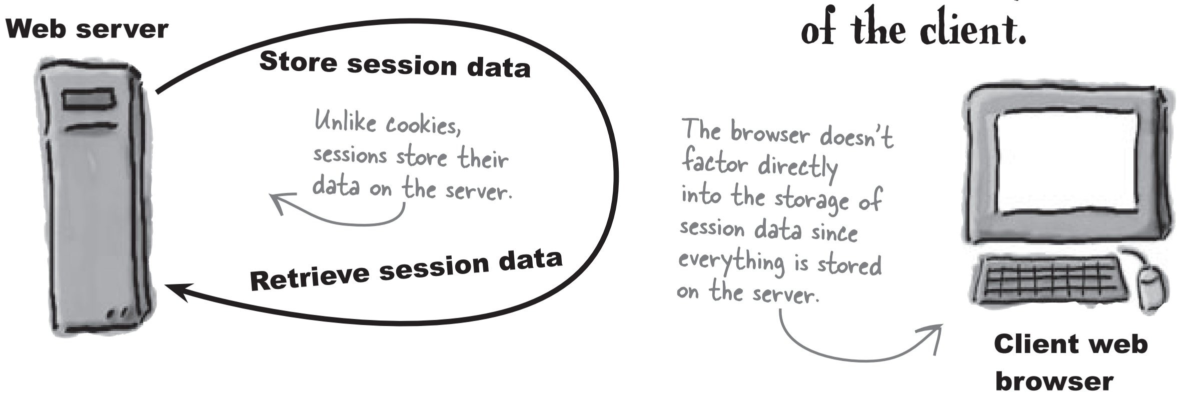
Yes, but web applications should be as accessible to as many people as possible.

Some people just aren't comfortable using cookies, so they opt for the added security of having them disabled. Knowing this, it's worth trying to accommodate users who can't rely on cookies to log in. But there's more. It turns out that there's another option that **uses the server to store log-in data**, as opposed to the client. And since our scripts are already running on the server, it only makes sense to store log-in data there as well.

Sessions aren't dependent on the client

Cookies are powerful little guys, but they do have their limitations, such as being subject to limitations beyond your control. But what if we didn't have to depend on the browser? What if we could store data directly on the server? **Sessions** do just that, and they allow you to store away individual pieces of information just like with cookies, but the data gets stored on the server instead of the client. This puts session data outside of the browser limitations of cookies.

Sessions allow you to persistently store small pieces of data on the <u>server</u>, independently of the client.

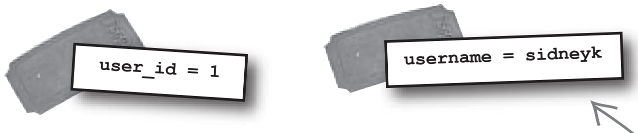


Web server
Store session data
Unlike cookies, sessions store their data on the server.
Retrieve session data
The browser doesn't factor directly into the storage of session data since everything is stored on the server.
Client web browser

Sessions store data in **session variables**, which are logically equivalent to cookies on the server. When you place data in a session variable using PHP code, it is stored on the server. You can then access the data in the session variable from PHP code, and it remains persistent across multiple pages (scripts). Like with cookies, you can delete a session variable at any time, making it possible to continue to offer a log-out feature with session-based code.

Since session data is stored on the server, it is <u>more</u> <u>secure</u> and <u>more</u> <u>reliable</u> than data stored in cookies.

A user can't manually delete session data using their browser, which can be a problem with cookies.



There isn't an expiration date associated with session variables because they are automatically deleted when a session ends.

Surely there's a catch, right? Sort of. Unlike cookies, sessions don't offer as much control over how long a session variable stores data. Session variables are **automatically destroyed as soon as a session ends**, which usually coincides with the user shutting down the browser. So even though session variables aren't stored on the browser, they are indirectly affected by the browser since they get deleted when a browser session ends.

The life and times of sessions

Sessions are called sessions for a reason—they have a very clear start and finish. Data associated with a session lives and dies according to the lifespan of the session, which you control through PHP code. The only situation where you don't have control of the session life cycle is when the user closes the browser, which results in a session ending, whether you like it or not.

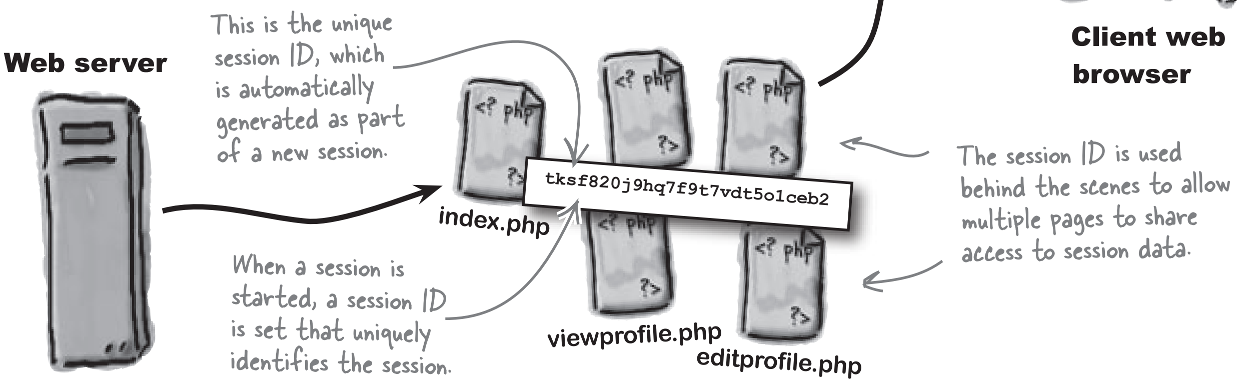
You must tell a session when you're ready to start it up by calling the `session_start()` PHP function.

The PHP session_start() function starts a session and allows you to begin storing data in session variables.

```
session_start();
```

This PHP function starts a session.

Calling the `session_start()` function doesn't set any data—its job is to get the session up and running. The session is identified internally by a unique session identifier, which you typically don't have to concern yourself with. This ID is used by the web browser to associate a session with multiple pages.



Web server
This is the unique session ID, which is automatically generated as part of a new session.
<? php
?>
tksf820j9hq7f9t7vdt5o1ceb2
index.php
viewprofile.php
editprofile.php
When a session is started, a session ID is set that uniquely identifies the session.
The session ID is used behind the scenes to allow multiple pages to share access to session data.
Client web browser

The session ID isn't destroyed until the session is closed, which happens either when the browser is closed or when you call the `session_destroy()` function.

```
session_destroy();
```

This PHP function ends a session.

The session_destroy() function closes a session.

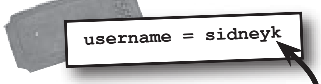
If you close a session yourself with this function, it doesn't automatically destroy any session variables you've stored. Let's take a closer look at how sessions store data to uncover why this is so.

Keeping up with session data

The cool thing about sessions is that they're very similar to cookies in terms of how you use them. Once you've started a session with a call to `session_start()`, you can begin setting session variables, such as Mismatch log-in data, with the **`$_SESSION`** superglobal.

```
$_SESSION['username'] = 'sidneyk';
```

The session variable is created and stored on the server.



The name of the session variable is used as an index into the $_SESSION superglobal.

The value to be stored is just assigned to the $_SESSION superglobal.

```
echo('<p class="login">You are logged in as ' . $_SESSION['username'] . '.</p>');
```

To access the session variable, just use the $_SESSION superglobal and the session variable name.

Unlike cookies, session variables don't require any kind of special function to set them—you just assign a value to the `$_SESSION` superglobal, making sure to use the session variable name as the array index.

What about deleting session variables? Destroying a session via `session_destroy()` doesn't actually destroy session variables, so you must manually delete your session variables if you want them to be killed prior to the user shutting down the browser (log-outs!). A quick and effective way to destroy all of the variables for a session is to set the `$_SESSION` superglobal to an empty array.

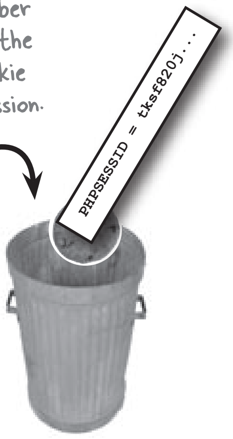
Session variables are <u>not</u> automatically deleted when a session is destroyed.

```
$_SESSION = array();
```

This code kills all of the session variables in the current session.

But we're not quite done. Sessions can actually use cookies behind the scenes. If the browser allows cookies, a session may possibly set a cookie that temporarily stores the session ID. So to fully close a session via PHP code, you must also delete any cookie that might have been automatically created to store the session ID on the browser. Like any other cookie, you destroy this cookie by setting its expiration to some time in the past. All you need to know is the name of the cookie, which can be found using the `session_name()` function.

If a session is using a cookie to help remember the session ID, then the ID is stored in a cookie named after the session.



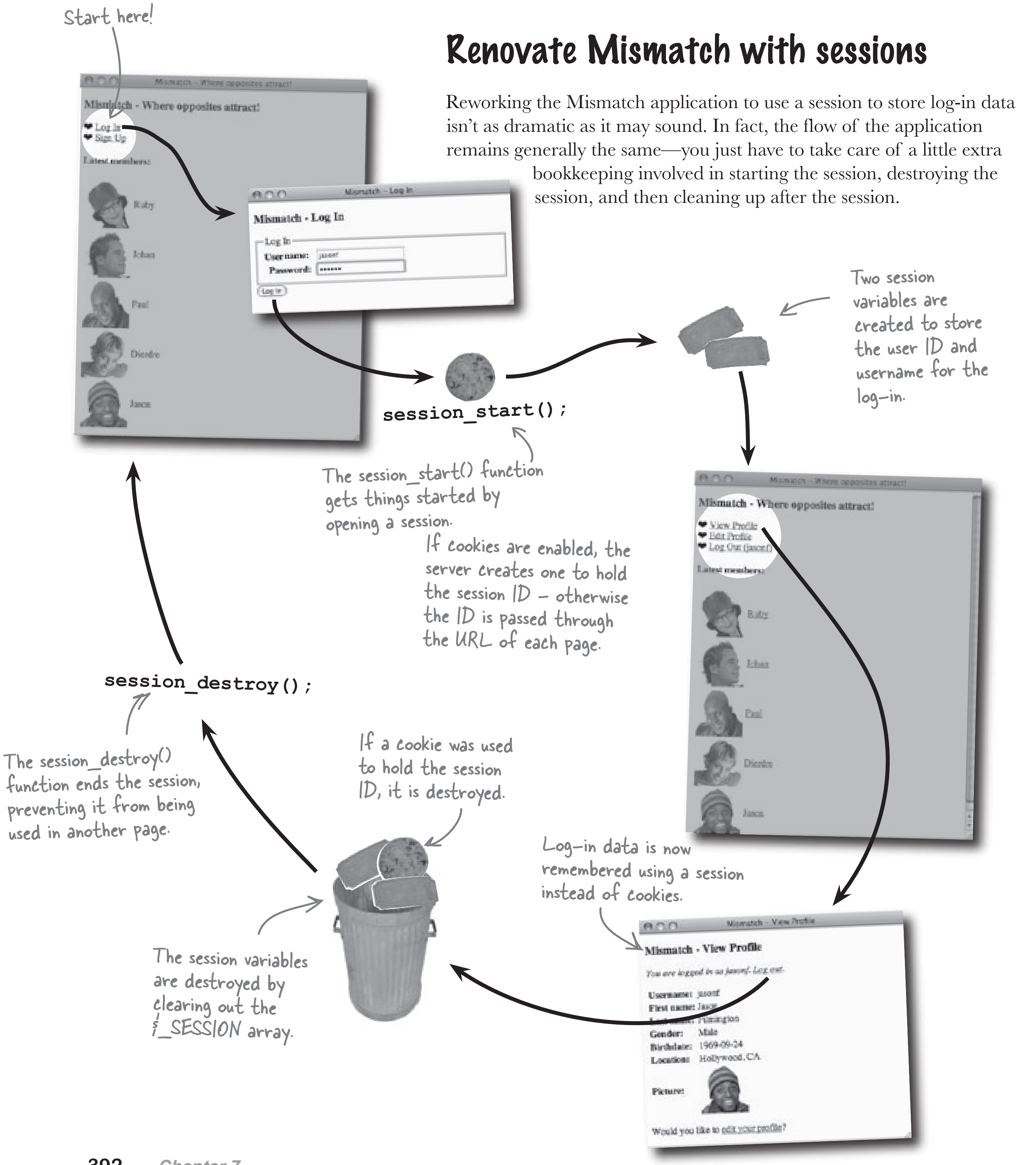
```
if (isset($_COOKIE[session_name()])) {
  setcookie(session_name(), '', time() - 3600);
}
```

First check to see if a session cookie actually exists.

Destroy the session cookie by setting its expiration to an hour in the past.

Renovate Mismatch with sessions

Reworking the Mismatch application to use a session to store log-in data isn't as dramatic as it may sound. In fact, the flow of the application remains generally the same—you just have to take care of a little extra bookkeeping involved in starting the session, destroying the session, and then cleaning up after the session.


Start here!
Mismatch - Where opposites attract!
Log In
Sign Up
Latest members:
Ruby
Johan
Paul
Dierdre
Jason
Mismatch - Log In
Log In
User name: jasonf
Password: ••••••
Log In
session_start();
The session_start() function gets things started by opening a session.
If cookies are enabled, the server creates one to hold the session ID – otherwise the ID is passed through the URL of each page.
Two session variables are created to store the user ID and username for the log-in.
Mismatch - Where opposites attract!
View Profile
Edit Profile
Log Out (jasonf)
Latest members:
Ruby
Johan
Paul
Dierdre
Jason
Log-in data is now remembered using a session instead of cookies.
Mismatch - View Profile
If a cookie was used to hold the session ID, it is destroyed.
The session variables are destroyed by clearing out the $_SESSION array.
session_destroy();
The session_destroy() function ends the session, preventing it from being used in another page.

Mismatch - View Profile

You are logged in as jasonf. Log out.

Username: jasonf
First name: Jason
Last name: Filmington
Gender: Male
Birthdate: 1969-09-24
Location: Hollywood, CA
Picture:

Would you like to edit your profile?

Log out with sessions

Logging a user out of Mismatch requires a little more work with sessions than the previous version with its pure usage of cookies. These steps must be taken to successfully log a user out of Mismatch using sessions.

1. **Delete the session variables.**
2. **Check to see if a session cookie exists, and if so, delete it.**
3. **Destroy the session.**
4. **Redirect the user to the home page.**

You don't know for certain if a session cookie is being used without checking.

OK, so this is a bonus step that isn't strictly required to log the user out, but is helpful nonetheless.

Sharpen your pencil

The Log-Out script for Mismatch is undergoing an overhaul to use sessions instead of pure cookies for log-in persistence. Write the missing code to "sessionize" the Log-Out script, and then annotate which step of the log-out process it corresponds to.

```
<?php
  // If the user is logged in, delete the session vars to log them out
  session_start();
  if ( ......................................... ) {
    // Delete the session vars by clearing the $_SESSION array

    ..................................

    // Delete the session cookie by setting its expiration to an hour ago (3600)
    if (isset($_COOKIE[session_name()])) {

      ...............................................................
    }


    // Destroy the session

    ...........................
  }


  // Redirect to the home page
  $home_url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
  header('Location: ' . $home_url);
?>
```

Sharpen your pencil Solution

The Log-Out script for Mismatch is undergoing an overhaul to use sessions instead of pure cookies for log-in persistence. Write the missing code to "sessionize" the Log-Out script, and then annotate which step of the log-out process it corresponds to.

1. Delete the session variables.
2. Check to see if a session cookie exists, and if so, delete it.
3. Destroy the session.
4. Redirect the user to the home page.

Even when logging out, you have to first start the session in order to access the session variables.

```
<?php
  // If the user is logged in, delete the session vars to log them out
  session_start();
  if ( isset($_SESSION['user_id']) ) {
    // Delete the session vars by clearing the $_SESSION array
    $_SESSION = array();   1

    // Delete the session cookie by setting its expiration to an hour ago (3600)
    if (isset($_COOKIE[session_name()])) {
      setcookie(session_name(), '', time() - 3600);   2
    }

    // Destroy the session
    session_destroy();   3
  }

  // Redirect to the home page
  $home_url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
  header('Location: ' . $home_url);   4
?>
```

Now a session variable is used to check the log-in status instead of a cookie.

To clear out the session variables, assign the $_SESSION superglobal an empty array.

If a session cookie exists, delete it by setting its expiration to an hour ago.

Destroy the session with a call to the built-in session_destroy() function.

How do I change?

The move from cookies to sessions impacts more than just the Log-Out script. Match the other pieces of the Mismatch application with how they need to change to accommodate sessions.

appvars.php

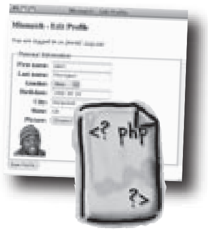
connectvars.php

login.php

signup.php

index.php

viewprofile.php


editprofile.php

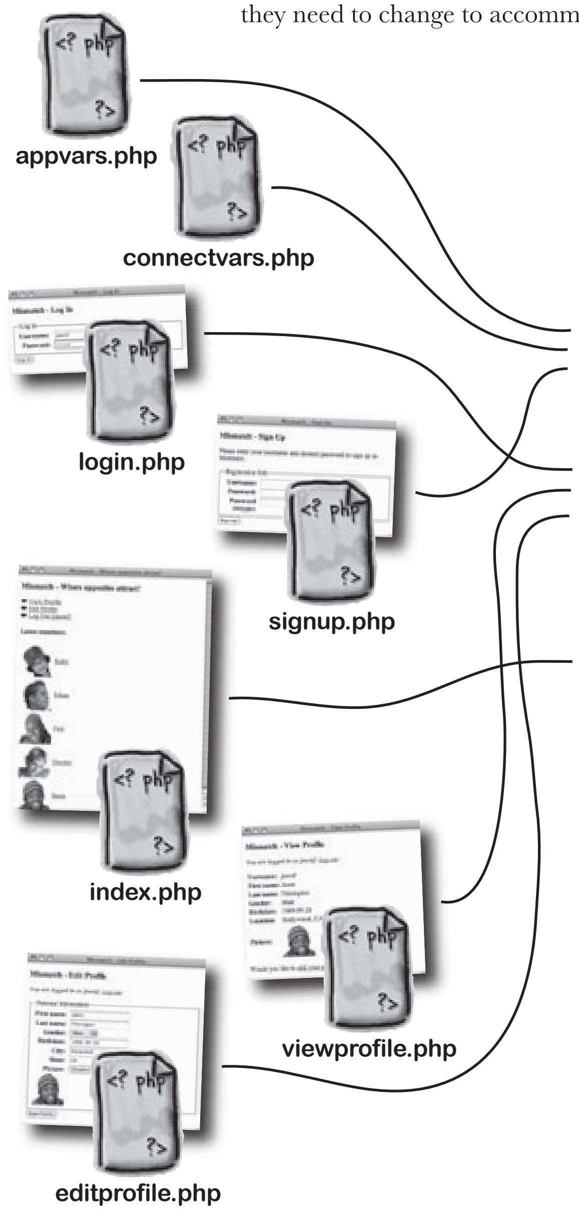
No change since the script has no direct dependence on log-in persistence.

Sessions are required to remember who the user is. Call the `session_start()` function to start the session, and then change `$_COOKIE` references to `$_SESSION`.

Sessions are required to control the navigation menu. Call the `session_start()` function to start the session, and then change `$_COOKIE` references to `$_SESSION`.

How do I change? Solution

The move from cookies to sessions impacts more than just the Log-Out script. Match the other pieces of the Mismatch application with how they need to change to accommodate sessions.



No change since the script has no direct dependence on log-in persistence.

Sessions are required to remember who the user is. Call the `session_start()` function to start the session, and then change `$_COOKIE` references to `$_SESSION`.

Sessions are required to control the navigation menu. Call the `session_start()` function to start the session, and then change `$_COOKIE` references to `$_SESSION`.

BULLET POINTS

- HTTP authentication is handy for restricting access to individual pages, but it doesn't offer a good way to "log out" a user when they're finished accessing a page.
- Cookies let you store small pieces of data on the client (web browser), such as the log-in data for a user.
- All cookies have an expiration date, which can be far into the future or as near as the end of the browser session.
- To delete a cookie, you just set its expiration to a time in the past.
- Sessions offer similar storage as cookies but are stored on the server and, therefore, aren't subject to the same browser limitations, such as cookies being disabled.
- Session variables have a limited lifespan and are always destroyed once a session is over (for example, when the browser is closed).

there are no Dumb Questions

Q: **The `session_start()` function gets called in a lot of different places, even after a session has been started. Are multiple sessions being created with each call to `session_start()`?**

A: No. The `session_start()` function doesn't just start a new session—it also taps into an existing session. So when a script calls `session_start()`, the function first checks to see if a session already exists by looking for the presence of a session ID. If no session exists, it generates a new session ID and creates the new session. Future calls to `session_start()` from within the same application will recognize the existing session and use it instead of creating another one.

Q: **So how does the session ID get stored? Is that where sessions sometimes use cookies?**

A: Yes. Even though session data gets stored on the server and, therefore, gains the benefit of being more secure and outside of the browser's control, there still has to be a mechanism for a script to know about the session data. This is what the session ID is for—it uniquely identifies a session and the data associated with it. This ID must somehow persist on the client in order for multiple pages to be part of the same session. One way this session ID persistence is carried out is through a cookie, meaning that the ID is stored in a cookie, which is then used to associate a script with a given session.

Q: **If sessions are dependent on cookies anyway, then what's the big deal about using them instead of cookies?**

A: Sessions are not entirely dependent on cookies. It's important to understand that cookies serve as an **optimization** for preserving the session ID across multiple scripts, not as a necessity. If cookies are disabled, the session ID gets passed from script to script through a URL, similar to how you've seen data passed in a GET request. So sessions can work perfectly fine without cookies. The specifics of how sessions react in response to cookies being disabled are controlled in the php.ini configuration file on the web server via the `session.use_cookies`, `session.use_only_cookies`, and `session.use_trans_sid` settings.

Q: **It still seems strange that sessions could use cookies when the whole point is that sessions are supposed to be better than cookies. What gives?**

A: While sessions do offer some clear benefits over cookies in certain scenarios, they don't necessarily have an either/or relationship with cookies. Sessions certainly have the benefit of being stored on the server instead of the client, which makes them more secure and dependable. So if you ever need to store sensitive data persistently, then a session variable would provide more security than a cookie. Sessions are also capable of storing larger amounts of data than cookies. So there are clear advantages to using sessions regardless of whether cookies are available.

For the purposes of Mismatch, sessions offer a convenient server-side solution for storing log-in data. For users who have cookies enabled, sessions provide improved security and reliability while still using cookies as an optimization. And in the case of users who don't have cookies enabled, sessions can still work by passing the session ID through a URL, foregoing cookies altogether.

Complete the session transformation

Even though the different parts of Mismatch affected by sessions use them to accomplish different things, the scripts ultimately require similar changes in making the migration from cookies to sessions. For one, they all must call the `session_start()` function to get rolling with sessions initially. Beyond that, all of the changes involve moving from the `$_COOKIE` superglobal to the `$_SESSION` superglobal, which is responsible for storing session variables.

```
<?php
  session_start();
?>
```

All of the session-powered scripts start out with a call to session_start() to get the session up and running.

```
// If the user isn't logged in, try to log them in
if (!isset($_SESSION['user_id'])) {
  if (isset($_POST['submit'])) {
    // Connect to the database
    $dbc = mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

    // Grab the user-entered log-in data
    $user_username = mysqli_real_escape_string($dbc, trim($_POST['username']));
    $user_password = mysqli_real_escape_string($dbc, trim($_POST['password']));

    if (!empty($user_username) && !empty($user_password)) {
      // Look up the username and password in the database
      $query = "SELECT user_id, username FROM mismatch_user WHERE username = '$user_username' AND " .
        "password = SHA('$user_password')";
      $data = mysqli_query($dbc, $query);

      if (mysqli_num_rows($data) == 1) {
        // The log-in is OK so set the user ID and username session vars, and redirect to the home page
        $row = mysqli_fetch_array($data);
        $_SESSION['user_id'] = $row['user_id'];
        $_SESSION['username'] = $row['username'];
        $home_url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
        header('Location: ' . $home_url);
      }
      else {
        // The username/password are incorrect so set an error message
        $error_msg = 'Sorry, you must enter a valid username and password to log in.';
      }
    }
```

login.php

The Log-In script uses sessions to remember the user ID and username for log-in persistence, and it does so by relying on the $_SESSION superglobal instead of $_COOKIE.

```
// Generate the navigation menu
if (isset($_SESSION['username'])) {
  echo '&#10084; <a href="viewprofile.php">View Profile</a><br />';
  echo '&#10084; <a href="editprofile.php">Edit Profile</a><br />';
  echo '&#10084; <a href="logout.php">Log Out (' . $_SESSION['username'] . ')</a>';
}
else {
  echo '&#10084; <a href="login.php">Log In</a><br />';
  echo '&#10084; <a href="signup.php">Sign Up</a>';
}

...

// Loop through the array of user data, formatting it as HTML
echo '<h4>Latest members:</h4>';
echo '<table>';
while ($row = mysqli_fetch_array($data)) {
  ...
  if (isset($_SESSION['user_id'])) {
    echo '<td><a href="viewprofile.php?user_id=' . $row['user_id'] . '">' .
      $row['first_name'] . '</a></td></tr>';
  }
  else {
    echo '<td>' . $row['first_name'] . '</td></tr>';
  }
}
echo '</table>';
```

The Mismatch home page uses the $_SESSION superglobal instead of $_COOKIE to access log-in data while generating the menu and choosing whether or not to provide a link to the "latest members" profiles.

index.php

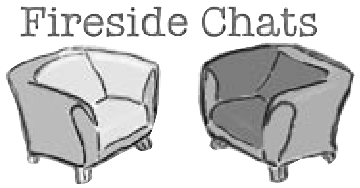
Similar to the Log-In and home pages, the Edit Profile script now uses $_SESSION to access log-in data instead of $_COOKIE.

```
  // Make sure the user is logged in before going any further.
if (!isset($_SESSION['user_id'])) {
  echo '<p class="login">Please <a href="login.php">log in</a> to access this page.</p>';
  exit();
}
else {
  echo('<p class="login">You are logged in as ' . $_SESSION['username'] .
    '. <a href="logout.php">Log out</a>.</p>');
}

...

    if (!empty($first_name) && !empty($last_name) && !empty($gender) && !empty($birthdate) &&
      !empty($city) && !empty($state)) {
      // Only set the picture column if there is a new picture
      if (!empty($new_picture)) {
        $query = "UPDATE mismatch_user SET first_name = '$first_name', last_name = '$last_name', " .
          "gender = '$gender', birthdate = '$birthdate', city = '$city', state = '$state', " .
          "picture = '$new_picture' WHERE user_id = '" . $_SESSION['user_id'] . "'";
      }
      else {
        $query = "UPDATE mismatch_user SET first_name = '$first_name', last_name = '$last_name', " .
          "gender = '$gender', birthdate = '$birthdate', city = '$city', state = '$state' " .
          "WHERE user_id = '" . $_SESSION['user_id'] . "'";
      }
      mysqli_query($dbc, $query);
```

editprofile.php

Although not shown, the View Profile script uses sessions in much the same way as Edit Profile.

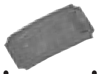
viewprofile.php



Tonight's talk: **Cookie and session variable get down and dirty about who has the best memory**

Cookie:

There's been a lot of talk around here among us cookies about what exactly goes on over there on the server. Rumor is you're trying to move in on our territory and steal data storage jobs. What gives?



Session variable:

Come on now, steal is a strong word. The truth is sometimes it just makes more sense to store data on the server.

Cookie: That doesn't make any sense to me. The browser is a perfectly good place to store data, and I'm just the guy to do it.

Session variable: What about when the user disables you?

Cookie: Uh, well, that's a completely different issue. And if the user decides to disable me, then clearly they don't have any need to store data.

Session variable: Not true. The user often doesn't even know a web application is storing data because in many cases, it is behind-the-scenes data, like a username. So if you're not available, they're left with nothing.

Cookie: So I suppose your answer is to store the data on the server? How convenient.

Session variable: Exactly. And the cool thing is that the user doesn't have the ability to disable anything on the server, so you don't have to worry about whether or not the data is really able to be stored.

Cookie: Alright, Einstein. Since you seem to have it all figured out, why is it that you still sometimes use me to store your precious little ID on the browser?

Session variable: Er, well, most people really don't know about that, so there's no need to get into it here. We can talk about that off the record. The important thing is that I'm always around, ready to store data on the server.

Cookie:

Come on, tell me how much you need me!

Session variable:

Alright, I will admit that from time to time I do lean on you a little to help me keep up with things across multiple pages. But I can get by without you if I need to.

Cookie:

Oh I know you can, but the truth is you'd rather not. And maybe deep down you really kinda like me.

Session variable:

Look, I don't have any problem with you. I just wish you were a little more secure. And you have that size limitation. You know, not every piece of persistent data is bite-sized.

Cookie:

Ah, so you're going to resort to picking on the little guy. Sure, I may not be able to store quite as much as you, and I'll admit that living on the client makes me a little less secure. But it sure is more exciting! And I have something you can only dream about.

Session variable:

Is that so? Do tell.

Cookie:

Well, all that storage space and security you're so proud of comes at a cost... a short lifespan! I didn't want to be the one to have to tell you, but your entire existence is hinging on a single browser session. I think that's how you got your name.

Session variable:

You mean you can go on living beyond a single session? How is that possible?!

Cookie:

It's simple. I don't die with a session, I just expire. So I can be set to live a long, full life, far beyond the whim of some click-happy web surfer who thinks it's cute to open and close the browser every chance he gets.

Session variable:

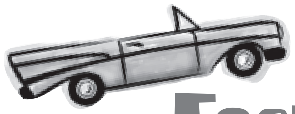
Wow. What a feeling that must be to experience immortality. My only hope is that some slacker scripter accidentally forgets to destroy me when he closes a session... but the browser will still do me in whenever it gets shut down.

Cookie:

Problem is, those same scripters often set my expiration to such a short period that I don't really get to experience the long life I truly deserve. I mean, I...

Session variable:

Hello? Are you there? Geez, expiration is harsh.

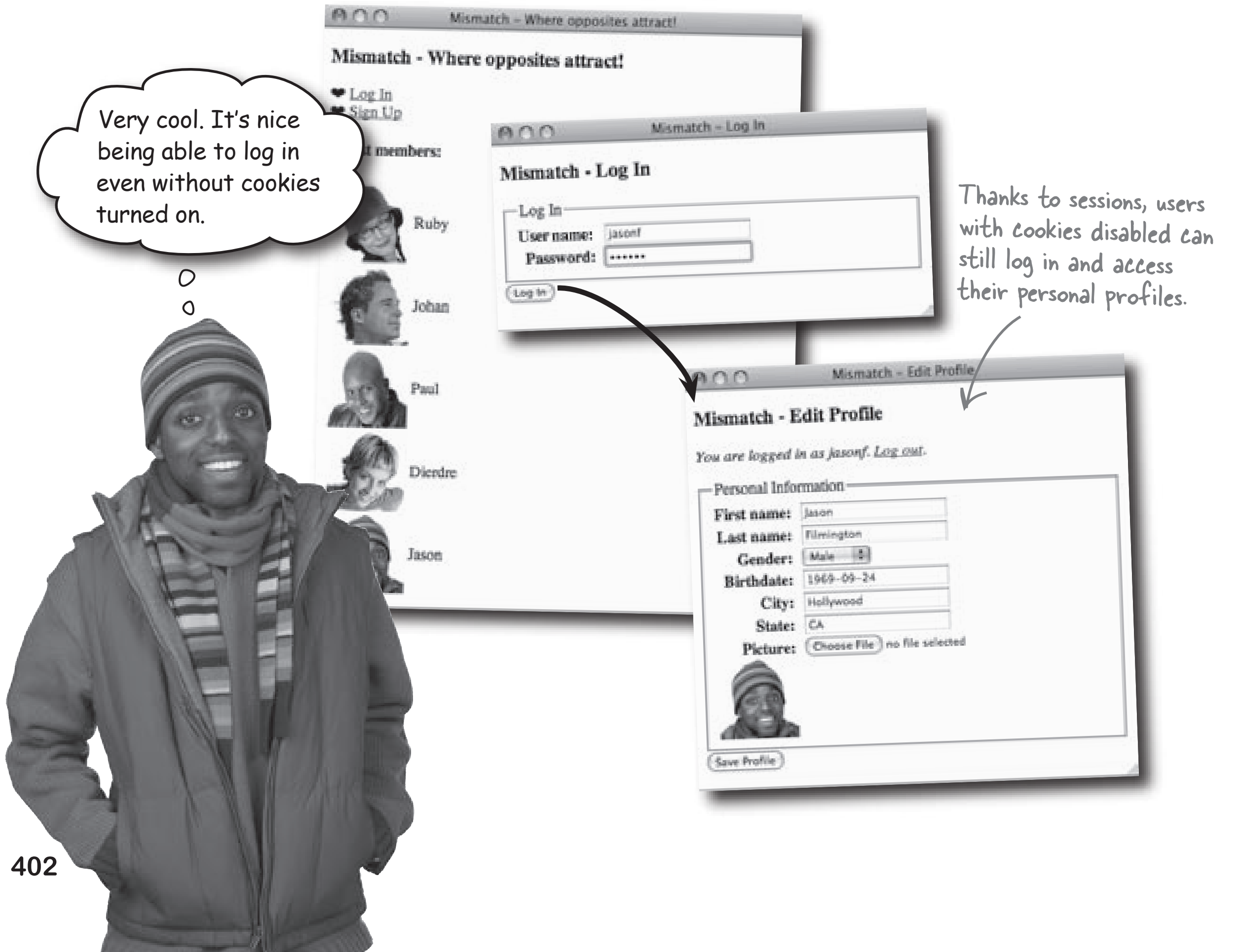


Test Drive

Change Mismatch to use sessions instead of cookies.

Modify the Mismatch scripts so that they use sessions instead of cookies to support log-in persistence (or download the scripts from the Head First Labs site at `www.headfirstlabs.com/books/hfphp`). The session modifications involve changes to the `index.php`, `login.php`, `logout.php`, `editprofile.php`, and `viewprofile.php` scripts, and primarily involve starting the session with a call to the `session_start()` function and changing `$_COOKIE` superglobal references to use `$_SESSION` instead.

Upload the scripts to your web server, and then open the main Mismatch page (`index.php`) in a web browser. Try logging in and out to make sure everything works the same as before. Unless you had cookies disabled earlier, you shouldn't notice any difference—that's a good thing!



Mismatch - Where opposites attract!

Mismatch - Where opposites attract!

❤ Log In
❤ Sign Up

t members:

Ruby
Johan
Paul
Dierdre
Jason

Mismatch - Log In

Mismatch - Log In

Log In
User name:	jasonf
Password:	•••••••

Log In

Mismatch - Edit Profile

Mismatch - Edit Profile

You are logged in as jasonf. Log out.

Personal Information
First name:	Jason
Last name:	Filmington
Gender:	Male
Birthdate:	1969-09-24
City:	Hollywood
State:	CA
Picture:	Choose File no file selected

Save Profile

Very cool. It's nice being able to log in even without cookies turned on.
Thanks to sessions, users with cookies disabled can still log in and access their personal profiles.

Watch it!

Sessions without cookies may not work if your PHP settings in `php.ini` aren't configured properly on the server.

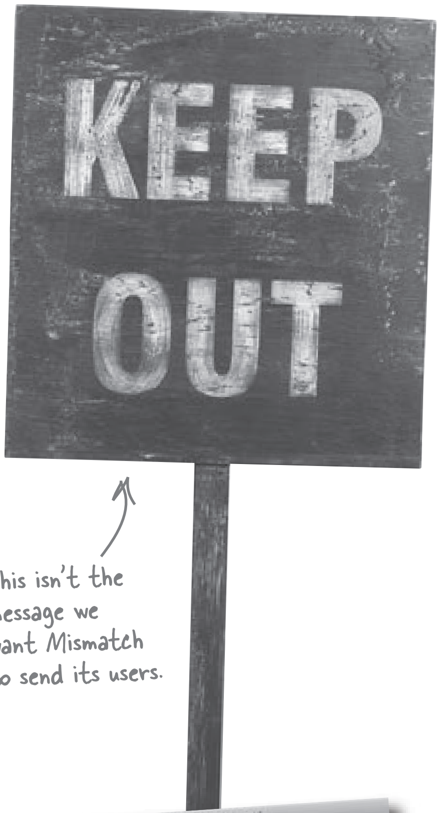
In order for sessions to work with cookies disabled, there needs to be another mechanism for passing the session ID among different pages. This mechanism involves appending the session ID to the URL of each page, which takes place automatically if the `session.use_trans_id` *setting is set to* `1` *(true) in the* `php.ini` *file on the server. If you don't have the ability to alter this file on your web server, you'll have to manually append the session ID to the URL of session pages if cookies are disabled with code like this:*

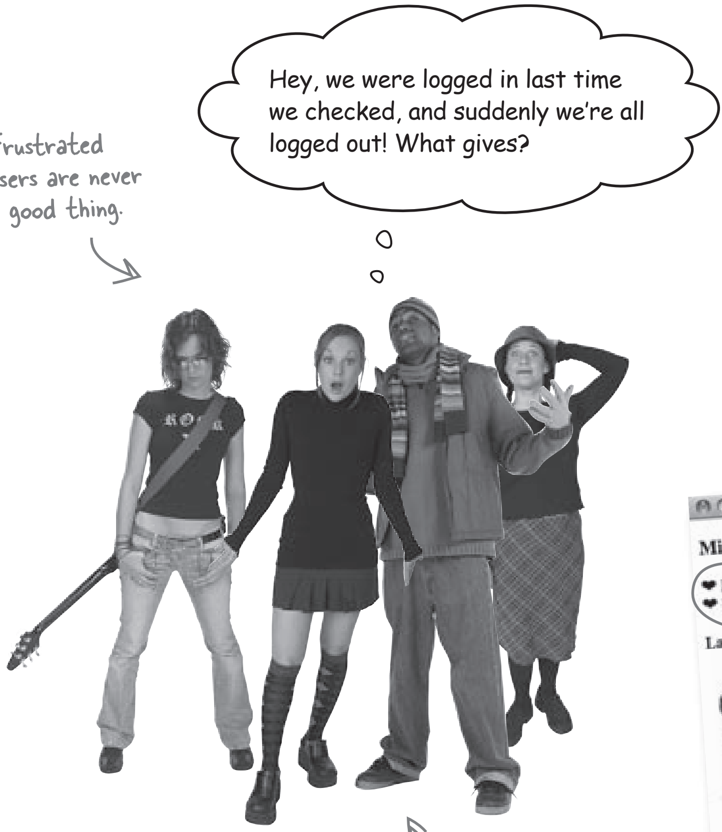
```
<a href="viewprofile.php?<?php echo SID; ?>">view your profile</a>
```

The SID superglobal holds the session ID, which is being passed along through the URL so that the View Profile page knows about the session.

Users aren't feeling welcome

Despite serving as a nice little improvement over cookies, something about the new session-powered Mismatch application isn't quite right. Several users have reported getting logged out of the application despite never clicking the "Log Out" link. The application doesn't exactly feel personal anymore... this is a big problem.

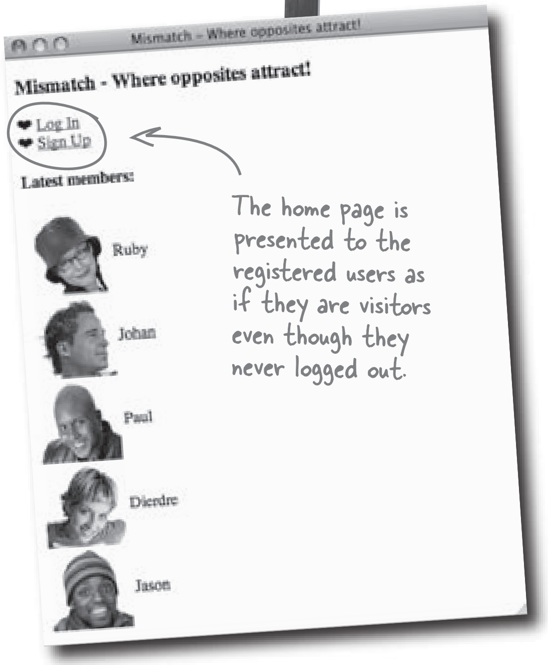



Hey, we were logged in last time we checked, and suddenly we're all logged out! What gives?

Frustrated users are never a good thing.

This isn't the message we want Mismatch to send its users.

Users are being logged out of Mismatch without ever clicking the "Log Out" link.



Mismatch - Where opposites attract!

❤ Log In
❤ Sign Up

Latest members:

Ruby
Johan
Paul
Dierdre
Jason

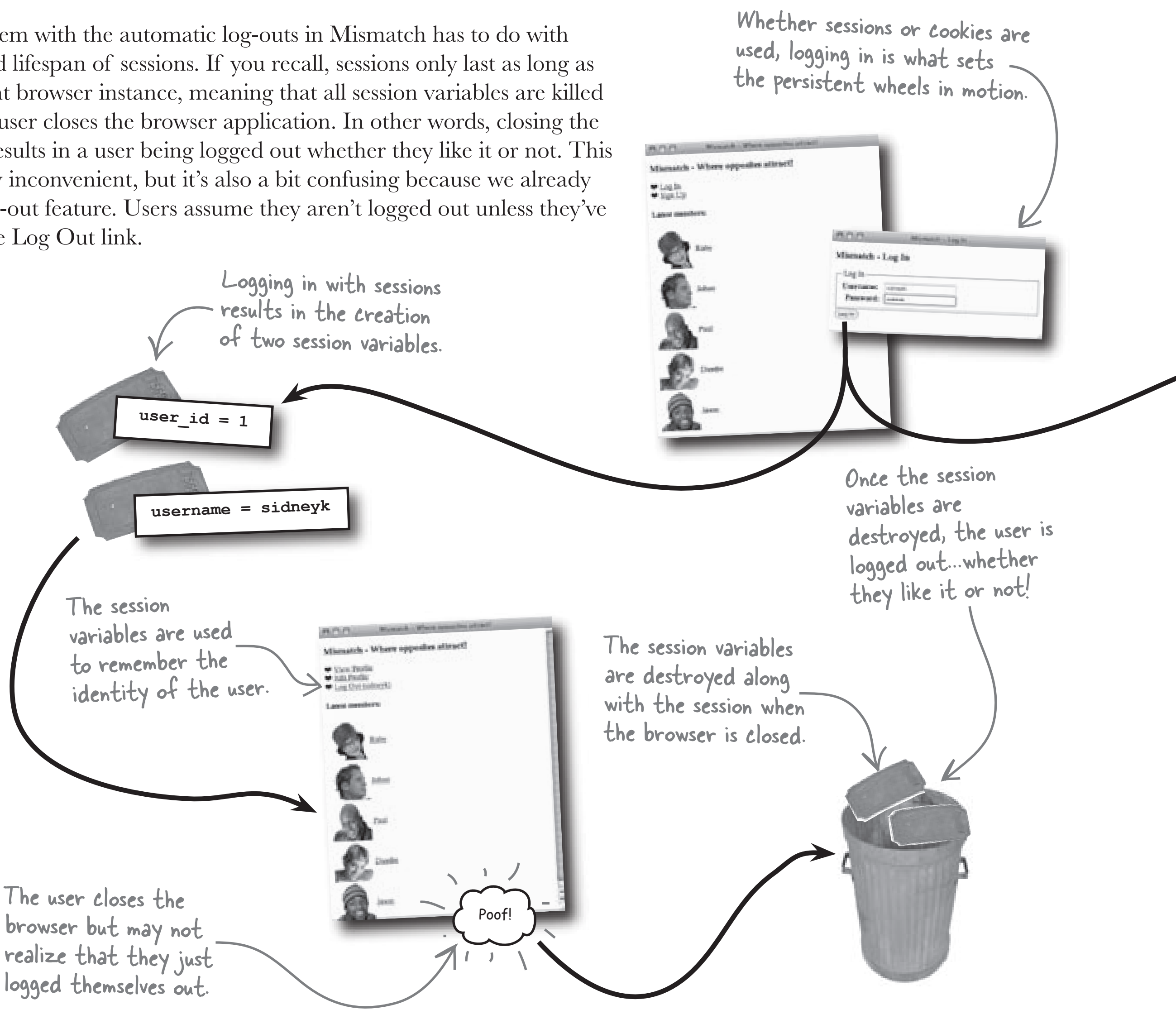
The home page is presented to the registered users as if they are visitors even though they never logged out.

BRAIN POWER

What do you think is causing users to be automatically logged out of Mismatch? Is it something they've done inadvertently?

Sessions are short-lived...

The problem with the automatic log-outs in Mismatch has to do with the limited lifespan of sessions. If you recall, sessions only last as long as the current browser instance, meaning that all session variables are killed when the user closes the browser application. In other words, closing the browser results in a user being logged out whether they like it or not. This is not only inconvenient, but it's also a bit confusing because we already have a log-out feature. Users assume they aren't logged out unless they've clicked the Log Out link.


Whether sessions or cookies are used, logging in is what sets the persistent wheels in motion.
Logging in with sessions results in the creation of two session variables.
user_id = 1
username = sidneyk
Once the session variables are destroyed, the user is logged out...whether they like it or not!
The session variables are used to remember the identity of the user.
The session variables are destroyed along with the session when the browser is closed.
The user closes the browser but may not realize that they just logged themselves out.
Poof!

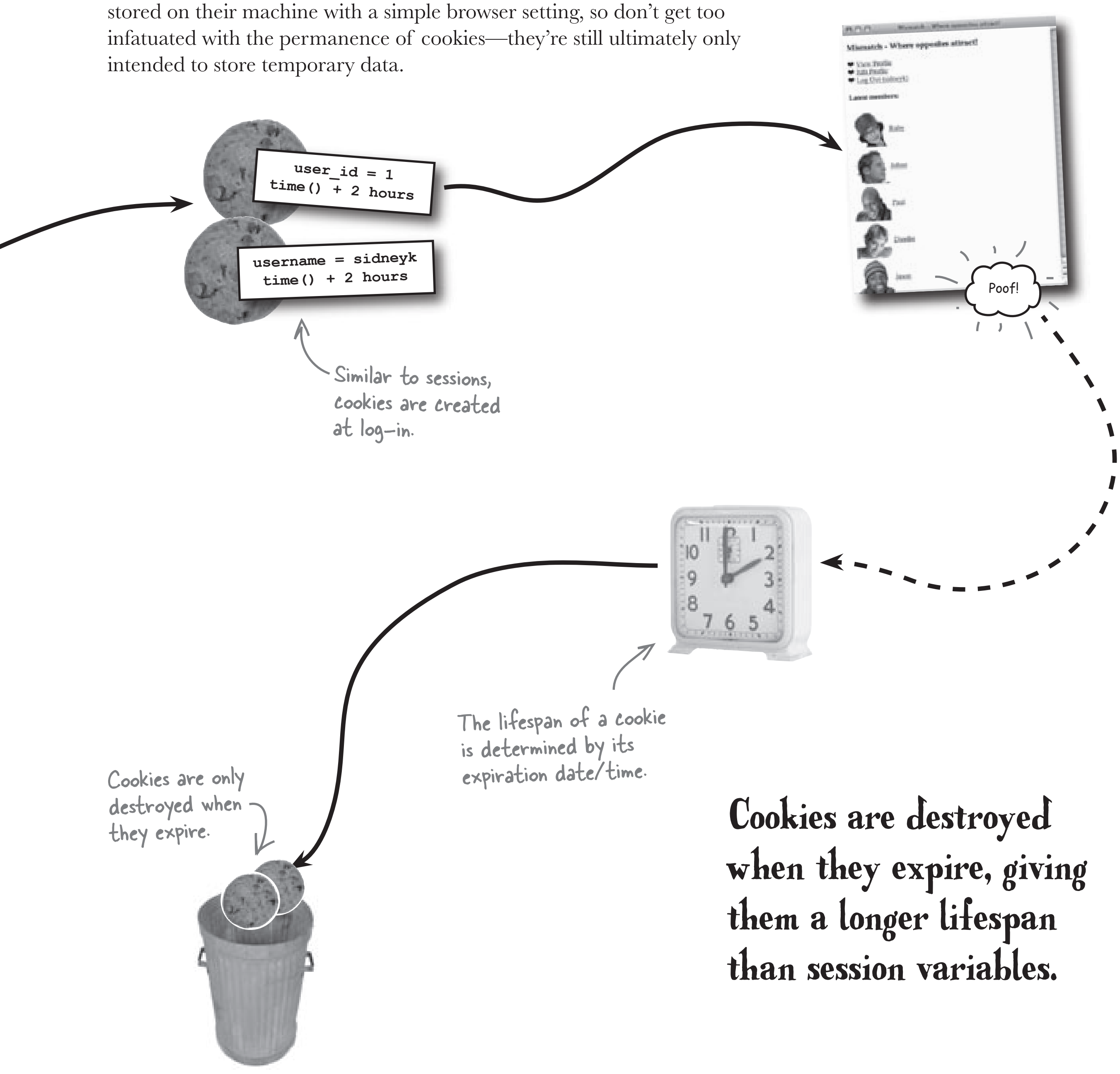
Even though you can destroy a session when you're finished with it, you can't prolong it beyond a browser instance. So sessions are more of a short-term storage solution than cookies, since cookies have an expiration date that can be set hours, days, months, or even years into the future. Does that mean sessions are inferior to cookies? No, not at all. But it does mean that sessions present a problem if you're trying to remember information beyond a single browser instance... such as log-in data!

Session variables are destroyed when the user ends a session by closing the browser.

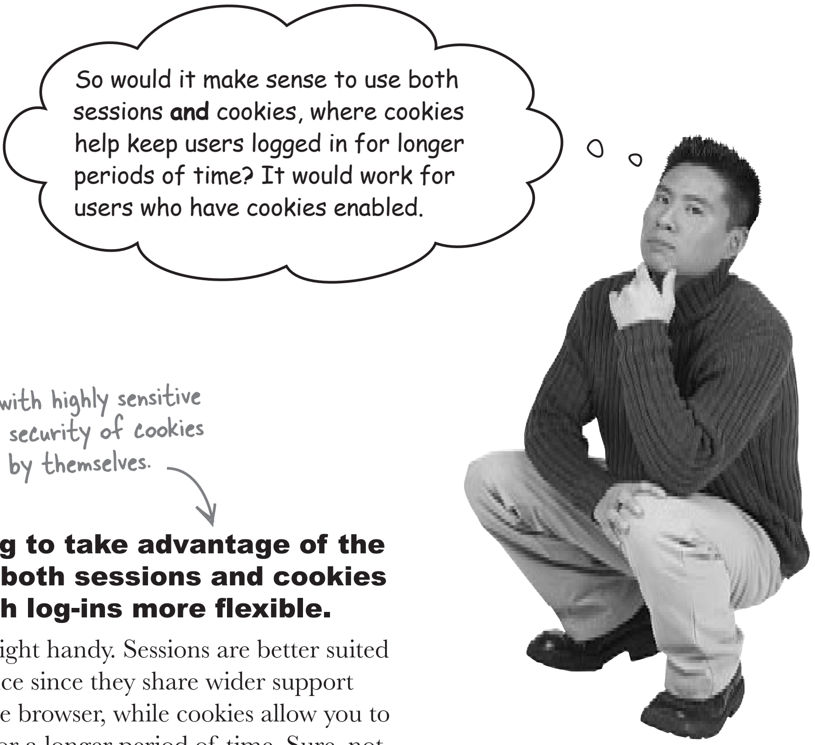
... but cookies can last forever!

Maybe not forever, but plenty long enough to outlast a session.

Unlike session variables, the lifespan of a cookie isn't tied to a browser instance, so cookies can live on and on, at least until their expiration date arrives. Problem is, users have the ability to destroy all of the cookies stored on their machine with a simple browser setting, so don't get too infatuated with the permanence of cookies—they're still ultimately only intended to store temporary data.


user_id = 1
time() + 2 hours
username = sidneyk
time() + 2 hours
Similar to sessions, cookies are created at log-in.
Mismatch - Where opposites attract!
Poof!
The lifespan of a cookie is determined by its expiration date/time.
Cookies are only destroyed when they expire.

Cookies are destroyed when they expire, giving them a longer lifespan than session variables.


So would it make sense to use both sessions **and** cookies, where cookies help keep users logged in for longer periods of time? It would work for users who have cookies enabled.

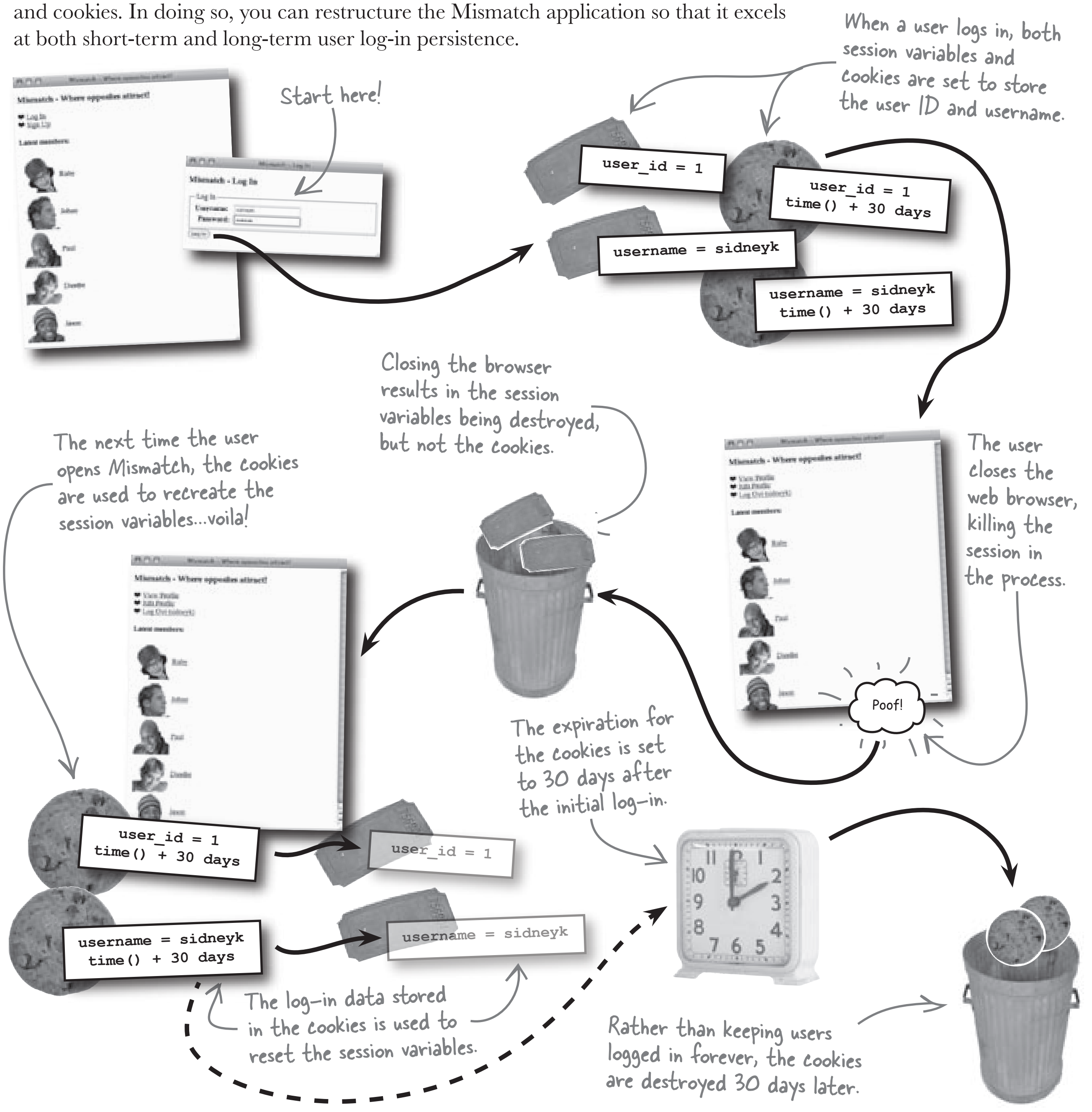
As long as you're not dealing with highly sensitive data, in which case, the weak security of cookies would argue for using sessions by themselves.

Yes, it's not wrong to take advantage of the unique assets of both sessions and cookies to make Mismatch log-ins more flexible.

In fact, it can be downright handy. Sessions are better suited for short-term persistence since they share wider support and aren't limited by the browser, while cookies allow you to remember log-in data for a longer period of time. Sure, not everyone will be able to benefit from the cookie improvement, but enough people will that it matters. Any time you can improve the user experience of a significant portion of your user base without detracting from others, it's a win.

Sessions + Cookies = Superior log-in persistence

For the ultimate in log-in persistence, you have to get more creative and combine all of what you've learned in this chapter to take advantage of the benefits of both sessions and cookies. In doing so, you can restructure the Mismatch application so that it excels at both short-term and long-term user log-in persistence.


Start here!
When a user logs in, both session variables and cookies are set to store the user ID and username.
user_id = 1
user_id = 1
time() + 30 days
username = sidneyk
username = sidneyk
time() + 30 days
Closing the browser results in the session variables being destroyed, but not the cookies.
The user closes the web browser, killing the session in the process.
Poof!
The next time the user opens Mismatch, the cookies are used to recreate the session variables...voila!
The expiration for the cookies is set to 30 days after the initial log-in.
user_id = 1
time() + 30 days
user_id = 1
username = sidneyk
time() + 30 days
username = sidneyk
The log-in data stored in the cookies is used to reset the session variables.
Rather than keeping users logged in forever, the cookies are destroyed 30 days later.

there are no Dumb Questions

Q: So is short-term vs. long-term persistence the reason to choose between sessions and cookies?

A: No. This happened to be the strategy that helped guide the design of the Mismatch application, but every application is different, and there are other aspects of sessions and cookies that often must be weighed. For example, the data stored in a session is more secure than the data stored in a cookie. So even if cookies are enabled and a cookie is being used solely to keep track of the session ID, the actual data stored in the session is more secure than if it was being stored directly in a cookie. The reason is because session data is stored on the server, making it very difficult for unprivileged users to access it. So if you're dealing with data that must be secure, sessions get the nod over cookies.

Q: What about the size of data? Does that play a role?

A: Yes. The size of the data matters as well. Sessions are capable of storing larger pieces of data than cookies, so that's another reason to lean toward sessions if you have the need to store data beyond a few simple text strings. Of course, a MySQL database is even better for storing large pieces of data, so make sure you don't get carried away even when working with sessions.

Q: So why would I choose a session or cookie over a MySQL database?

A: Convenience. It takes much more effort to store data in a database, and don't forget that databases are ideally suited for holding **permanent** data. Log-in data really isn't all that permanent in the grand scheme of things. That's where cookies and sessions enter the picture—they're better for data that you need to remember for a little while and then throw away.

PHP Magnets

The Mismatch application has been redesigned to use both sessions and cookies for the ultimate in user log-in persistence. Problem is, some of the code is missing. Use the session and cookie magnets to add back the missing code.

$_COOKIE $_COOKIE COOKIE $_COOKI $_COOKIE

login.php

```
...
if (mysqli_num_rows($data) == 1) {
  // The log-in is OK so set the user ID and username session vars (and cookies),
  // and redirect to the home page
  $row = mysqli_fetch_array($data);

  ....................['user_id'] = $row['user_id'];
  ....................['username'] = $row['username'];

  setcookie('user_id', $row['user_id'], time() + (60 * 60 * 24 * 30));    // expires in 30 days
  setcookie('username', $row['username'], time() + (60 * 60 * 24 * 30));  // expires in 30 days
  $home_url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
  header('Location: ' . $home_url);
}
...
```

logout.php

```
<?php
  // If the user is logged in, delete the session vars to log them out
  session_start();

  if (isset(....................['user_id'])) {

    // Delete the session vars by clearing the $_SESSION array

    .................... = array();

    // Delete the session cookie by setting its expiration to an hour ago (3600)

    if (isset(.................... [session_name()])) {
      setcookie(session_name(), '', time() - 3600);
    }

    // Destroy the session
    session_destroy();
  }

  // Delete the user ID and username cookies by setting their expirations to an hour ago (3600)
  setcookie('user_id', '', time() - 3600);
  setcookie('username', '', time() - 3600);
  ...
```

$_SESSION $_SESSION $_SESSION $_SESSION $_SESSION $_SESSION $_SESSION

index.php

```
<?php
  session_start();

  // If the session vars aren't set, try to set them with a cookie

  if (!isset(....................['user_id'])) {
    if (isset(....................['user_id']) && isset(....................['username'])) {
      ....................['user_id'] = ....................['user_id'];
      ....................['username'] = ....................['username'];
    }
  }
?>
...
```

PHP Magnets Solution

The Mismatch application has been redesigned to use both sessions and cookies for the ultimate in user log-in persistence. Problem is, some of the code is missing. Use the session and cookie magnets to add back the missing code.

```
...
if (mysqli_num_rows($data) == 1) {
  // The log-in is OK so set the user ID and username session vars (and cookies),
  // and redirect to the home page
  $row = mysqli_fetch_array($data);

  $_SESSION ['user_id'] = $row['user_id'];
  $_SESSION ['username'] = $row['username'];

  setcookie('user_id', $row['user_id'], time() + (60 * 60 * 24 * 30));   // expires in 30 days
  setcookie('username', $row['username'], time() + (60 * 60 * 24 * 30));  // expires in 30 days
  $home_url = 'http://' . $_SERVER['HTTP_HOST'] . dirname($_SERVER['PHP_SELF']) . '/index.php';
  header('Location: ' . $home_url);
}
...
```

The new cookies are set in addition to the session variables.

login.php

```
<?php
  // If the user is logged in, delete the session vars to log them out
  session_start();

  if (isset( $_SESSION ['user_id'])) {
    // Delete the session vars by clearing the $_SESSION array
    $_SESSION = array();

    // Delete the session cookie by setting its expiration to an hour ago (3600)
    if (isset( $_COOKIE [session_name()])) {
      setcookie(session_name(), '', time() - 3600);
    }

    // Destroy the session
    session_destroy();
  }

  // Delete the user ID and username cookies by setting their expirations to an hour ago (3600)
  setcookie('user_id', '', time() - 3600);
  setcookie('username', '', time() - 3600);
  ...
```

Logging out now requires deleting both the session cookie and the new log-in cookies.

logout.php

```
<?php
  session_start();

  // If the session vars aren't set, try to set them with a cookie
  if (!isset( $_SESSION ['user_id'])) {
    if (isset( $_COOKIE ['user_id']) && isset( $_COOKIE ['username'])) {
      $_SESSION ['user_id'] = $_COOKIE ['user_id'];
      $_SESSION ['username'] = $_COOKIE ['username'];
    }
  }
?>
...
```

If the user isn't logged in via the session, check to see if the cookies are set.

This same cookie/session code must also go in editprofile.php and viewprofile.php.

Set the session variables using the cookies.

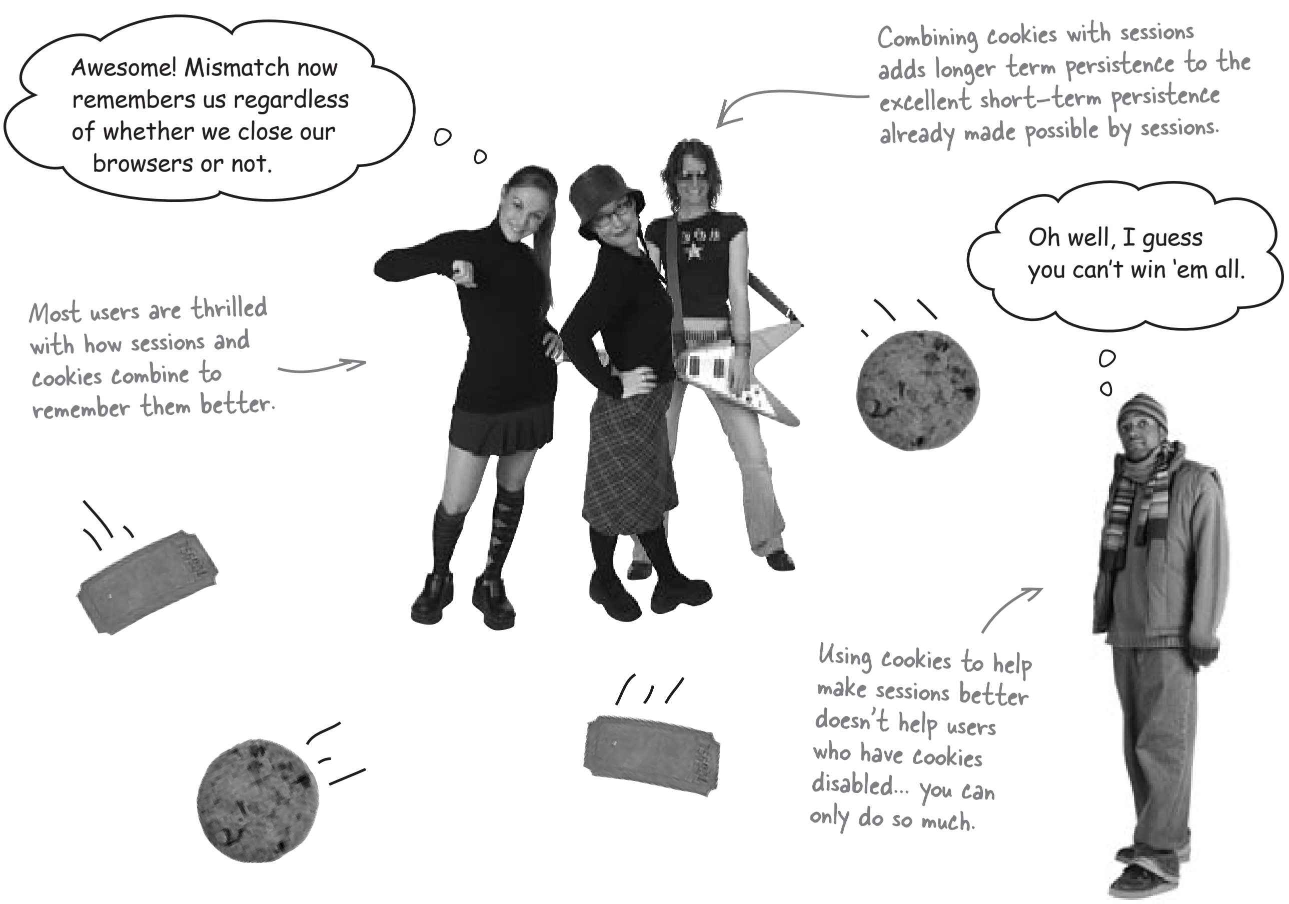
index.php

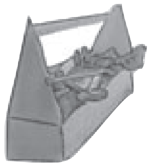
Test Drive

Change Mismatch to use both sessions and cookies.

Modify the Mismatch scripts so that they use both sessions and cookies to support log-in persistence (or download the scripts from the Head First Labs site at `www.headfirstlabs.com/books/hfphp`. This requires changes to the `index.php`, `login.php`, `logout.php`, `editprofile.php`, and `viewprofile.php` scripts.

Upload the scripts to your web server, and then open the main Mismatch page (`index.php`) in a web browser. Try logging in and then closing the web browser, which will cause the session variables to get destroyed. Re-open the main page and check to see if you're still logged in—cookies make this possible since they persist beyond a given browser session.


Awesome! Mismatch now remembers us regardless of whether we close our browsers or not.
Combining cookies with sessions adds longer term persistence to the excellent short-term persistence already made possible by sessions.
Oh well, I guess you can't win 'em all.
Most users are thrilled with how sessions and cookies combine to remember them better.
Using cookies to help make sessions better doesn't help users who have cookies disabled... you can only do so much.



Your PHP & MySQL Toolbox

You've covered quite a bit of new territory in building a user management system as part of the Mismatch application. Let's recap some of the highlights.

SHA(*value*)

This MySQL function encrypts a piece of text, resulting in a string of 40 hexadecimal characters. This function provides a great way to encrypt data that needs to remain unrecognizable within the database. It is a one-way encryption, however, meaning that there is no "decrypt" function.

setcookie()

This built-in PHP function is used to set a cookie on the browser, including an optional expiration date, after which the cookie is destroyed. If no expiration is provided, the cookie is deleted when the browser is closed.

$_COOKIE

This built-in PHP superglobal is used to access cookie data. It is an array, and each cookie is stored as an entry in the array. So accessing a cookie value involves specifying the name of the cookie as the array index.

session_destroy()

This built-in PHP function closes a session, and should be called when you're finished with a particular session. This function does not destroy session variables; however, so it's important to manually clean those up by clearing out the $_SESSION superglobal.

session_start()

This built-in PHP function starts a new session or re-starts a pre-existing session. You must call this function prior to accessing any session variables.

$_SESSION

This built-in PHP superglobal is used to access session data. It is an array, and each session variable is stored as an entry in the array. So accessing the value of a session variable involves specifying the name of the variable as the array index.

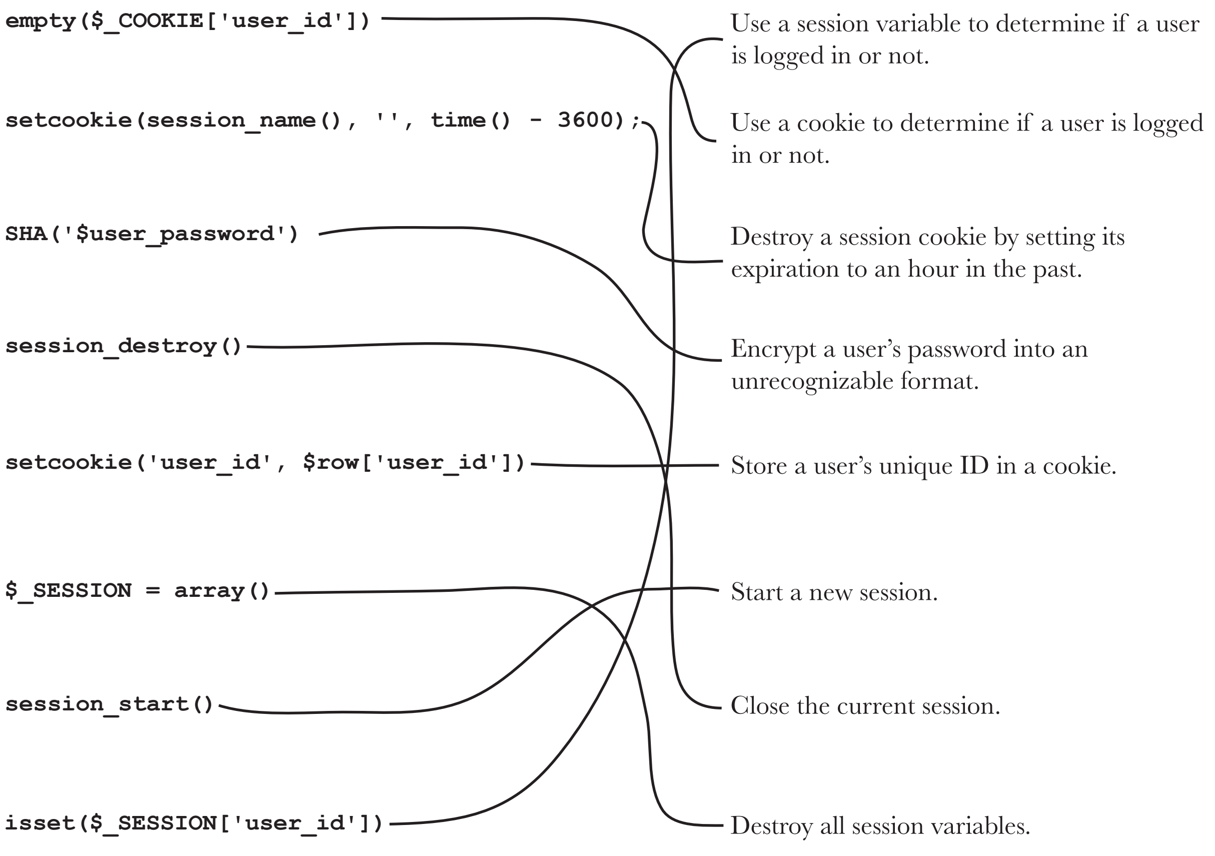
WHO DOES WHAT?

Several pieces of code from the Mismatch application have been pulled out, and we can't remember what they do. Draw lines connecting each piece of code with what it does.

PHP/MySQL Code	Description
`empty($_COOKIE['user_id'])`	Use a session variable to determine if a user is logged in or not.
`setcookie(session_name(), '', time() - 3600);`	Use a cookie to determine if a user is logged in or not.
`SHA('$user_password')`	Destroy a session cookie by setting its expiration to an hour in the past.
`session_destroy()`	Encrypt a user's password into an unrecognizable format.
`setcookie('user_id', $row['user_id'])`	Store a user's unique ID in a cookie.
`$_SESSION = array()`	Start a new session.
`session_start()`	Close the current session.
`isset($_SESSION['user_id'])`	Destroy all session variables.

WHO DOES WHAT?

Several pieces of code from the Mismatch application have been pulled out, and we can't remember what they do. Draw lines connecting each piece of code with what it does.



PHP/MySQL Code	Description
`empty($_COOKIE['user_id'])`	Use a session variable to determine if a user is logged in or not.
`setcookie(session_name(), '', time() - 3600);`	Use a cookie to determine if a user is logged in or not.
`SHA('$user_password')`	Destroy a session cookie by setting its expiration to an hour in the past.
`session_destroy()`	Encrypt a user's password into an unrecognizable format.
`setcookie('user_id', $row['user_id'])`	Store a user's unique ID in a cookie.
`$_SESSION = array()`	Start a new session.
`session_start()`	Close the current session.
`isset($_SESSION['user_id'])`	Destroy all session variables.