Caffeinated Crash Course in PHP

SIPB IAP 2009

Instructor: Steve Levine (MIT '11)

http://sipb-iap.scripts.mit.edu/2009/cccphp/
sipb-iap-caffeinatedphp@mit.edu
(or, just sjlevine@mit.edu)

About This Class

- Hi! I'm Steve Levine (MIT '11). Welcome to this caffeinated crash course in PHP!
- No, there isn't actually caffeine, sorry... :-)
- Feel free to ask questions (raise hands or yell, I don't really care) at any time
- If you're confused about something, chances are someone else is too and I didn't explain it clearly. Please ask questions!!

About This Class

- We'll be moving along briskly today!
- PHP is a big language, and we only have ~3 hrs
- I certainly won't expect you all to be PHP experts (why would you be here, then?)
- I will, however, assume that you have had at least some very basic exposure to programming (i.e., know what a variable is, have an understanding about what an if statement should do)
- A bit of knowledge of HTML might help, but certainly isn't required. You'll pick it up as we go.

Table of Contents

- What is PHP?
- How does PHP work?
- PHP syntax – control structures, functions, etc.
- Making Websites with PHP
- Using MySQL with PHP
- PHP Security
- If we have time – other stuff (cool extensions, etc.)?

About This Class

- Any questions before we start?

What is PHP?

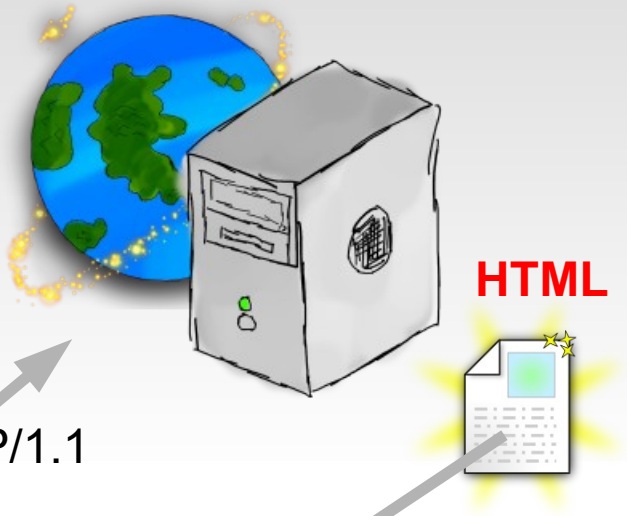
What is PHP?

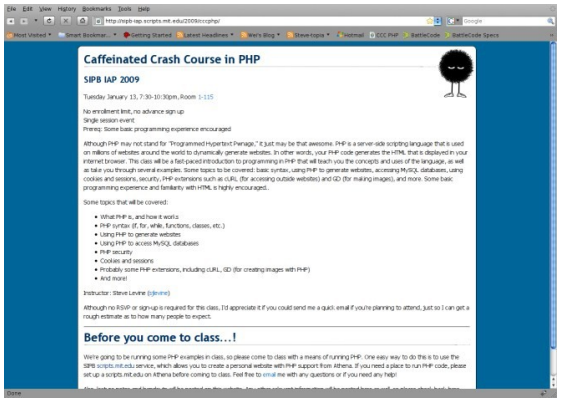
- PHP is a general-purpose programming language
- Although almost always used in practice on the web
- Stands for **P**HP: **H**ypertext **P**reprocessor (yes, a recursive acronym...)
- Interpreted language, not compiled
- Good at interacting with HTML
- Can add extensions to PHP for added functionality

What is PHP?

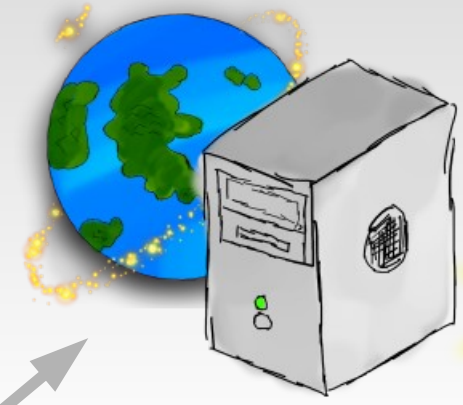
- PHP runs on a web-server ('server side' script), not on the client (i.e., your own) computer
- Dynamically generates the websites you see in your internet browser.
  - For regular, non-PHP websites: The HTML for some webpage is just sent straight to your browser with no modification.
  - With PHP however, websites files are first passed through PHP, and then sent to your browser (see next slide)

Viewing a Website: HTML





Viewing a Website: PHP

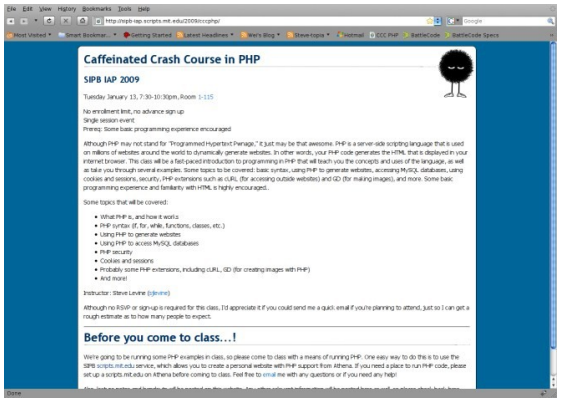


PHP

PHP interpreter

HTML

GET /index.**php** HTTP/1.1



How PHP Works on the Web

- When a webserver receives a request for a PHP page, it first passes it through the PHP interpreter.
- The PHP interpreter runs your PHP code, which outputs HTML.
- Your code can do whatever it wants to create the HTML – access a MySQL database, etc.
- This HTML is sent back to the client who made the request, and displayed in the web browser.

How PHP Works

- PHP files look like HTML files, except with special PHP tags to mark sections of code: <?php and ?>
- The PHP parser evaluates the code within this block, and replaces it with the resulting output
- Anything not in the PHP tag is ignored by the PHP parser and just outputted as HTML
- The filename extension is nearly always .php (this can be changed, though)

Example (example1.php)

PHP source code:

```
 1 <html>
 2 <head>I love PHP!</title>
 3 </head>
 4 <body>
 5
 6 <?php
 7         print "<p>Hello, World!</p>";
 8         print "<p>This is some PHP code. Note that it is embedded in a
regular HTML file!</p>";
 9 ?>
10
11 </body>
12 </html>
```

Example (example1.php)

Output from PHP parser:

```
1 <html>
2 <head>I love PHP!</title>
3 </head>
4 <body>
5
6 <p>Hello, World!</p><p>This is some PHP code. Note that it is embedded in
a regular HTML file!</p>
7 </body>
8 </html>
```

- Note that the <?php and ?> tags don't appear in the output.
- Only the output appears

How PHP Works

- Note that PHP outputs HTML-formatted data.
- The client computer will only see the resultant HTML output. It won't see the original PHP source code
- The client has no idea that the page is being generated dynamically on the server – all it does is display the HTML! **Abstraction**.

PHP Configuration

PHP Configuration

- PHP has a number of options that it can be configured with
- Default settings are in a file called php.ini (location dependent upon installation)
- Additionally, you can put a php.ini file in the folder of your webpage that applies just to that folder
- Can use php.ini to tell PHP to load certain extensions for your code (i.e., MySQLi, which will be discussed later)

PHP Configuration

- To see all of PHP's current settings: php_info() function
- Causes PHP to print a nice, pretty webpage that lists MANY settings
- Useful for debugging, but don't leave php_info() on your website. Hackers could try and find potential vulnerabilties to exploit
- Some example settings include: max_execution_time, memory_limit, etc.

PHP Configuration

```php
<?php

phpinfo();

?>
```

PHP Version 5.2.6

System	Linux bees-knees.mit.edu 2.6.27.9-73.fc9.x86_64 #1 SMP Tue Dec 16 14:54:03 EST 2008 x86_64
Build Date	May 8 2008 10:23:05
Configure Command	'./configure' '--build=x86_64-redhat-linux-gnu' '--host=x86_64-redhat-linux-gnu' '--target=x86_64-redhat-linux-gnu' '--program-prefix=' '--prefix=/usr' '--exec-prefix=/usr' '--bindir=/usr/bin' '--sbindir=/usr/sbin' '--sysconfdir=/etc' '--datadir=/usr/share' '--includedir=/usr/include' '--libdir=/usr/lib64' '--libexecdir=/usr/libexec' '--localstatedir=/var' '--sharedstatedir=/usr/com' '--mandir=/usr/share/man' '--infodir=/usr/share/info' '--cache-file=../config.cache' '--with-libdir=lib64' '--with-config-file-path=/etc' '--with-config-file-scan-dir=/etc/php.d' '--disable-debug' '--with-pic' '--disable-rpath' '--without-pear' '--with-bz2' '--with-curl' '--with-exec-dir=/usr/bin' '--with-freetype-dir=/usr' '--with-png-dir=/usr' '--enable-gd-native-ttf' '--without-gdbm' '--with-gettext' '--with-gmp' '--with-iconv' '--with-jpeg-dir=/usr' '--with-openssl' '--with-png' '--with-expat-dir=/usr' '--with-pcre-regex=/usr' '--with-zlib' '--with-layout=GNU' '--enable-exif' '--enable-ftp' '--enable-magic-quotes' '--enable-sockets' '--enable-sysvsem' '--enable-sysvshm' '--enable-sysvmsg' '--enable-track-vars' '--enable-trans-sid' '--enable-yp' '--enable-wddx' '--with-kerberos' '--enable-ucd-snmp-hack' '--with-unixODBC=shared,/usr' '--enable-memory-limit' '--enable-shmop' '--enable-calendar' '--enable-dbx' '--enable-dio' '--without-mime-magic' '--without-sqlite' '--with-libxml-dir=/usr' '--with-xml' '--with-system-tzdata' '--enable-force-cgi-redirect' '--enable-pcntl' '--with-imap=shared' '--with-imap-ssl' '--enable-mbstring=shared' '--enable-mbstr-enc-trans' '--enable-mbregex' '--with-ncurses=shared' '--with-gd=shared' '--enable-bcmath=shared' '--enable-dba=shared' '--with-db4=/usr' '--with-xmlrpc=shared' '--with-ldap=shared' '--with-ldap-sasl' '--with-mysql=shared,/usr' '--with-mysqli=shared,/usr/bin/mysql_config' '--enable-dom=shared' '--with-dom-xslt=/usr' '--with-dom-exslt=/usr' '--with-pgsql=shared' '--with-snmp=shared,/usr' '--enable-soap=shared' '--with-xsl=shared,/usr' '--enable-xmlreader=shared' '--enable-xmlwriter=shared' '--enable-fastcgi' '--enable-pdo=shared' '--with-pdo-odbc=shared,unixODBC,/usr' '--with-pdo-mysql=shared,/usr' '--with-pdo-pgsql=shared,/usr' '--with-pdo-sqlite=shared,/usr' '--enable-json=shared' '--enable-zip=shared' '--with-readline' '--enable-dbase=shared' '--with-pspell=shared' '--with-mcrypt=shared,/usr' '--with-mhash=shared,/usr' '--with-tidy=shared,/usr' '--with-mssql=shared,/usr'
Server API	CGI/FastCGI
Virtual Directory Support	disabled
Configuration	/etc

PHP Configuration

- Other way to set PHP options: from the script itself
- The ini_set() and ini_get() functions allow you to get and set configuration options
- Ex: `ini_set('display_errors','On');`

PHP Syntax

Syntax - Overview

- Now that we've finally covered PHP configuration, let's talk about programming!
- PHP syntax is similar to C/C++/Java, but it is loose with types like Python
- PHP always lies between <?php and ?> tags. Anything not between these tags will just be outputted directly from the parser and not evaluated
- End lines with a ; just like Java, C, and C++
- PHP is case sensitive (like most other languages)

Syntax – Variables

- Like other programming
- languages, PHP has variables
- Variables hold values
- Unlike most other programming languages, all PHP variables must begin with a dollar-sign ($)
- Ex. $username, $color, etc.
- Can hold many types of values: integers, decimals, strings of text, classes

Syntax - Variables

- Don't need to declare variables before you use them
- Variables can hold values of different types at different times
  - I.e., $var = 4; Later on: $var = 'This is text';
- Although being loose with types like this can be useful, it also opens the door for some common coding errors

Syntax – Assignment

- Use the = operator to assign values to variables

```
// Strings
$yummy = "potato";



// Number variables
$c = 3;
print "<p>" . $c/2 . "</p>"; // You don't have to worry about
                            // integers/decimals in PHP.
```

Syntax - Comments

- PHP supports C-style and C++-style comments

```
3 // This is a comment. It isn't interpreted by
4 // the PHP compiler.
5
6 /* This is
7    also a comment. Just like in C. */
```

Syntax - Expressions

- Expressions evaluate to values in PHP.

```
31 // Expressions example
32 $c = 3.141592653;
33
34 $c + 1;      // Evaluates to 4.141592653
35 2*($c + 1); // Evaluates to 8.283185306
36 // Assignments also evaluate to things
37 $c = 5; // Evaluates to 5
```

Syntax - Operators

- Common operators:
  - + Addition
  - - Subtraction
  - * Multiplication
  - / Divide
  - % Modulus
  - . String Concatenation
  - -> Member field/function (discussed later)
  - ( ) Parenthesis
  - == Equal to
  - != Not equal to
  - >, >= Greater than
  - <, <= Less than
  - ! Not
  - and (&&)
  - or (||)
  - xor

Syntax – String Quoting

- PHP has two ways to quote strings – double quotes (”) and single quotes (').
- When writing a string, you must put quotes at the beginning and at the end of the string.
- Single quotes are interpreted **literally** – exactly what is inside the quotes
- Double quotes can have values embedded in them – not interpreted literally

Syntax – String Quoting

- Here's an example:

```
22 // An example demonstrating the difference between
23 // single and double quoting.
24 $food = "potatoes";
25 $double_quoted = "I like to eat $food.";
26 $single_quoted = 'I like to eat $food.';
27 print "<p>$double_quoted</p>";
28 print "<p>$single_quoted</p>";
```

- The first example will print:

  I like to eat potatoes.

- The second will print:

  I like to eat $food.

Syntax - Arrays

- Arrays are collections of values
- PHP arrays sort of like 'dictionaries' in other languages such as Python
- Accessed via an **key.** Each key is associated with exactly one value
- A **key** may be an integer ('index'), a string, or any other type

Arrays - Indexing

- Here's an example that uses an integers as keys:

```
    // Arrays
63 $languages = array('PHP', 'Java', 'C++', 'Python');
64 print "<p>The first language is " . $languages[0] . "!</p>";
65 print "<p>The second language is " . $languages[1] . "!</p>";
```

- Prints:
  - The first language is PHP!
  - The second language is Java!

Arrays – Adding New Values

- To add a value to the end of an array:

```
74 // Adding to an array example
75 $languages[] = "Perl";
```

- Arrays, unlike languages such as Java, C, and C++, can be made arbitrarily long using this technique
- However, doing this a lot is ***slow***

Syntax - if

- If statements are arguably one of the most important concepts in computer science!

```
    // If statement example
42  if($favorite_color == "blue") {
43          // This code will execute if
44          // $favorite_color is blue
45          // ...
46
47  } else if ($favorite_color == "red") {
48          // The favorite color is red.
49          // ...
50  } else {
51          // None of the above! The favorite
52          // color must be something else.
53          // ...
54  }
```

Syntax - if

- Note – the whatever you supply as the **condition** of the if statement will be converted by PHP to a boolean (either 'true' or 'false')
- Things that get converted to 'false'
  - The boolean value 'false' (trivial case)
  - The number 0
  - Zero-length strings
  - Null
  - Zero-length errors
- Other things converted to true

Syntax – for loop

- The for loop:

```
63 $languages = array('PHP', 'Java', 'C++', 'Python');
67 // For-loop example with an array
68 for($c = 0; $c < count($languages); $c++) {
69         $lang = $languages[$c];
70         print "<p>The language at index $c is $lang.</p>";
71 }
```

- Paramters: *Initialization, Looping condition, Increment condition*
- But, there's actually a better way to loop through arrays...

Syntax – foreach loop

- The foreach loop:

```
 79 // Example of using non-integer keys
 80 $languages = array('best' => 'PHP', 'tastiest' => 'Java',
'funkiest' => 'Scheme');
 81
 82
 83 // A for-each loop.
 84 foreach($languages as $adjective => $lang) {
 85         print "<p>The $adjective language is $lang.</p>";
 86 }
```

Syntax – foreach loop

- Or, if you don't really care about the keys:

```
88 // If we didn't care about the key, we could also say:
89 foreach($languages as $lang) {
90         print "<p>$lang is a programming language.</p>";
91 }
```

Syntax - functions

- Functions are very important! They help group your code together, save you time, and make your code better.
- Functions can return values, or not return values.
  - Ex. $num = count($arr);
  - Count is a function that returns the size of array $arr. This returned value is stored in the variable $num.
- To return a value, use the `return` statement.

Syntax - functions

- Declare a function with the `function` keyword

```
95 function hereIsAFunction() {
96         print "This is a function.";
97         print "It doesn't really do very much.";
98 }
```

- Note that this function doesn't return a value (it just returns things)
- Calling functions

Syntax - functions

- Functions can optionally accept **argument**, or values that allow a function to do what you want it to do
- Functions can call other functions
- Functions can call themselves ('recursion')

Syntax - functions

- Here are some examples:

```
101 function countDown($start) {
102         while ($start >= 0) {
103                 print "<p>$start...</p>";
104                 $start--;
105         }
106 }
107
108 function squareNumber($x) {
109         return $x * $x;
110 }
```

Some Useful Functions

- isset($var) – Returns true if $var has been set.
  - Examples:
  - isset($favorite_color)
  - isset($_GET['name']) will return true if $_GET['name'] has a value (namely, the browser has sent a 'name' parameter in the URL)
- explode($delimiter, $string) – breaks up $string into an array of substrings, separaring using $delimiter
- count()

Some Useful Functions

- print_r($array) – prints an array. Useful for debugging
- require_once($filename) – Include another file
- ini_set($setting, value) (discussed earlier)
- header($data) – If at the beginning, outputs HTTP header data
  - Can be used for redirects (i.e., to secure HTTPS pages)

Syntax – Variable Scope

- Variables used within a function are local only to that function
  - They can't be accessed outside of the function – they basically disappear after the function is called
- If you've declared a variable *outside* of a function and you want to use it *inside* of the function, use the `global` keyword

Syntax - Variable Scope

- Here is an example:

```
114 // Global variables example
115 $global_var = 4;
116 function foo() {
117         // Now we can use $global_var. Otherwise it would
118         // just be treated as any old local variable
119         global $global_var;
120         print $globalvar;
121 }
```

Syntax - classes

- Classes are the building block of object-oriented programming (OOP)
- I'm not going to go too in depth in this; just the basics (OOP is a pretty large subject!)
- Don't worry if you're confused and you haven't seen OOP before; I'm skipping over a lot of details

Syntax - Classes

- Classes are basically bundles of variables and functions that should logically be grouped together
- PHP classes start with the `class` keyword
- :: Scoping resolution operator – used to access a super-class and call it's functions
- -> operator use to access functions/fields of classes
- Fields and functions in a class can be public and private

Syntax - Classes

```
126 // Class example
127 class user {
128         public $ID;
129         public $first_name;
130         public $last_name;
131
132         function __construct($first, $last, $id) {
133                 $this->ID = $id;
134                 $this->first_name = $first;
135                 $this->last_name = $last;
136         }
137
138         function getName() {
139                 return $this->first_name . " " . $this->last_name;
140         }
141
142         function toString() {
143                 return $this->getName() . "(" . $this->ID . ")";
144         }
145 }
```

Syntax – Extend classes

```
147 class facebook_user extends user {
148         // Since this class extends user, it has everything that
149         // user has. However, override the toString function to do
150         // something else
151         public $network;
152 
153         // Provide a new constructor that uses the old one
154         function __construct($first, $last, $id, $net) {
155                 parent::__construct($first, $last, $id);
156                 $this->network = $net;
157         }
158 
159         // Override user::toString()
160         function toString() {
161                 return $this->getName() . "(" . $this->network . ")";
162         }
163 }
```

Syntax - Classes

- Classes can be used as follows:

```
165 $some_user = new user("Steve", "Levine", "123");
166 print "<p>".$some_user->toString()."</p>";
167
168 $another_user = new facebook_user("Steve", "Levine", "123",
"MIT");
169 print "<p>".$another_user->toString()."</p>";
```

- Note the use of the 'new' keyword in generating instances of classes
- Also, note the user of the '->' operator to access member functions

Syntax

- That's about it for syntax!
- Are there any questions?

Using PHP to make Websites!

PHP with Websites

- Although PHP is technically a general-purpose language, it is almost always used to output HTML, or at least web-related XML
- From the previous slides on syntax, you already actually know most of what you need to know to make websites!
- Just a few more modifications needed to effectively make a website!

PHP Magic Variables

- There are several useful PHP 'super global' arrays that are automatically created by PHP when you a view a website
- $_GET, $_POST, $_REQUEST, $_SERVER, $_COOKIE, and $_SESSION.
- I'll talk about $_COOKIE and $_SESSION later
- All of these are arrays that contain useful information that you can use in making your website

$_GET

- $_GET contains parameters passed to your script from the web-browser
- Unlike POST parameters, GET parameters are visible in your URL bar
  - http://example.com/index.php?key1=value1&key2=value2&...
- Within index.php, $_GET['key1'] == 'value1', and $_GET['key2'] == 'value2'.
- $_GET it? (haha... excuse me...)

$_POST

- $_POST works just like $_GET, except that parameters aren't passed in the URL.
- This is the recommended way to pass form data – so that way it won't be easily visible (and changeable!) by the user
- Accessing the $_POST variable works just like accessing the $_GET variable.

$_REQUEST

- The $_REQUEST variable is the union of $_GET, $_SET, and $_COOKIE (to be discussed later).
- It exists merely as a convenience, for situations in which you don't really care where the input comes from

PHP Cookies and Sessions

Cookies

- Cookies are little data files that websites can store on your computer
- They come in several different flavors: chocolate, raisin, butterscotch ... oh wait, wrong kind of cookie.
- Cookies are used to store information about the user, or the website state, on the computer
  - Ex., you could store a shopping cart on the user's computer, or a username, or authentication information.

Cookies

- Commonly use form:
  - `bool setcookie($name, $value, $expire)`
- Cookie information exchanges happens at the beginning of the HTTP protocol
  - Happens before any real data is sent
- As a result, the setcookie() method **MUST** be called at the very beginning of your PHP file, before anything else. No <html>, no whitespace, etc.

Cookies Example

```php
<?php
// Cookies happen at the beginning of the HTTP

// See if we've stored a cookie yet.
if (isset($_COOKIE['username'])) {
        // The cookie is set. This means that there is a non-
        // expired cookie, so the user was already here.
        $username = $_COOKIE['username'];
        print "<h1>Welcome back!</h1>";
        print "<p>I see you've been here before, $username!</p>";
} else {
        // The cookie is not set, so assume that the user
        // hasn't been here yet.

        // Check and see if the user just submitted.
        if (isset($_POST['username'])) {
                // The user just submitted. Store a cookie!
                $username = $_POST['username'];
                setcookie('username',$username, time() + 120);
                print "<h1>Nice to meet you!</h1>";
                print "<p>It is nice to meet you, $username!</p>";
        } else {
                // The user didn't submit, and we don't have a cookie.
                // Display a form.
                print "<h1>Hello, stranger!</h1>";
                print "<p>What is your name?</p>";
                print "<form method=\"POST\" action=\"cookie.php\">";
                print "Username: <input type=\"text\" name=\"username\" />";
                print "<input type=\"submit\" value=\"Hello!\" />";
                print "<form>";
        }
}

?>
```

Sessions

- Sessions are another way to store information about your websites' users, kind of like cookies.
- Unlike cookies however, sessions are stored on the server, not on the client's computer.
- This means that you can trust that sessions data you store about a user hasn't been tampered or altered, since you have control of it on the server.

Sessions

- Data that you store in sessions can be accessed across different pages in your website, making them very useful
- Very easy to use; PHP does most of the hard work

How Sessions Work

- There's a magic, super-global array called $_SESSION. It acts like $_GET, $_POST, and $_REQUEST
- To save information in a session, you set a string-valued key of $_SESSION to the data you want:

```
$_SESSION['name'] = 'William B. Rogers';
```

- After you set a session variable, it will be visible to all other pages in your website.

Sessions

- You can use sessions to implement a login-system
- Shopping carts
- Basically anything where you need to remember something about the user

Session Example

```php
<?php
// Sessions must also happen at the beginning of the HTTP
session_start();

if (isset($_SESSION['username'])) {
        // The cookie is set. This means that there is a non-
        // expired cookie, so the user was already here.
        $username = $_SESSION['username'];
        print "<h1>Welcome back!</h1>";
        print "<p>I see you've been here before, $username!</p>";
} else {
        // hasn't been here yet.

        // Check and see if the user just submitted.
        if (isset($_POST['username'])) {
                // The user just submitted. Set a session var!
                $username = $_POST['username'];
                $_SESSION['username'] = $username;
                print "<h1>Nice to meet you!</h1>";
                print "<p>It is nice to meet you, $username!</p>";
        } else {
                // The user didn't submit, and we don't have a cookie.
                // Display a form.
                print "<h1>Hello, stranger!</h1>";
                print "<p>What is your name?</p>";
                print "<form method=\"POST\" action=\"session.php\">";
                print "Username: <input type=\"text\" name=\"username\" />";
                print "<input type=\"submit\" value=\"Hello!\" />";
                print "<form>";
        }
}

?>
```

MySQL

About MySQL

- MySQL is a free database commonly used on websites to store information
- PHP supports accessing MySQL databases
- Very useful for generating websites
- You can store information about users, preferences, and many other kinds of information in MySQL databases
- Then, you can use this information in generating webpage's HTML!

About MySQL

- To use data stored in a MySQL, you must first connect to the database server (a program on the server that handles database requests).
- Connecting usually requires a username and password
- The MySQL server divides data into **databases**. Within each database is one or more **table**.
- The actual data records are stored in these tables. Tables have columns and rows.

About MySQL

- For example, here is a sample MySQL layout:
- (Connection to server sql.mit.edu, logged in with username 'sjlevine'):

→ **Database:** *sipbtest+cccphp*

→ **Table:** *people*

name	course	gender	favorite_programming_language
Steve	٦	Male	PHP
Alice	٢	Female	Java
Bob	٨	Male	C++
Susan	٣	Female	PHP
Betty	٦	Female	Python

MySQL

- There's a web-interface for administering databases called phpMyAdmin (hey look, it's written in PHP!)
  - Useful for setting up databases for your website
- Alternatively, if you're a command-line lover, you can use the mysql program on your server.

MySQL: Introducing MySQLi

- The current recommended way to access MySQL databases is via the MySQLi (MySQL *improved*) extension for PHP
- There is also a different extension called MySQL (not to be confused with MySQL**i**), but it's mainly for use with older versions of MySQL.
- You should probably use MySQLi.

Using MySQLi

- General strategy for accessing data from an already-existing database:
    1. Connect to the MySQL database
    2. Prepare your 'query' (the question you're asking the database)
    3. Actually execute your query
    4. Process the results (the answer from the database)

        (Repeat 2-4 as necessary)
    5. Close your connection to the database

Using MySQLi

- MySQLi comes with a bunch of tasty, object-oriented classes to accomplish all of these steps
- There are multiple ways in PHP to connect to a MySQL database; I'll show you one here (the recommended way).
- The way I'll show you uses classes, and makes it hard for MySQL injection attacks (will be described later)!

Using MySQLi

- 3 Main Classes:
  - MySQLi – represents a connection
  - MySQLi_STMT – represents a query
  - MySQLi_Result – represents the result of a query (You don't always need to use this)
- Easiest way to see these in action is to actually take a look at some code. First I'll show it all to you, then we'll go through it piece-by-piece

MySQLi

```php
<?php
// For debugging, I'll turn on error output
ini_set('display_errors','On');
error_reporting(E_ALL);

// I have my MySQL database username and password in
// a separate file, so import them.

require_once('private/mysql_info.php');

// Make a MySQLi class, representing a connection to the
// MySQL database. I will open the database 'sjlevine+cccphp'
$conn = new mysqli('sql.mit.edu',MYSQL_USER, MYSQL_PASSWORD, 'sjlevine+cccphp');
// See if we connected successfully
if (mysqli_connect_errno()) {
        print "<p>Couldn't connect to the MySQL server. Bummer!</p>";
        exit();
}

// If we get here, then we connected just dandily to the MySQL server.
// Prepare a query for the MySQL database
$query = $conn->prepare("SELECT * FROM people WHERE gender=?");

// Bind the parameters (correspond to the ?'s) that we're looking for
$gender = 'Male';
$query->bind_param('s',$gender);

// Actually execute the query
$query->execute();

// Now, process the results. First bind variables corresonding to columns in our results:
$query->bind_result($name, $course, $gender, $fav_prog_lang);
print "<table><tr><td>Name</td><td>Course</td><td>Gender</td><td>Favorite Programming Langauge</td></tr>";

// Now, continually fetch new results that match our query, putting the answer in
// the bound variables. Keep going until no more results to "fetch."
while($query->fetch()) {
        print "<tr><td>$name</td><td>$course</td><td>$gender</td><td>$fav_prog_lang</td></tr>";
}
print "</table>";
$conn->close();
?>
```

MySQLi - Connect

- Here we 1.) Connect

```
12 // I have my MySQL database username and password in
13 // a separate file, so import them.
14
15 require_once('private/mysql_info.php');
16
17 // Make a MySQLi class, representing a connection to the
18 // MySQL database. I will open the database 'sjlevine+cccphp'
19 $conn = new mysqli('sql.mit.edu',MYSQL_USER, MYSQL_PASSWORD,
'sjlevine+cccphp');
20 // See if we connected successfully
21 if (mysqli_connect_errno()) {
22         print "<p>Couldn't connect to MySQL. Bummer!</p>";
23         exit();
24 }
```

MySQLi – Prepare Query

- 2.) Prepare the query

```
26 // If we get here, we connected just dandily to MySQL server.
27 // Prepare a query for the MySQL database
28 $query = $conn->prepare("SELECT * FROM people WHERE gender=?");
29
30 // Bind the parameters (correspond to the ?'s) that we're
looking for
31 $gender = 'Male';
32 $query->bind_param('s',$gender);
```

- bind_param: Fills in all of the '?' in the query string with the values. The first argument is the type: s for string, i for int, etc.

MySQLi - Execute

- 3.) Actually execute the query
- The easiest part of all!

```
34 // Actually execute the query
35 $query->execute();
```

MySQLi – Process Results

- 4.) Process the results. Here I make a table

```
// Now, process the results. First bind variables corresonding to columns
in our results:
$query->bind_result($name, $course, $gender, $fav_prog_lang);
print "<table><tr><td>Name</td><td>Course</td><td>Gender</td><td>Favorite
    Programming Langauge</td></tr>";

// Now, continually fetch new results that match our query,
//putting the answer in
// the bound variables. Keep going until no more results to "fetch."
while($query->fetch()) {
          print "<tr><td>$name</td><td>$course</td><td>$gender</td>
    <td>$fav_prog_lang</td></tr>";
}
print "</table>";
```

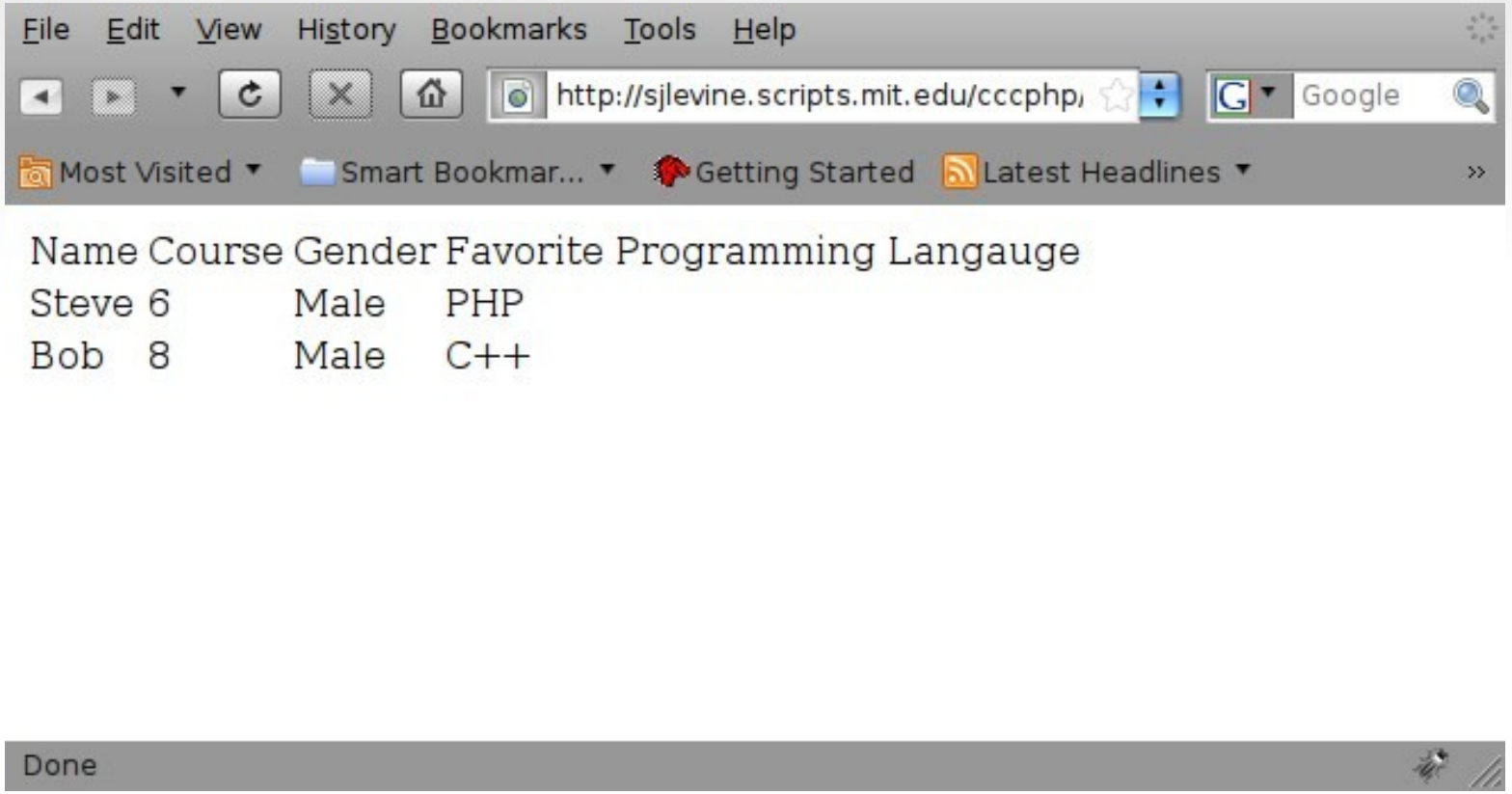
MySQLi - Close

- 5.) Close the connection to MySQL

```
// (W)rap it up, yo
$conn->close();
```

MySQLi

- And the output looks like:



PHP Security

Security

- Now is a good time to talk about PHP security, starting with MySQL security.
- You may not think security is that important ('No one will try to attack my simple little website...')
- You'd be surprised!!
- Attacks are all over the place, so you should program defensively. Especially on the web!

MySQL Injection Attacks

- Although MySQL is really cool, it can be prone to 'injection attacks' if you're not careful
- There are other ways to do queries in MySQLi:

```
$result = $conn->query("SELECT * FROM people WHERE name=\"$name\"");
```

- Although this may look shorter and sweeter than what I presented before, it opens up the door for injection attacks and probably shouldn't be used

MySQL Injection Attacks

```
$result = $conn->query("SELECT * FROM people WHERE name=\"$name\"");
```

- Assume that $name is retrieved directly from form data (i.e., `$name = $_POST['name'];`)
- This will work just fine if $name is something sane like 'Bob' or Jane'.
- But, consider the case of the malicious user who, instead of typing 'Bob', types the following (quotes included):

```
"; DROP people; SELECT * FROM foo WHERE name="
```

MySQL Injection Attacks

- Then, in this case, the MySQL query string will read:

```
SELECT * FROM people WHERE name=””; DROP people;
  SELECT * FROM foo WHERE name=””;
```

- Oh no, someone has just deleted our MySQL table! COMPROMISED!!
- Solution: Escape (i.e., encode quotes properly) any text used in a MySQL query.
- When you use bind_param in MySQLi, this happens automatically

XSS Attacks

- Another kind of attack: XSS
- XSS stands for 'Cross-Site Scripting'
- On many websites, data entered and entered by one user is stored in a database, and then later displayed to another user (ex. Forums, blogs, wiki's, etc.)
- This is usually okay, for regular data.
- But what if a malicious user attempts to enter HTML or JavaScript into the database (into a forum post)?

XSS Attacks

- Remember the table of people from before?

name	course	gender	favorite_programming_language
Steve	6	Male	PHP
Alice	2	Female	Java
Bob	8	Male	C++
Susan	3	Female	PHP
Betty	6	Female	Python

- Let's now add a hacker who stores something malicious in the MySQL database:

name	course	gender	favorite_programming_language
Steve	6	Male	PHP
Alice	2	Female	Java
Bob	8	Male	C++
Susan	3	Female	PHP
Betty	6	Female	Python
Jack	19	Male	</td></tr></table><script>alert("XSS!");</SCRIPT>

XSS Attacks

- If PHP outputs this data as is, it will be interpreted by the web-browser as HTML (just like anything else outputted from PHP)
- The hacker's entry will close off the HTML table we made, and start executing arbitrary JavaScript code
- This is very bad!!

XSS Attacks

- In this example, the code doesn't do much – just shows a messagebox.
- However, a hacker could potentially read and send-off your cookies (perhaps via HTTPRequests/AJAX)
- PHP Sessions are implemented by storing a session-ID cookie on the computer. If this is stolen, someone else can impersonate you on those websites!
  - Called 'Session hijacking!'

Ways to Prevent XSS

- XSS can be, in general, hard to prevent in all circumstances
- Hackers are clever!
- One way to make things a lot safer: use the PHP function htmlspecialchars()
  - Takes as input a string, and encodes it in a way such the content remains the same, but it will not be interpreted as HTML

Ways to Prevent XSS

- For example,
  `htmlspecialchars('<script>alert("XSS!");</script>', ENT_QUOTES) will output:`

  <script>alert("XSS!");</script>
- Note that this is 'HTML-ified'; the browser will interpret this as text, not as HTML
- Will prevent some types of XSS attacks

Security

- Importance of validation of any user input
- register_global_variables
- Don't reveal source code to users (or show error messages)!
  - Just asking for a hacker to come and exploit a vulnerability
-

Security: Protect Yourself!

- You may think that security is just about your users.
- It's very important to protect yourself, too!
- Consider this: You're the CEO of a multi-billion dollar corporation, and you have millions of credit-card numbers for your useres stored in your databases.
- And then some hacker comes along, gets into your databases, and steals all of the credit card numbers.
- You'll be in trouble!!

Security: Protect Yourself

- There is however a way to make this easier: encrypt all sensitive data stored in your databases
- PHP has several encryption extensions, including one called mcrypt that works well
- Another idea: instead of storing user's passwords in your database, store md5 hash values instead. That way, passwords cannot easily be stolen
- Unix-like systems (like Linux) do this. (Others probably do, too)

Security: Protect Yourself!

- Another way to protect yourself: Don't let your users see your PHP code
- This usually isn't an issue, since a default PHP installation won't let this happen
- However, if you turn on error display in PHP (using ini_set or the php.ini file), and if something goes wrong on your website, some information about your PHP code will be shown to your users
- So, for production services, turn off error output. Log to a server file instead.

Security: Protect Yourself!

- One last way to protect yourself: Turn register_global_variables off
- It is the default now in new versions of PHP, because it was a security risk
- register_global_variables used to be an alternate mechanism for $_GET and $_POST that was more dangerous

Security Summary

- Don't trust form data!! Always check verify it's sane
- If you don't use bind_param, always escape your MySQL commands to prevent injection attacks
- Use PHP functions like htmlspecialchars() to help prevent XSS attacks
- White-listing is better than black-listing
- Protect yourself! Encrypt sensitive information
- **Overall Moral**: Trust no user-data. Always be cautious

Other Cool Stuff: PHP Extensions

Other Cool PHP Stuff

- PHP can send emails, using the email() function
- The cURL extension lets you download other webpages, or talk to other websites
  - Nifty for getting price quotes from other websites, or getting directions from GoogleMaps in your PHP
- The GD extension allows PHP to generate images.
  - Useful for creating CAPTHA systems
- The mcrypt extension useful for encryption

Tips and Tricks

Common Mistakes:

- If, when you run code, you just see a blank page, you made a syntax mistake in your code
- Forgetting a semicolon on the end of lines
- () { } mismatches
- To see these (and other errors) instead of an annoying blank page, enable debugging and warning messages:

```
1 error_reporting(E_ALL);;
22ini_set('display_errors','On');
```

Other Resources

- php.net – PHP's official website. Extensive, useful documentation
- Wicked Cool PHP, by William Steinmetz and Brian Ward. Very useful (and wicked cool!) book
- Ask me! If you have any questions, feel free to shoot me an email, or just come ask me now.
- Good luck in you noble PHP adventures!
- Thanks for coming, everyone!