PHP
#DEAR FUTURE SELF,
#
# YOU'RE LOOKING AT THIS FILE BECAUSE
# THE PARSE FUNCTION FINALLY BROKE.
#
# IT'S NOT FIXABLE. YOU HAVE TO REWRITE IT.
# SINCERELY, PAST SELF

DEAR PAST SELF, IT'S KINDA
CREepy HOW YOU DO THAT.

#ALSO, IT'S PROBABLY AT LEAST
# 2013, DID YOU EVER TAKE
# THAT TRIP TO ICELAND?

STOP JUDGING ME!

http://xkcd.com/1421
PHP

- Personal Home Page
- PHP Hypertext Preprocessor
- Pretty Horrible Programming Language
PHP

"There are only two kinds of languages: the ones people complain about and the ones nobody uses"

—Bjarne Stroustrup (the creator of C++)

http://www.stroustrup.com/bs_faq.html#really-say-that

• PHP gets a lot of hate, but it is an easy to approach language that is the basis for a lot of very successful projects.
PHP: History

• 1994 - Rasmus Lerdorf wrote a series of Common Gateway Interface (CGI) binaries in C to maintain his homepage.

• 1995 - Lerdorf released “PHP Tools 1.0”

• 1997 - Zeev Suraski and Andi Gutmans rewrote the parser which formed the basis for PHP 3.

• 2000 - PHP 4 released

• 2004 - PHP 5 released, adding true objects, and an improved PHP Standard Library

• PHP 5.6 - 2014 We'll be working on this version

• PHP 7 - Just released December 2015

http://en.wikipedia.org/wiki/PHP
PHP Basics

• PHP has a REPL too
  • `php -a`
  • Except it doesn’t work on windows…
Variables

- All PHP variables are prefixed with a dollar sign: 

- Variable names must start with a letter or an underscore.

- Variable names can consist of letters, numbers, underscores, and the bytes 127 through 255.
Variables

• Like Javascript, variables in PHP are not typed.

• This doesn’t mean there are no types in PHP, it just means that a particular named variable is not tied to any one data type.
Type Checking

• Slight aside… Type Checking

• Instead of thinking about “Strongly Typed” or “Untyped” languages, think about when type checking is performed.

<table>
<thead>
<tr>
<th></th>
<th>Compile Time</th>
<th>Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Only</td>
<td>None</td>
</tr>
<tr>
<td>Java</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PHP</td>
<td>None</td>
<td>Only</td>
</tr>
<tr>
<td>Python</td>
<td>None</td>
<td>Only</td>
</tr>
</tbody>
</table>
Variables

- Variable names are case-sensitive. `$foo` and `$FOO` are different variables.

- Variables do not need to be declared. They spring magically into existence wherever they’re needed.

- This can be a good thing, and a bad thing.

```php
$isComplete = true;

if ($isComplete) {
    echo "All Done\n";
} else {
    echo "Not Done Yet\n";
}
```
var_dump()

- What’s in a variable?
- `var_dump` will show you the type and contents of any variable.
- Prints its output directly to STDOUT

```php
php > var_dump(3.1415);
float(3.1415)
```
Error Reporting

- You can change the level of error reporting.
- Config file, or at run time.
- Using `error_reporting( ... )` at runtime
Error Reporting

• Setting the error reporting level down to E_NOTICE can be very useful during development.

• Incredibly spammy in production!

```php
error_reporting(E_ERROR | E_WARNING | E_NOTICE | E_PARSE);

$isComplete = true;
if ($isComplete) {
    echo "All Done\n";
} else {
    echo "Not Done Yet\n";
}
```
PHP Structure

• PHP is sort of like the inverse of most languages when it comes to what gets output.

• Most languages have special features for printing things to the screen (or browser), and everything else is code.

• PHP has special features for defining where the code is, and everything else is output!
PHP Structure

- A Perl program and its output

```perl
#!/usr/bin/perl

use strict;

my $timestamp = time();

print "<!doctype html>
";
print "<html>
";
print "<head>
";
print "  <title>Hello World</title>
";
print "</head>
";
print "<body>
";
print "  <h1>Hello World: $timestamp</h1>
";
print "</body>
";
print "</html>
";
```
PHP Structure

• A PHP program and its output

```html
<!doctype html>
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  <h1>Hello World: <?php echo time(); ?></h1>
</body>
</html>
```
PHP Structure

• The PHP parsing engine only executes code the follows a `<?php` sequence.

• The closing portion `?>` is required to stop parsing of PHP code.

• The End of File (EOF) is treated the same as a closing `?>`
PHP Structure

• PHP Web Pages typically begin with HTML and have blocks of PHP code interspersed within it.

• PHP Code Files typically begin with an opening `<<?php` tag right on the first line of the file, and then have no closing `?>` tag, leaving the EOF to close the PHP code.

• This prevents stray characters outside of the `<<?php // code ?>` blocks from being sent as output
<!doctype html>
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  <h1>Hello World: <?php echo time(); ?></h1>
</body>
</html>

<?php
/**
 * Sample PHP Class
 * filename: php/structure1.php
 */
class first {
  private $foo;
  private $bar;

  public __construct() {
    $this->foo = "4";
    $this->bar = "2";
  }

  public answer() {
    return $this->foo . $this->bar;
  }
}

Web Page

Pure Code File
Web Servers and PHP

• The Web Server does a lot before PHP ever gets invoked.

• PHP does a lot of setup work before our code gets invoked.
Web Servers and PHP

Incoming HTTP Request

Web Server

- Looks for requested resource
- If resource is a PHP file, passes off to PHP
- Our code’s output text is returned to the web server
- Sets up a bunch of environment based on server and request
- Runs our code!

Response Sent To Client

PHP engine

- Response Sent To Client
Web Servers and PHP

• What’s in all that setup that the web server and PHP does before we ever get to our code?

• The Web Server may re-write the request path, add additional information, etc.

• PHP creates a set of “Super Global” variables which we have access to.
$_SERVER

- The $_SERVER superglobal contains a bunch of information about the request, the server, and our environment.

```php
<?php
print_r($_SERVER);
?>
```
$_GET

- The $_GET superglobal contains all variables passed in via the Query String portion of the URL
Query String

http://user:pass@example.com:80/path?query=yes#fragment

- Scheme
- Username
- Password
- Host
- Port
- Path
- Query String
- Fragment

- Key / Value Pairs
- URL Encoded Values
Forms

- Forms processing is one of the major uses for server side code.
- More HTML elements!!
- Example
Forms

- A bunch of different HTML form elements.

### Form HTML Elements

<table>
<thead>
<tr>
<th>HTML Element</th>
<th>attributes</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>type=&quot;text&quot;</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>type=&quot;checkbox&quot;</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>type=&quot;radio&quot;</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>type=&quot;submit&quot;</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>type=&quot;color&quot;</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>type=&quot;date&quot;</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>type=&quot;file&quot;</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>type=&quot;range&quot;</td>
<td></td>
</tr>
<tr>
<td>textarea</td>
<td></td>
<td></td>
</tr>
<tr>
<td>select</td>
<td></td>
<td>Option One</td>
</tr>
<tr>
<td>select</td>
<td>size=&quot;4&quot;</td>
<td></td>
</tr>
<tr>
<td>meter</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The `<form>` element defines an HTML form, and dictates where the form data is sent, and how.

- the `action` attribute says where to send this form’s data when the form is submitted.

- the `method` attribute says how to send the data, either with an HTTP GET command or POST.

```html
<form action="forms3.php" method="POST">
  <input type="text" name="input" size="20">
  <input type="submit" value="GO!">
</form>
```
• The `<input>` element is the basic, and most flexible of the form elements.

• Basic text input fields.

• Submit buttons.

• Password fields.

• Checkboxes and radio buttons.
$_GET and $_POST

- PHP provides us with these superglobal arrays
- User input
- Don’t Trust it!
$_POST

- The $_POST superglobal array contains all key/value pairs passed in via a POST HTTP request.
- Usually as the result of a Form submission
<?php
    $input = "";
    if (!empty($_POST)) {
        $input = print_r($_POST, true);
    }
?>

<html>
<head>
    <title>php/forms1.php</title>
</head>
<body>
    <h1>A Sample Form</h1>
    <form action="forms1.php" method="POST">
        <input type="text" name="input" size="20">
        <input type="submit" value="GO!">
    </form>
    <pre>
<?php
    echo $input;
?>
</pre>
</body>
</html>
Datatypes

- PHP only does type checking at *run time*.
- Variables have an internal type, but are aggressively type converted based on situation.
Datatypes

- Boolean
- Integer
- Float (Double)
- String
- Array
- Object
- Resource
- NULL

Built In Functions

• PHP has ‘em. Seriously, lots of them.

• Different from Java, or C, where the language defines very little in the way of functionality.

  • Functions are included manually via import statements.

• PHP defines hundreds of built-in functions, available in the global scope.
String Functions

- It's probably more useful to talk about datatypes as they relate to built-in functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>echo</td>
<td>Outputs a string</td>
</tr>
<tr>
<td>printf</td>
<td>Print a C style formatted string</td>
</tr>
<tr>
<td>strlen</td>
<td>Gets the length of a string</td>
</tr>
<tr>
<td>strtoupper</td>
<td>Returns an uppercase string</td>
</tr>
<tr>
<td>trim</td>
<td>Remove whitespace from the beginning and end of a string</td>
</tr>
<tr>
<td>ucfirst</td>
<td>Uppercase the first character of a string</td>
</tr>
</tbody>
</table>

nearly 100 more…

String Functions

```php
<?php
$s = "a long time ago...
";
echo $s;
echo strlen($s) . "\n";
echo strtoupper($s);
echo ucfirst($s);

$w = "    a padded string   ";
echo "'" . $w . "'\n";
echo "'" . trim($w) . "'\n";

```
String Escaped Characters

• Standard sort of escape mechanism for things like newlines and tabs.

• \n for a newline

• \t for a tab

```php
<?php
$s = "a long time ago...
";

echo $s;
echo strlen($s) . "\n";
echo strtoupper($s);
echo ucfirst($s);

$w = "    a padded string   ";
echo ""." . $w . ""." . "\n";
echo ""." . trim($w) . ""." . "\n";
```
String Concatenation

- The period . is our concatenation operator in PHP

```php
<?php
$s = "a long time ago...
";

echo $s;
echo strlen($s) . "\n";
echo strtoupper($s);
echo ucfirst($s);

$w = "    a padded string   ";

echo "'" . $w . "'\n";
echo "'" . trim($w) . "'\n";
```
Integers

- Formally, an integer in PHP is a member of the set:

\[ \mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\} \]

- \$a = 0;           // A decimal integer
- \$a = -123;        // A negative decimal integer
- \$a = 0123;        // An octal integer: 83
- \$a = 0x2A;        // A hexadecimal integer: 42
- \$a = 0b11111111;  // A binary integer: 255
Floats

• Floats, Doubles, Reals. PHP calls them all Floats

• $a = 3.1415;

• $a = 1.2e4;

• $a = 7E-10;

• All ways to define a float value
Arithmetic Operators

• You can, you know… do math, and stuff.

• PHP will convert an integer to a float before arithmetic

• $a = 1 + 2;       // int = int + int

• $a = 5 - 2.45;    // float = (int cast to float) + float

• $a = 5.5 / 0.5;   // float = float ÷ float
# Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$a + $b</code></td>
<td>Addition</td>
<td></td>
</tr>
<tr>
<td><code>$a - $b</code></td>
<td>Subtraction</td>
<td></td>
</tr>
<tr>
<td><code>-a</code></td>
<td>Negation</td>
<td></td>
</tr>
<tr>
<td><code>$a / $b</code></td>
<td>Division</td>
<td></td>
</tr>
<tr>
<td><code>$a * $b</code></td>
<td>Multiplication</td>
<td></td>
</tr>
<tr>
<td><code>$a % $b</code></td>
<td>Modulo</td>
<td></td>
</tr>
<tr>
<td><code>$a ** $b</code></td>
<td>Exponent ($a raised to the $b power)</td>
<td>New in PHP 5.6</td>
</tr>
</tbody>
</table>
Arrays

- Arrays in PHP are all *ordered Maps* under the hood.

- A map associates Keys and Values

- The basic array structure associates numerical keys (0, 1, 2, 3, 4) with their values.

- $a = \text{array}('a', 'b', 'c', 'd');$

```php
php > $a = array('a', 'b', 'c', 'd');
php > print_r($a);
Array
(
    [0] => a
    [1] => b
    [2] => c
    [3] => d
)
```
Arrays

- You can specify the keys for arrays using the `key => value` syntax.

- \$a = array('a' => 'A');

```php
php > $a = array('a' => 'A', 'b' => 'B');
php > print_r($a);
Array
{
    [a] => A
    [b] => B
}
php >
```
Arrays

- Any element who’s key is not explicitly set receives and auto-increment key.

- They start incrementing as they’re used, so $a[0]$ does not always indicate the first element of an array!
Arrays

• Array values can be any valid type.

• A given array can have values of many different types.

• You can have arrays as values in an array element, leading to complex nested structures.
Array Functions

- There are quite a lot of array functions!

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a[&quot;foo&quot;] = 1</td>
<td>Assigns the value 1 to the element with a key of “key”</td>
</tr>
<tr>
<td>array_push($a, 2)</td>
<td>Appends a new element to the end of the array with a value 2</td>
</tr>
<tr>
<td>$a[] = 2</td>
<td>Same as above. Shortcut for array_push()</td>
</tr>
<tr>
<td>array_pop($a)</td>
<td>Pops an element off the end of the array and returns its value.</td>
</tr>
<tr>
<td>array_keys($a)</td>
<td>Returns an array of all the keys for the array $a</td>
</tr>
<tr>
<td>sort($a)</td>
<td>Sort the elements in array $a by their keys.</td>
</tr>
<tr>
<td></td>
<td>There are 75 more!</td>
</tr>
</tbody>
</table>

print_r()

- Similar to var_dump(), print_r() will print the contents of an object to STDOUT
  - Can be made to return a string instead of printing to STDOUT
- It doesn’t report anything about data types
- Looks a little bit nicer
- Doesn’t append line breaks
print_r()

- Similar to var_dump(), print_r() will print the contents of an object to STDOUT

- Can be made to return a string instead of printing to STDOUT

- It doesn’t report as much about data types (still some though)

- Looks a little bit nicer

- Doesn’t append line breaks (except with arrays and objects)
Booleans

• Truth or dare! Well.. true or false

• Case insensitive

  • true TRUE True trUE  // All of these are true!

  • FALSE false fALsE  // Yup, all false
Booleans

• Most values in PHP are true, there are also many which are false.

• Some of the things that are false (there are others):
  
  • false (well… duh)
  
  • the integer value 0 // this one causes us problems later…
  
  • the float value 0.0
  
  • an empty string, i.e. ""
  
  • an array with zero elements
  
  • the special type NULL
  
  • any unset variable (think undefined from javascript)

Objects

- PHP gained true object oriented support in PHP 5.0
- Classes are declared and inherited
- Instances are created of classes via the `new` keyword.
Objects

- Objects can have properties, methods, constructors
- Supports single inheritance
- Supports public, private, protected visibility
- Lots more on objects as we go

```php
class foo {
    private $a = 1;
    private $b = 2;

    public function f() {
        return $this->a + $this->b;
    }
}
```
How long can you work on making a routine task more efficient before you're spending more time than you save? (across five years)

<table>
<thead>
<tr>
<th>How Much Time You Shave Off</th>
<th>How Often You Do the Task</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50/day</td>
</tr>
<tr>
<td>1 second</td>
<td>1 day</td>
</tr>
<tr>
<td>5 seconds</td>
<td>5 days</td>
</tr>
<tr>
<td>30 seconds</td>
<td>4 weeks</td>
</tr>
<tr>
<td>1 minute</td>
<td>8 weeks</td>
</tr>
<tr>
<td>5 minutes</td>
<td>9 months</td>
</tr>
<tr>
<td>30 minutes</td>
<td>6 months</td>
</tr>
<tr>
<td>1 hour</td>
<td>10 months</td>
</tr>
<tr>
<td>6 hours</td>
<td>2 months</td>
</tr>
<tr>
<td>1 day</td>
<td>8 weeks</td>
</tr>
</tbody>
</table>
Functions

- PHP began life as a procedural & function based language.
- Only added Objects late in life.
- PHP loves functions.
Anatomy of a Function

```php
function addTwo($a1, $a2)
{
    $sum = $a1 + $a2;
    return $sum;
}
```

- **name** of the function
- **argument** list (two args in this case)
- **function** keyword
- { } surround function statements
- can return a single thing
Functions

• Functions can be declared at the top level, or inside other functions

• Functions have global scope, no matter where they are declared

• Scope is different than namespaces, we won’t go into namespaces

```php
<?php
function foo() {
    function foo2() {
        return "bar!";
    }
    return foo2();
}
// Cannot call foo2() here, // it doesn't exist yet!
var_dump(foo());
// Now we can call foo2, its been // defined by calling foo()
var_dump(foo2());

string(4) "bar!"
string(4) "bar!"
```
File IO

• Reading from a local or remote file is pretty straightforward

• Writing to files is a bit more complicated
Reading from a File

• `file_get_contents(“path/to/file”)`

• Reads the entire contents of a file into memory and returns it as a string.

```php
<?php
$fileText = file_get_contents('file.txt');
echo $fileText;

• This example reads the entire contents of ‘file.txt’ into a variable called `$fileText`;```
Reading from a File

- `fopen('path/to/file', 'r')`
- Creates a file handle that can be referenced by further function calls.
- Can open files in read mode, or write mode.
- Doesn’t read the entire file into memory, so useful for working with large files, or for files where you don’t want everything, just specific pieces.
This is a text file. It has a few lines of text in it. Nothing much to see here.

```php
<?php
// Open a file handle to 'file.txt'
$fileHandle = fopen('file.txt', 'r');

// Read one line from the $fileHandle
$aLine = fgets($fileHandle);

// Read another line from the $fileHandle
$anotherLine = fgets($fileHandle);

echo $anotherLine;
```
Remote Files

- Most PHP file operations that take a path can accept any type of stream.
- Get the remote contents of a URL

```php
<?php
$webpage = file_get_contents("http://www.example.com");
echo $webpage;
```
Objects

- PHP 5 introduced full well thought out objects.

Diagram:
- Class Definition
  - Instantiation via `new` keyword
- Instance Object
  - Instantiation via `new` keyword
- Instance Object
  - Instantiation via `new` keyword
- Instance Object
Objects

- Classes are defined with the `class` keyword.
- New objects are created with the `new` keyword.

```php
<?php

class droid
{
    $type = "";

    function __construct($setType) {
        $this->type = $setType;
    }
}

$droid1 = new droid('protocol');
$droid2 = new droid('astromech');
```
```php
<?php

class droid
{
    $type = "";
    $name = "";

    function __construct($setType) {
        $this->type = $setType;
    }

    function setName($n) {
        $this->name = $n;
    }
}
```
PHP uses the `->` characters to do object access. Works pretty much the same way that a period `.` does in Java and Javascript.

```php
<?php
$droid1 = new droid('astromech');
$droid1->setName('R2D2');
echo $droid1->name;

```
Objects

• Special \texttt{\_construct()} method

• This method is called and passed any parameters when being instantiated via the \texttt{new} keyword.

```php
<?php

class droid
{
    $type = "";

    function \_construct($setType) {
        $this->type = $setType;
    }
}

$droid1 = new droid('protocol');
$droid2 = new droid('astromech');
```
Control Structures

• if .. else
• for
• foreach
• while
• continue
• break
if ... elseif ... else

- Basic branching logic.
- If an expression is TRUE, do one thing, otherwise do something else

```php
<?php
$expression = false;

if ($expression == true) {
    echo "Something is true.\n";
} else {
    echo "Something is false.\n";
}
```

if ... elseif ... else

• Can test multiple conditions with the elseif keyword

• It’s all one word – elseif not two words

• else-if

```php
<?php
$something = 'Green';

if ($something == 'Blue') {
    echo "Something is blue.\n";
} elseif ($something == 'Green') {
    echo "Something is green.\n";
} else {
    echo "Something is not Blue or Green.\n";
}
```
for (;;) {} 

• Basic C style for loop

```php
$colors = array("red", "orange", "yellow");

for($i = 0; $i < count($colors); $i++) {
    echo "Color: " . $colors[$i];
}
```
for (;;) {

for ($i = 0; $i < count($colors); $i++) {
...
}

foreach()

• Do something **for each** element in a collection

```php
<?php
$colors = array("red", "orange", "yellow");

foreach($colors as $c) {
    echo "Color: $c\n";
}
```
foreach()

- Works on all types of keys, not just numerical

```php
<?php
$person = array(
    "name"  => "Mark Fischer",
    "role"  => "Instructor"
);

foreach($colors as $key => $val) {
    echo "$key: $val
";
}
```
while()

• Keep doing something until a condition is false

```php
<?php
    $fh = fopen('somefile.txt', 'r');
    while ($line = fgets($fh)) {
        doWorkOn($line);
    }
    fclose($fh);
```
continue

- Stop this iteration of a loop, and go on to the next iteration

```php
<?php
$people = array(
    array("name" => "Mark Fischer","role" => "Instructor"),
    array("name" => "Margrit Mcintosh","role" => "Student"),
    array("name" => "Michale Hirst","role" => "Student"),
);

// Echo only students
foreach($people as $p) {
    if ($p['role'] == "Instructor") {
        continue;
    }

    echo $p['name'] . "\n";
}
```
break

- Stops all iterations of a loop

```php
<?php
$numbers = range(0, 100);

$numEvens = 0;
foreach($numbers as $n) {
    echo $n . "\n";
    if ((n % 2) == 0 ) {
        $numEvens++;
    }
    if ($numEvens >= 5) {
        break;
    }
}
```
Troubleshooting

• White Screen of Death
• Error Reporting
• Display Errors
Whenever I learn a new skill, I concoct elaborate fantasy scenarios where it lets me save the day.

Oh no! The killer must have followed her on vacation!

But to find them, we'd have to search through 200 MB of emails looking for something formatted like an address!

It's hopeless!

Everybody stand back.

I know regular expressions.

Boom! Tap tap.
PHP Sessions

• One way to solve the stateless nature of the Web
• Each Request is an isolated event
• How do we keep track of people between page views?
Web Servers and PHP

Incoming HTTP Request

Web Server

Looks for requested resource

Our code's output text is returned to the web server

If resource is a PHP file, passes off to PHP

Response Sent To Client

Sets up a bunch of environment based on server and request

Runs our code!
Cookies

• Web browsers allow sites to store small bits of information – cookies – locally on our computers.

• Cookies are sent to the browser as part of the HTTP response headers.

• Sent back to the server on subsequent requests.

• The server keeps track of who has which cookie ID, and can keep track of visitors.
Cookies

Web Application Creates and Stores Cookie Value and returns value in response headers

Initial Page Request

Browser Stores Cookie Value

Cookie Set

Web App sees cookies in a Request. Looks up the value locally, and reconnects visitor to state

Browser sends Cookies back to the same server

Response Sent

Subsequent Page Request
PHP Cookies

- PHP has a `setcookie()` function that handles the details of constructing a properly formatted `Set-Cookie` response header.

```php
// Set a new cookie
$value = "SomeValueString";
$cookieName = "CS337-Test-Cookie";
$expiration = time()+3600;
setcookie($cookieName, $value, $expiration);
```

http://localhost/cc337/p/cookies.php
PHP Sessions

• PHP has a session handling system built in.

• Based on cookies, and server-side file storage by default.

• Beginning a PHP session sets a cookie on the client.

• That cookie is then used to retrieve locally stored data from the server, and present it in the \$_SESSION superglobal.
<?php
session_start();

// If we're POSTing to this page, it's probably a form update
if (!empty($_POST)) {
    $newSavedString = $_POST['saveString'];
    $_SESSION['saveString'] = $newSavedString;
    // Redirect to the page via GET to fix the back button issue
    header('Location: sessions.php');
}

<!doctype html>
<html>
<head>
    <title>php/sessions.php</title>
</head>
<body>
    <section>
        <h2>Current Saved String: <?php echo $_SESSION['saveString']; ?></h2>
    </section>
    <section>
        <form action="sessions.php" method="POST">
            <input name="saveString" type="text">
            <input type="submit" value="Update Saved String">
        </form>
    </section>
</body>
</html>
Current Saved String: Our Saved String
PHP Sessions

- *MUST* call `session_start()` before sending ANY response to the browser.
- Once the server begins sending text back to the browser, all headers must be sent first.
- Since sessions depend on cookies, the cookie must be sent along with the response headers, before any content.

```php
<?php
session_start();

// If we're POSTing to this
if (!empty($_POST)) {
    $newSavedString = $_POST['saveString']
    $_SESSION['saveString'] = $newSavedString;

    // Redirect to the page
    header('Location: sessions.php');
}
?>

<!doctype html>
<html>
<head>
    <title>php/sessions.php</title>
</head>
<body>
    <section>
        <h2>Current Saved String: <?php echo $_SESSION['saveString']; ?></h2>
    </section>
    <section>
        <form action="sessions.php" method="POST">
            <input name="saveString" type="text">
        </form>
    </section>
</body>
</html>
```
PHP Sessions

• What Can I keep in \$_SESSION?

• Any \textit{serializable} value
  
  • Scalars (int, float, string, bool, etc)

  • Arrays – As long as all array elements are also serializable

  • Objects – Again, as long as all properties are serializable
PHP Sessions

• What isn’t allowed in $_SESSION?

• Mostly resources:
  • Open file handles
  • Network sockets
  • Streams

• Closures

• Some objects – Any objects with references to non-serializable things
Go Talk About MySQL