



The PHP Company

**Participant
Guide**



PHP II: Higher Structures

This page is a placeholder for online viewing... not to be printed.

Table of Contents

COURSE INTRODUCTION	i
Zend PHP II: Higher Structures Course.....	iii
MODULE 1: INTRODUCTION TO THE COURSE	3
Introduction	4
Pre-Requisite Knowledge for Taking this Course	5
MODULE 2 PHP SYNTAX REVIEW	6
Quotes	8
Comments	10
Special Characters	11
Data Types	12
Precedence and Definitions	15
Arrays	18
Operators	26
Conditionals and Loops	31
Functions	41
MODULE 3: PHP LANGUAGE CONCEPTS	48
Including Files	50
Globals	53
References	57
File System Basics	62
MODULE 4: CONFIGURING PHP	85
Setting Up a Development Environment.....	87
PHP.INI Settings	92
MODULE 5: REGULAR EXPRESSIONS	102
Characters and Symbols	105
RegEx Functions	108
Pattern Modifiers.....	117

MODULE 6: PHP WEB CONCEPTS	124
Server Communication	128
HTTP Headers	132
Output Buffering	142
Browser Caching.....	146
Cookies.....	154
Sessions	160
Email	167
Forms.....	176
MODULE 7: PHP OBJECT-ORIENTED PROGRAMMING	187
Classes	191
Static Context	200
Visibility (PPP)	204
Overriding Functions	211
Interfaces	217
Cloning.....	225
MODULE 8: PHP DATABASE BASICS	229
Relational Databases	233
SQL and MySQL	235
Connecting PHP and MySQL	238
PDO	240
Stored Procedures	250
Transactions.....	255
MODULE 9: CRITICAL ASPECTS OF BUILDING PHP APPLICATIONS	262
Testing and Debugging	265
Troubleshooting Procedures: Development	272
Troubleshooting Procedures: Deployment	275
phpDocumentor, phpDoc	276
Exceptions.....	283
Web Services.....	287
Performance Enhancements.....	301
Security and Validation	306
MODULE 10: PHP APPLICATION SUMMARY PROJECT	316
Summary Project Exercises	319
Class Review	320
Next Steps, Additional Resources.....	329

Introduction

Zend PHP II: Higher Structures Course--Objectives

The Zend PHP II: Higher Structures course is designed to further advance the PHP skills of participants who already know the basics of PHP syntax, language constructs, and web site functionality. This intermediate-level course builds upon knowledge gained in PHP I: Foundations. It utilizes a hands-on approach with numerous examples and practical exercises, primarily related to the course's Blog project, to enhance learning. You will also have the opportunity to use the Zend Studio IDE (Integrated Development Environment) to continue honing your coding skills utilizing best practices and effective tools.

The Zend Training Center allows you to code exercises live during the course with your instructor able to both see and coach your progress.

This course is offered online - with a live instructor - for 12 hours (6 two-hour sessions) or can be adapted to a classroom setting.

Experienced programmers in Procedural and OO languages should consult the specially designed "Quick Start: PHP for Experienced Programmers" course description before deciding to take this course. ILE programmers are coached to complete the "PHP I: Foundations" and "PHP II: Higher Structures" course sequence, as the language syntax they are used to is unique and quite different from PHP.

About Zend

Zend is the PHP company. Businesses utilizing PHP know Zend as the place to go for PHP expertise and sound technology solutions. Zend delivers premier web application platform products and services for PHP applications. With commercial products and services that enable developers and IT personnel to deliver business-critical PHP applications, Zend is taking the power of PHP to the enterprise.

Zend provides a suite of products that supports the entire PHP lifecycle, from development to production, of your business-critical PHP applications. Often, PHP programmers work in demanding environments, where they are called upon to fulfill more than one role. Zend products allow those who work in web application development - as well as related fields like system administration - to seamlessly utilize all their relevant features to rapidly produce a dynamic, robust web application in a stable, reliable environment.

Zend Platform

Setting the Standard in PHP Application Performance & Availability

Zend Platform is the all-in-one production environment that ensures your PHP applications are available, fast, reliable and scalable. It uniquely guarantees application uptime and reliability through enhanced PHP monitoring and immediate problem resolution.

Zend Studio

The Proven PHP Development Environment

Zend Studio is the leading PHP Integrated Development Environment (IDE) designed for professional developers, which includes all the development components necessary for the full PHP application lifecycle.

Zend Guard

Protect your Code and IP

Zend Guard provides independent software vendors and IT managers with the ability to safely distribute and manage the distribution of their PHP applications while protecting their source code.

Zend Optimizer

Optimize your PHP code

Zend Optimizer is a free application that runs files encoded using Zend Guard and enhances the overall performance of your PHP applications.

Zend Core

Save Time with Certified Versions of PHP

Zend Core is a Zend certified and supported version of the open source PHP. It uniquely delivers a seamless out-of-the-box experience by bundling all the necessary drivers and third party libraries to work with the database of your choice.

Zend Engine

The Heart of PHP

At the center of PHP is the Zend Engine, the component that parses and executes PHP files. It is open source software and available under an Apache-style license.

Welcome to PHP II: Higher Structures

This section of the guide contains slide images and notes.

Slide 1



PHP II: Higher Structures

Copyright © 2006-2009, Zend Technologies Inc.

Slide 2



I: Course Introduction

Slide 3

Introduction

- **This class builds upon your PHP knowledge and presents PHP programming concepts at the intermediate level**
 - It focuses on the use of PHP on the web
 - it briefly reviews basic programming concepts
- **The content of this course will enable you:**
 - Create simple web-based applications, connected to a database, and HTML forms
 - Use basic programming constructs such as conditional statements, looping, and functions
 - Use PHP to manipulate data from a database
 - Recognize code that uses classes and apply some of the underlying concepts of OOP



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 3

Slide 4

Prerequisite Knowledge

- **This class presumes you already are familiar with, and can use:**
 - Quotes and Comments
 - Data Types
 - Order of Precedence
 - Constants and Variables
 - Arrays and Multidimensional Arrays
 - Conditionals and Looping Constructs
 - Functions (internal, how to create and use your own)



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 4

Slide 5



II: PHP Syntax Review

Slide 6

Syntax Review Introduction

- **This module reviews:**
 - Quotes and Comments
 - Data Types (and introduces some new data types)
 - Order of Precedence
 - Constants and Variables
 - Arrays and Multidimensional Arrays
 - Conditionals and Looping Constructs
 - Functions (calling and defining)



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 6

Slide 7

Quotes

- Quotes indicate text (or information that is treated as text) that text is referred to as a text *string*
- Single quotes:

```
1 <?php
2     echo $a; // displays the current contents of the variable $a
3     echo '$a'; // displays the TEXT "$a"
4 ?>
```

- To specify a literal single quote, you need to put a backslash in front of it:

```
1 <?php
2     echo 'Who said: "I\'ll be back"?';
3     // Displays: Who said: "I'll be back"?
4 ?>
```



Copyright © 2006-2009, Zend Technologies Inc.

| 7

Strings are surrounded by either single or double quotes. If you use double quotes, you can put a variable in a string and the results of echoing or printing that string will include the value of the variable.

- "Here is a string with a \$myNewVar variable in it";
- "Here is a string without a variable";
- 'Here is a string without a variable';
- 'Here is a string with \$myNewVar in it that may not do what you expect';

Quotes (continued)

- Remember, double quotes are more useful because they let you embed a variable in a string and the results of echoing or printing that string will include the value of the variable:

```
1 <?php
2     $a = "Kelly";
3     echo "Who is $a, do you think?";
4         // Displays: Who is Kelly, do you think?
5 ?>
```

- This is called "Variable Interpolation"



Copyright © 2006-2009, Zend Technologies Inc.

| 8

To specify a literal single quote, you will need to escape it with a backslash (\), like in many other languages. If a backslash needs to occur before a single quote or at the end of the string, you need to double it.

Slide 9

Comments

- Well documented, commented code is easier to maintain
- In PHP, single-line comments start with # or //
- Comment indicators do not have to be at the beginning of a line

Examples of single-line comments

```
1 <?php
2     # This is a comment
3     echo "Hello world" // This part of the line is a comment
4     //This is yet another comment
5 ?>
```

- Multi-line comments start with /* and end with */

```
1 <?php
2     /* This is a multiline
3         comment and can be used to comment out code */
4 ?>
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 9

- PHP offers three different ways to make comments in your code: two single-line techniques, and a multi-line technique
 - Single-line comments often after the end of a line of code, and are used to explain what the code does
 - Multi-line comments can be used to offer more explanation, for example, to describe the intent and workings of a block of code
 - Multi-line commenting can also be used to temporarily comment out a block of code during development or debugging

Multi-line comments start with /* and end with */

Special Characters

- Sometimes you need to enter special characters such as a new line or a tab
- Here are a few common special characters you may encounter
 - New line \n
 - Tab \t
 - Escape character \" (before whatever you are escaping)
- Here are a few others special characters you might also use:
 - Carriage return \r
 - Double quotes \"
 - Backslash \\



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

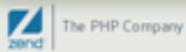
| 10

To enter special formatting characters in a double-quoted string, such as forcing a line break and carriage return, you can use escape sequences

Data Types Defined

The type of a variable is decided when the code is run by PHP, depending on the context in which that variable is used

- **Basic types of data in PHP**
 - String: text or information treated as text (numbers within quotes)
 - Integer : whole numbers
 - Float : decimal numbers
 - Boolean (or Bool) : True or False
- **Compound types:**
 - Object
 - Array
- **Special types:**
 - NULL
 - Resource



Copyright © 2006-2009, Zend Technologies Inc.

| 11

PHP is a Loosely Typed Language

- **Typed languages**
 - Strongly typed and weakly typed
 - C++, Java, are strongly typed with predefined types of data
 - PHP is weakly typed—it is only important that a variable exists, not what its type is
- **This code sample shows each variable being defined by the type of data that goes into it—these variables were not defined earlier**

```
$price=10;  
$tax=$price*.8;  
$shipping=4;  
$total_price=($price + $tax + $shipping);
```



Copyright © 2006-2009, Zend Technologies Inc.

| 12

- For example, there are some difficult to debug mistakes you can make by making a typo. For instance, you can create a new variable such as `$total_price` when you meant to use an existing variable `$Total_Price`.
- In PHP, the type of a variable is set by how the variable is used. If you create a variable named `address`, but put a float value in it, it is treated as a float rather than the string that you may have intended.
- Type checking allows you to control what kinds of data go into variables (a string can't be entered into a numeric variable, and more importantly you won't end up multiplying numbers that resulted from converting a string to a number when that was not your intent).
- A strongly typed language can help avoid hard to find mistakes. A loosely-typed language one lets you quickly create variables intended for specific purposes and narrow scope.

Slide 13

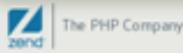
Types and Type Checking

- **Dynamic type checking**

- Static: All data types are defined before running the program
- Dynamic: Allows assignment of values at run

- **Type juggling and type casting**

- Type juggling: automatic PHP rules
- Type casting: how different types are converted into each other



Copyright © 2006-2009, Zend Technologies Inc.

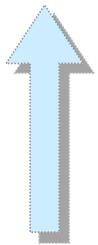
| 13

- Static or Dynamic Typing Benefits (from <http://www.procata.com/blog/archives/2006/02/09/comparing-php-with-other-languages/>) The benefit of dynamic typing is that you do not waste programmer and attention with typecasting ("static types get in my way"). The benefit of static typing is that the compiler can catch certain errors ("The compiler finds my mistakes").
- Risk tolerant: PHP, Ruby, Python, Smalltalk, Perl Risk averse: Java, C#
- Typed Languages: If a computer language is typed, it implies that operations allowed for one type of data cannot be used by another type of data. If this happens, for example, if you enter a name with as a character string instead of numbers as in a date of birth, then the error is either detected during compilation of the code -Static type checking, results in compilation error - or detected when the program is actually run - Dynamic type checking, run error. If no error checking is not implemented, then the string may be converted to a numeric value with unexpected results.
- Most languages are not so rigid as to *never* accept another data type executing an unusual function... there are workarounds, or some allowance is built-in. Languages that allow for plasticity are known as "Weak" type languages, while languages with more stringent rules are known as "Stronger" type. As you can guess, languages fall along a continuum from weak to strong.
- Static vs. Dynamic typing. Static means that all expressions have their data types defined prior to program run (java, C++). Dynamic typing allows for assignment of values during the running of the program. Non-computational languages, such as markup languages like HTML are usually not considered programming language
- PHP does not require (or support) explicit type definition in variable declaration; a variable's type is determined by the context in which that variable is used. That is to say, if you assign a string value to variable \$var, \$var becomes a string. If you then assign an integer value to \$var, it becomes an integer.
- An example of PHP's automatic type conversion is the addition operator '+'. If any of the operands is a float, then all operands are evaluated as floats, and the result will be a float. Otherwise, the operands will be interpreted as integers, and the result will also be an integer. Note that this does NOT change the types of the operands themselves; the only change is in how the operands are evaluated.
- If you wish to force a variable to be evaluated as a certain type, the name of the desired type is written in parentheses before the variable which is to be cast (this may be too advanced)
- The casts allowed are:
- (int), (integer) - cast to integer OR (bool), (boolean) - cast to boolean OR (float), (double), (real) - cast to float
- (string) - cast to string OR (array) - cast to array OR (object) - cast to object

Operator Precedence

- You can force precedence using parentheses; otherwise, they are evaluated in this order:

Highest Precedence



Lowest precedence

*	/	%
+	-	.
<	<=	>
>=		
&&		
and		
xor		
or		



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 14

Variables Defined

- A *variable* is a placeholder to contain information.
- Naming Convention: Variables start with a \$, are case sensitive, and contain only alphanumeric characters and underscore(s).
-  **TIP** Traditionally, programmers have started their variables with a lowercase string indicating the type followed by a descriptive name for the variable (e.g., \$intCounter)
-  **TIP** Always use meaningful variable names so you don't have to remember what \$a and \$b is
-  **Tip:** Define variables before using them

Constants Defined

- A *constant* is an identifier for a simple value that does not change during the execution of the PHP script
- Naming Convention: As a best practice, constants start with a letter or underscore, are case sensitive, and contain only alphanumeric characters and underscores
- Constants may be defined and accessed anywhere within a program, however,
 - you must define them before you use them
 - they may not be redefined or undefined once they have been set
-  **TIP** Programmers typically write constants with all uppercase letters. This is not a requirement, but a best practice... start writing your constants this way to more easily identify them and make the program read better overall

Slide 17

Arrays

- **An array is a group of elements given one name**
 - An array named Days could contain the days of the week
 - Arrays start counting with element zero (not one)

Monday	Tuesday	Wednesday	Thursday	Friday
Array named Days				

● SPOT QUIZ: If this is a numeric array, which element is Days[1] in this array?



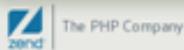
Copyright © 2006-2009, Zend Technologies Inc.

| 17

- An array is another data type--it is a collection of data that is given one name. An array is made up of a group of elements, which may be either numbers or character strings. Each element has a key and a value indicated by key => value.
- An array is arranged so that each item in the array can be located by using the key when needed.

Arrays Defined

- An array is a structure that maps one value to another
 - For example, we can map the integer value 1 to the string "Monday"
 - The value 1 in this case is the key
 - The string Monday is the key's value
- An array is made up of these key/value pairs
- An array is arranged so each value can be retrieved using the correct key
- Array types are:
 - Associative: maps a named key to a value
(💡 TIP: look for => which denotes the key value)
 - Enumerative (also called Numeric or indexed): maps an index to a value
(integer keys are manually or automatically assigned. Keys start with the value 0, a common source of error)
(💡 TIP: the index value for the first element in an array is zero)
 - Multidimensional (contains values that are arrays)



Copyright © 2006-2009, Zend Technologies Inc.

| 18

Arrays allow you to create a single variable that can contain multiple values
PHP array syntax is especially flexible... it offers two different ways to create and access arrays

Array Types

- Which of these arrays is numeric?

```
$myarray =  
array("Fruits","Vegetables","Dairy","Meat");  
  
$employee=array("last_name"=>"Karwin","first_name"=>  
"Kevin","birth_year"=> "1979")
```



Copyright © 2006-2009, Zend Technologies Inc.

| 19

- Other Examples:

```
$myarray = array("Hearts", "Clubs", "Spades", "Diamonds");  
$cards = array('red' => array(1,2,3,4), 'blue' => array(1,2,3,4), 'green' => array(1,2,3,4),  
    'yellow' => array(1,2,3,4));  
$employee=array("last_name"=>"McGregor","first_name"=> "Krishna","birth_year"=>  
    "1999")  
$employees => array("1234","Pedro","Garcia"),  
  
$films = array(  
    "genres" => array("comedy", "tragedy", "action", "romance"),  
    "film_titles" => array("Big", "Star Wars", "Titanic", "French Kiss"),  
    "stars" => array("Bill Murray", "Mark Hammell ", "Leonard DiCaprio", "Cate Blanchet"),  
);
```

Slide 20

Multidimensional Arrays Review

Here are a few ways to visualize a multidimensional array of a deck of cards as an example

Array consists of [card suit] [card face value]

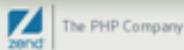
Diamonds	Hearts	Spades	Clubs
A	A	A	A
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7			



```
array (4)
["d"]=>
[1]=
[2]=
[3]= etc.

["h"]=>
[1]=
[2]=
[3]= etc.

["s"]=>
[1]= etc.
```



Copyright © 2006-2009, Zend Technologies Inc.

| 20

- In a multidimensional array, each element in the main array can also be an array.
- And each element in the sub-array can be an array, and so on.
- You can use var_dump in Studio to demo an array (to see output that looks similar to the blue box)

Slide 21

Array Exercise

- Complete the following starting code

```
// Starting code
$var = array(
    '2' => 1,
    3
);

// Insert code here
var_dump($var);
```

So that var_dump() displays the following data

```
// Output array(4)
{
    [2]=> int(1)
    [3]=> int(3)
    ["b"]=> int(4)
    [4]=> int(2)
}
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 21

Array Functions Review

- You can find the array functions at: <http://us.php.net/array>
- Here are some functions you may find particularly useful:
 - `is_array($val)` ...
 - Is `$val` an array?
 - `in_array($needle, $haystack, $strict)` ...
 - Is value `$needle` in array `$haystack`?
 - If set to `true`, the third parameter `$strict` ensures that the value stored in `$haystack` is the same type as the search value `$needle`
- There are also ArrayIterator functions at
<http://us2.php.net/manual/en/ref.spl.php>



- From John Coggeshall
- `is_array($val)` -- This is useful for testing if a variable is actually an array or not. It will return the value of "true" if the passed value is an array, "false" if it is not
- `in_array($needle, $haystack, $strict)` -- a useful function to check for the existence of a value `$needle` in array `$haystack`. The third parameter `$strict`, when set to `true`, will also ensure that the value stored in `$haystack` is the same type as the search value `$needle`. The default value for the `$strict` parameter is `false`.
- `count($val)` -- This returns the number of elements contained within an array. Note that in the event that `$val` is not a valid PHP variable, it will return a value of "0" instead of causing an error. It is recommended that `is_array()` be called prior to `count()` to ensure that the return value reflects the true nature of the variable being checked.

More Array Functions

- **Here are more functions you may find particularly useful:**
 - **count(\$val)**
 - Returns the number of elements contained within an array
 - If \$val is not a valid PHP variable, it returns a value of "0"
 - Call `is_array()` prior to `count()`
 - **array_rand(\$arrayName, \$num_required)**
 - Picks one or more random array keys out of a single dimensional array
 - **array_key_exists(\$key, \$array2search)**
 - Checks if the given key or index exists in the array and returns `true` or `false`

Yet More Array Functions

- **array_keys()** Return all the keys (numeric and string) of an array
 - If the optional `search_value` is specified, then only the keys for that value are returned
- **array_merge()** Merges one or more arrays
 - If the input arrays have the same string keys, then the later value for that key will overwrite the previous one
 - If, however, the arrays contain numeric keys, the later value will not overwrite the original value, but will be appended
- **Spot Quiz: What is the result of the following code?**

```
<?php
$array1 = array("color" => "red", 2, 4);
$array2 = array("a","b","color"=>"green","shape"=> "trapezoid",4);
$result = array_merge($array1, $array2);
print_r($result);
?>
```



Operators

- In programming and logic, an **operator symbol** is used to perform an operation on some value such as addition, subtraction, and so on
- Operators assign values, work with strings and numbers and can control program flow
- Types of operators
 - Assignment
 - Math
 - String
 - Comparison
 - Logical



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 25

Assignment Operators

- The Assignment operator (`=`) sets the value to the left of the operand to the value on the right
- You can also combine it with other operators

- Combined Operators

<code>\$a += \$b</code>	<code>\$a = \$a + \$b</code>	<u>Same as:</u>
<code>\$a -= \$b</code>	<code>\$a = \$a - \$b</code>	Addition
<code>\$a *= \$b</code>	<code>\$a = \$a * \$b</code>	Subtraction
<code>\$a /= \$b</code>	<code>\$a = \$a / \$b</code>	Multiplication
<code>\$a %= \$b</code>	<code>\$a = \$a % \$b</code>	Division
<code>\$a .= \$b</code>	<code>\$a = \$a . \$b</code>	Modulus
		Concatenate

- Concatenate operator `(.)` is often used to create output

 **TIP** Do not think of `=` as “equal”... think of it as “assignment”
 **TIP** Think of `==` as “equal”



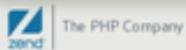
Math Operators

Example	Name	Result
<code>-\$a</code>	Negation	Negative Value of \$a
<code>\$a + \$b</code>	Addition	Sum of \$a and \$b
<code>\$a - \$b</code>	Subtraction	Difference of \$a and \$b
<code>\$a * \$b</code>	Multiplication	Product of \$a and \$b
<code>\$a / \$b</code>	Division	Quotient of \$a and \$b
<code>\$a % \$b</code>	Modulus	Remainder of \$a divided by \$b

 **TIP** The division operator ("/") returns a float value any, even if the two operands are integers (or strings that get converted to integers)

There are also increment/decrement operators `++` and `--`

`$a++` is equal to `$a+1`



String Operators: . and .=

- The **concatenation operator (.)** allows you to combine two strings together into a single string

- For example, both these lines of code have the same result:

```
echo "Hello " . "World";
echo "Hello World";
```

Spot Quiz: What is the result of the following code?



```
<?php
$a = "test";
$b = "This is a ".$a;
echo $b;
?>
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 28

Logical Operators

- You may already be familiar with some of these operators
 - **and** ("and" and `&&` symbols are both supported)
 - **or** ("or" and `||` symbols are both supported)
 - **xor** ("xor" and `^` symbols are both supported)
 - **not** ("not" and `!` symbols are both supported)
- Why have two symbols that do the same thing?
 - Order of precedence: depending on which symbol you use - for example, **and** or **&&** - the entire statement may evaluate to a different result



Copyright © 2006-2009, Zend Technologies Inc.

| 29

Slide 30

Conditionals

- **Here are the most common conditional statements:**
 - `if`
 - `if .. else`
 - `elseif`
 - `switch`



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 30

- **Conditional statements allow your program to make choices.**
- **A computer can typically make two choices, true or false. You can “chain” them together to allow more choices.**

Shorthand Conditionals

- Here is how to assign a variable based on a condition
 - `if($a > 10) $a = 10;`
 - Useful to make sure `$a` doesn't become greater than 10
- There is a shorthand operation called the *ternary operator* which makes this condition faster to type:
 - `(<condition>) ? <value of true> : <value if false>;`
- The above `if` conditional can also be re-written using the ternary operator as..
 - `$a = ($a > 10) ? 10 : $a;`



Copyright © 2006-2009, Zend Technologies Inc.

| 31

Conditional Exercises

- Write a conditional statement, using If and Switch, that executes if either \$a or \$b = true

```
// Starting code  
  
$a = true;  
$b = false;
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 32

Types of Loops

- **Types of loops:**
 - `for`
 - `while`
 - `do-while`
- **Important features of loops**
 - `continue`
 - `break` (you've already seen this used with `switch`)

Continue

- **continue is used within looping structures (do, while, for, foreach) to skip the rest of the current loop iteration and go to the beginning of the next evaluation**

Spot Quiz: What is the result of the following code?

```
<?php
for ($i=0; $i<=5; $i++) {
    if ($i==3){continue;}
    echo "The number is ".$i;
    echo "<br />";
}

?>
```

The result is



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 34

Break

- **break ends running of the current `for`, `foreach`, `while`, `do-while` or `switch` structure**

```
<?php
do {
    // some code
    if ($condition)
        break;
    // some more code
} while (true);
?>
```



Copyright © 2006-2009, Zend Technologies Inc.

| 35

- **break accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.**
- **Use this feature with caution.**
 - If you edit the code and change the number of loops, using a numeric argument could cause the break to occur in a different place after you've made the change.

Looping Practice

- Iterate over the array so the output is

1: 1, 2, 3, 4
3: 1, 2, 3, 4

```
// Starting code

$array = array(
    1 => range(1, 4),
    2 => range(1, 4),
    3 => range(1, 4)
);
```



Copyright © 2006-2009, Zend Technologies Inc.

| 36

Looping in Arrays

- There is one other looping structure we haven't reviewed yet...
- `foreach` works only on arrays, and will issue an error when you try to use it on a variable with a different data type, or with an uninitialized variable
- It has two syntax options depending on what kind of array you are working with:

```
foreach (array_expression as $value)
    // statement
```

```
foreach (array_expression as $key => $value)
    // statement
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 37

Examples of foreach

- **Associative array**

```
1 <?php
2     // foreach associative array example
3     $arr[0] = 1.2;
4     $arr[1] = 2.4;
5     $arr[2] = 3.6;
6     $arr[3] = 4.8;
7     foreach ($arr as $key => $value) { // key/value pair example
8         $arr[$key] = $value * 1.5;
9         echo "$arr[$key]<br> \n";
10    }
```

- **Enumerative array**

```
1 <?php
2     //foreach enumerative array example
3     $arr = Array(1,2,3,4);
4     foreach ($arr as $value) {
5         $value = $value * 1.5;
6         echo "$value <br / \n";
7     }
```



Exercise: foreach

- **Fill in the code to take the array and make it uppercase:**

```
// Starting Code

$array = explode(' ', 'Zend Training - Building Security
into your PHP Applications');

// Enter code here

var_dump($array);
```



Copyright © 2006-2009, Zend Technologies Inc.

| 39

Functions Review

- **Functions encapsulate code into “black boxes”**
 - Provide inputs, receive outputs (no concern about how)
- **There are several types of functions:**
 - internal or built-in functions provided with the language
 - user-defined functions (ones you create)
 - functions created by others
- **Function names should be descriptive and start with a letter or underscore, followed by any number of letters, numbers, or underscores**
- **Do you know approximately how many internal functions there are?**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 40

Calling Functions

- What common elements do these commands share?

```
▪ unset($myarray);  
▪ list();  
▪ totalPrice($price, $quantity, $tax);  
▪ $command = getConsoleInput($end_game);  
▪ $mydbconnection =  
    mysql_connect("localhost", "deirdre", "password");
```

Answer:

All functions



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 41

Declaring Functions

- It is good practice to declare functions before you call them
- Function names should be descriptive and start with a letter or underscore, followed by any number of letters, numbers, or underscores
- Functions by convention mix upper and lower case letters, which is called CamelCase because it looks like the humps of a camel
- Most code can be contained in functions, including calls to other functions or even itself

```
function myFunction(parameters) {  
    //block of code  
    yourFunction()  
}
```



Copyright © 2006-2009, Zend Technologies Inc.

| 42

- The area between the { and the } in functions is where you make the function do something using loops, conditionals, or simple print statements
- They often start with a verb: get, put, is...
- Important to give functions meaningful names
- Function code is between the { and }

Function Input: Parameters

- To run properly, Functions require information (referred to as input, a parameter, or an argument)
- This example connects to a database and needs three inputs: the name of the system, the username, and the password

```
$mydbconnection = mysql_connect("localhost", "username", "password");
```

- This example calculates the value of a hand of cards and requires one input

```
calculateHandValue($cards);
```



Copyright © 2006-2009, Zend Technologies Inc.

| 43

Optional Parameters

- Functions are typically designed to have any optional parameters at the end as in this example:

```
string trim ( string str [, string charlist] )
```
- This function returns a string with whitespace stripped from the beginning and end of str
- Without the second optional parameter `charlist`, `trim()` will strip characters including:
 - " " (ASCII 32 (0x20)), an ordinary space
 - "\t" (ASCII 9 (0x09)), a tab
 - "\n" (ASCII 10 (0x0A)), a new line (line feed)
- Consult PHP reference sites/materials to determine which parameters are optional

Function Output: Return Value

- Use `return` to return a value from a function at any time
- You can return either any value such as a string
- By default the function returns `NULL`, although it is best practice to always return from the function explicitly

```
function myFunction() {  
    return "hello there";  
}
```

```
function myFunction2() {  
    $char=1;  
    return $char;  
}
```



Copyright © 2006-2009, Zend Technologies Inc.

| 45

- You can do calculations and assign them to a variable and then return that value.
- For example within a function you could have the following lines:
`$total_price = $price + $tax;
return $total_price;`

Exercise: Writing Functions

- Write a function called `getName()` that has 3 parameters:

first name

last name

optional middle initial

and returns a string in the format of “Smith, John K.”

```
// Starting Code  
  
echo getName('john', 'smith', 'k');
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 46

Slide 47



III: Building On PHP Language Concepts

Slide 48

Building on Concepts Introduction

- **In this class, you will review some concepts and learn new ones including the following:**
 - Including and requiring files
 - Globals
 - References
 - List()
 - Working with files



The PHP Company

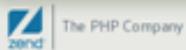
Copyright © 2006-2009, Zend Technologies Inc.

| 48

Including and Requiring Files Review

- Within PHP scripts, you can call other scripts or libraries by using one of the following commands:
 - `include` or `include_once`
 - `require` or `require_once`
- The `include` commands return a warning if they cannot find the file
- The `required` commands return a fatal error if they cannot find the file
- The `_once` variations limit the file to being loaded once

```
<?php  
include 'library.php';  
?>
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 49

Including and Requiring Files

- In your PHP.INI file, you need to specify which directory paths to look for files in addition to using one of the following statements in your code
 - `include` or `include_once`
 - `require` or `require_once`
- Additionally, there are some `php.ini` parameters that relate to including or requiring files
- `allow_url_include` allows the use of URL-aware fopen wrappers with the following functions: `include()`, `include_once()`, `require()`, `require_once()`
 - This setting requires `allow_url_fopen` to be on

Exercise: Including and Requiring Files

- Case 1: You have several functions that are used throughout the course of the program
- Case 2: You have several JavaScript files that are referenced throughout the program and you don't want to have the files downloaded more than once
- Case 3: There are multiple places where an HTML file needs to be included
- SPOT QUIZ: For each case above, which option should you choose?
 - include or include_once
 - require or require_once

Globals

- Here is an example of variables declared outside of a function that use the `global` keyword within the function

```
$a=10;  
$b=20;  
  
function myFunction() {  
    global $a;  
    global $b;  
  
    $c=$a+$b;
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 52

\$GLOBALS Array

- **\$GLOBALS is another way to use a variable - defined outside of a function - within that function**
- **The \$GLOBALS array is an associative array**
 - The name of the global variable is the key
 - The contents of that variable are the array element values
- **💡 TIP : use any global variable sparingly (similarly for explicitly defining a global variable using the global keyword)**

```
$myVar =2;           // declared outside a function

function myfunction() {
    $GLOBALS['myVar']++; // calling it within a function
}

myfunction();
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 53

Slide 54

register_globals

- This is an option that is set to ‘off’ by default
- It is widely recommended that you leave this set to off, to avoid making your code more vulnerable to security breaches
- To learn more, see http://us.php.net/register_globals



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 54

Static Variables

- A **static variable exists only in a local function scope, but it preserves the value when the function runs and leaves this scope**
- If you want a piece of data to persist between calls to the same function, use static variables (with caution, though)
- Use the **static keyword to define a static variable**

```
<?php
function MyTest()
{
    static $bVar = 1;
    echo $bVar;
    $bVar++;
}
?>
```



Copyright © 2006-2009, Zend Technologies Inc.

| 55

References

- **PHP References allow you to utilize the same variables using different names**
- **Think of them as “variable aliases”**
- **<http://www.php.net/manual/en/language.references.php>**
- **& denotes a reference - the remaining syntax depends on what you are doing:**

```
$ref0 = &$origVar;      // Two variable names for the same data
$ref1 = &$myarray;       // References the entire Array
$ref2 = &$myarray[0];    // References to the first index or
                      // item in the array
```

How References Work: PHP View

Simple assignment

```
$a = 'a';      'a'  
$b = 'b';      'b'
```

Copy on Write

```
$a = 'a';      'a'  
$b = $a;  
then...  
$b = 'b';  
$a is      'a'  
$b is      'b'
```

Reference

```
$a = 'a';      'a'  
$b = &$a;  
then...  
$b = 'b';  
$a is      'a'  
$b is      'b'
```

Passing by Reference

- **Variables can be passed as Parameters by Reference by specifying the & symbol before a parameter of a function**
 - `function myfunction(&$myvariable)`
- **When a parameter is passed by reference, any changes made to the variable by the function will change the variable you passed as a parameter**

```
$a = 10;

function myFunction(&$myvar) {
    $myvar = 20;
}

myFunction($a);

print $a; // prints 20
```

Exercise: References

- Write a function called `makeArrayUpper()` that takes the array and makes it uppercase and returns true on success, or false if an array has not been passed

```
// Starting Code

$array = explode(' ', 'Zend Training - Building
Security into your PHP Applications');

makeArrayUpper($array);

var_dump($array);
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 59

list()

- Like `array()`, this is not a function, but a language construct
- `list()` assigns a list of variables to an indexed array in one operation
- Note: `list()` only works on numerical arrays and assumes the numerical indices start at 0

```
list($usec, $sec) = explode(" ",micro());
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 60

Working with Files

- You can work with files or URLs using the file handling commands in PHP - these include commands to open files, read, write, append, delete and so on
- You will need to set some `PHP.INI` settings related to file-handling
- **💡 TIP :** When working with file handling commands, you must verify that you have appropriate permissions to read, write, or modify files on the server you are accessing. Otherwise, you'll spend a lot of time trying to debug code
- **💡 TIP :** Files must be closed before you can open them again

Pertinent PHP.INI Settings

- **allow_url_fopen (default On)**
 - Determines if fopen() calls can be done using URLs
 - `fopen('http://www zend.com/')`;
- **allow_url_include (default Off)**
 - Determines if require* and include* can require URL-based includes
 - `include('http://www zend.com/header.php');`
- **include_path**
 - Sets the path that require*, include* and, optionally, file functions can search for filenames
 - `include_path: ./:/usr/local/Zend/ZendFramework/library`
- **auto_append|prepend_file**
 - Automatically prepends or appends the specified file for inclusion in the HTTP request

File System Basics - Review

We are going to look at:

- **Constants**
 - Directory
 - Path
- **Commonly-used file functions**
 - Opening
 - Reading
 - Writing
 - Locking
- **File system performance**



Copyright © 2006-2009, Zend Technologies Inc.

| 63

File System Basics: Directory Constants

- Pertinent constants

- DIRECTORY_SEPARATOR constant

- Use instead of "/" when working on non-UNIX systems

Instead of \$file = '.../data/file.txt'

```
$file = sprintf('...%sdata%sfile.txt',
    DIRECTORY_SEPARATOR,
    DIRECTORY_SEPARATOR);
```

- Safer for cross-platform use

Slide 65

File System Basics: Path Constants

- **Pertinent constants**

- PATH_SEPARATOR Constant

Use instead of `set_include_path(get_include_path() .':../data');`

```
set_include_path(sprintf('%s%s%s%s',
    get_include_path(),
    PATH_SEPARATOR,
    '..',
    DIRECTORY_SEPARATOR,
    'data'
));
)
```

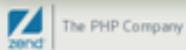
*** also `fopen()` example

- Safer for cross-platform use



Types of File Functions

- **Resource-based**
 - Generally denoted by “f” prefix
 - Examples include
 - `fopen`
 - `fread`
 - `fclose`
- **Filename-based**
 - Generally denoted by “file” prefix
 - Examples include
 - `file_get_contents`
 - `file_put_contents`
 - `file_exists`
 - `fileatime`



Opening Files

- Open files using `fopen()`

```
$resource = fopen($filename, $mode, $use_include_path,  
                  $resource, $context);
```

- Returns a resource on success, false on failure
- `$filename`: a string value that specifies the name of the filename to open
- `$mode`: a string value denoting how you want to interact with the file
- Mode options are shown on later slides

Slide 68

Opening Files (continued)

- **\$filename**
 - The name of the filename to open
- **\$mode**
 - A string value denoting how you want to interact with the file
 - (The mode options are shown on the next slide)



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 68

Slide 69

Opening Files (continued)

- **fopen() – optional parameters**
 - **bool \$use_include_path**
 - If a relative path is passed in to the \$filename parameter **fopen()** will search the current include_path for the file
 - **resource \$context**
 - Allows you to set various stream parameters
 - Parameters available differ based on the stream type
 - HTTP stream parameter examples
 - method – Denotes the request type, GET or POST
 - content – Content following the headers
 - Typically used with POST requests
 - proxy – The URL of a proxy server
 - HTTP-base authentication **



Copyright © 2006-2009, Zend Technologies Inc.

| 69

fopen() Modes

String Access Mode

- r** Open the file for read only, start access from beginning of the file
- r+** Open the file for read and write, start access from beginning of the file
- w** Open the file for write only
 - If the file exists, erase all contents
 - If file does not exist, attempt to create the file
- w+** Open the file for read and write
 - If the file exists erase all contents
 - If the file does not exist, attempt to create the file



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 70

fopen() Modes

String Access Mode

a Open the file for write only

If the file does not exist, attempt to create it

If the file exists, access starts from end of file (no erasing)

a+ Open the file for read and write

If the file does not exist, attempt to create it

If the file exists, access starts from the end of file (no erasing)

x Create and open the file for writing only. Returns false if the file exists already

x+ Create and open the file for reading and writing. Returns false if the file exists already

Note: Windows offers a text-mode translation flag ('t') which transparently translates \n to \r\n when working with the file. You can also use 'b' to force binary mode, which will not translate your data.

To use these flags, specify either 'b' or 't' as the last character of the *mode* parameter



Copyright © 2006-2009, Zend Technologies Inc.

| 71

Example: Using fopen()

- This example shows fopen used with error checking

```
1 <?php
2     $fh = fopen(__FILE__, 'r') or die('Could not open myself');
3     while (!feof($fh)) {
4         echo fread($fh, 1024);
5     }
6     fclose($fh);
7 ?>
```



Copyright © 2006-2009, Zend Technologies Inc.

| 72

- Note: there is a Test Your Knowledge on fopen later and an exercise using it

Reading From Files

- **💡 TIP : These functions also work for URLs**
 - `string fgets (resource $handle [, int $length])`
retrieves a line of a file from a file pointer (`$handle`)
 - If `$length` is not provided, `fgets()` reads 1024 bytes from the file until a new line character is read or until the function reaches the end of the file
 - Return value from `fgets()` is the string read from the file
 - `string fread (resource $handle, int $length)` reads up to `$length` bytes from the file pointer referenced by `$handle`

More File Reading Functions

- **fgetc** Gets a character from the file pointer
- **fgetcsv** Parses line it reads for fields in CSV format and returns an array containing the fields
- **fgetss** Gets a line from file pointer and strips HTML tags
- **fscanf** Parses (processes) input from a file according to a specified format
- **rewind** Rewinds the position of a file pointer
- **file_get_contents** Reads entire file into a string

Writing to Files

- `int fwrite(resource $handle, string $string[, int $length])`

Writes the contents of string to file stream pointed to by handle

- `fputs()`

Is the alias of `fwrite()`

- `fputcsv`

Formats a line as CSV and writes to file pointer

- `file_put_contents`

Writes a string to a file



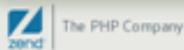
The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 75

Locking Files

- You can lock a file to prevent the possibility that data could be corrupted by 2 different processes/requests writing to the same file at the same time
- Done using `flock()` on a file resource
- 2 types of locks
 - Shared lock (`LOCK_SH`)
 - Any request can read from the locked file but only the resource variable that was locked can write to the file.
 - Exclusive lock (`LOCK_EX`)
 - Any read or write request will block until the resource is unlocked
- Locks are held for PHP... Applications outside of PHP may ignore the lock



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

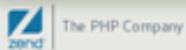
| 76

Closing Files and Other File Functions

`bool fclose (resource $handle)`

- closes file pointed to by `$handle` - bool indicates success or not

- `file` **Reads an entire file and returns contents in an array**
`$filename` **can be a URL**
- `glob` **Find pathnames matching a pattern**
- `is_dir` **Tells whether the filename is a directory**
- `is_file` **Tells whether the filename is a regular file**
- `unlink` **Deletes a file**
- `tmpfile` **Creates a temporary file**
- `fpassthru` **Reads from the current position to EOF**



Copyright © 2006-2009, Zend Technologies Inc.

| 77

Slide 78

Exercise: Working with Files

- **Read all of the directories in the application/views/scripts directory and for each directory print the following:**
 - File name
 - File size
 - Number of lines in the file



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 78

File System Performance

- **Simple/fast reading**

```
$filecontents = file_get_contents('data.txt');
```

- **Simple/fast writing**

```
file_put_contents('data.txt', $filecontents);
```

- **The long/slower way**

```
$fh = fopen('data', 'r');
$filecontents = fread($fh, filesize('data.txt'));
fclose($fh);
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 79

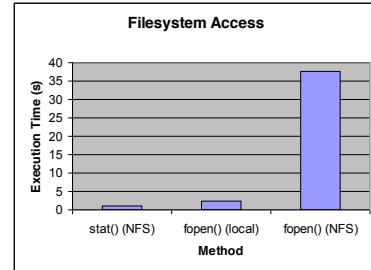
File System Performance

- **.htaccess on Apache (web server performance)**
 - .htaccess files are used to provide per-directory configuration settings outside of httpd.conf
 - Often they are used for setting a bootstrap file for an MVC implementation or setting directory-based permissions
- **Problem is that Apache will stat each directory from the request to the root directory, parsing and implementing .htaccess files**
- **Solution is to put .htaccess-type configuration options in httpd.conf using <Directory> or <Location> tags**

Slide 81

File System Performance

- **NFS**
 - Often in clustered environments, NFS is used to house source code in a single place so code rollouts are less troublesome
- **Problems**
 - Single point of failure
 - Because NFS is done over a network, 2 additional problems
 - Network latency
 - Inability of OS to cache disk pages



Home Work: File Handling

- Using the f* functions, display the contents of the admin home page (<http://<IP ADDR>/admin/?useHttpAuth=1>), stripping the content of any HTML tags using `strip_tags()` and display it on screen
 - Use php.net documentation to determine context settings

```
// Starting code
$header = 'Authorization: Basic
'.base64_encode('administrator:password')."\r\n";
$header .= 'Cookie: '.session_name().'='.md5(micro())."\r\n";
```



Slide 83

Module Summary

- Q&A



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 83

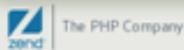
Slide 84



IV: Configuring PHP

Configuring PHP Introduction

- **In this module, you will learn the following:**
 - Setting up a PHP environment
 - More about `php.ini`



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 85

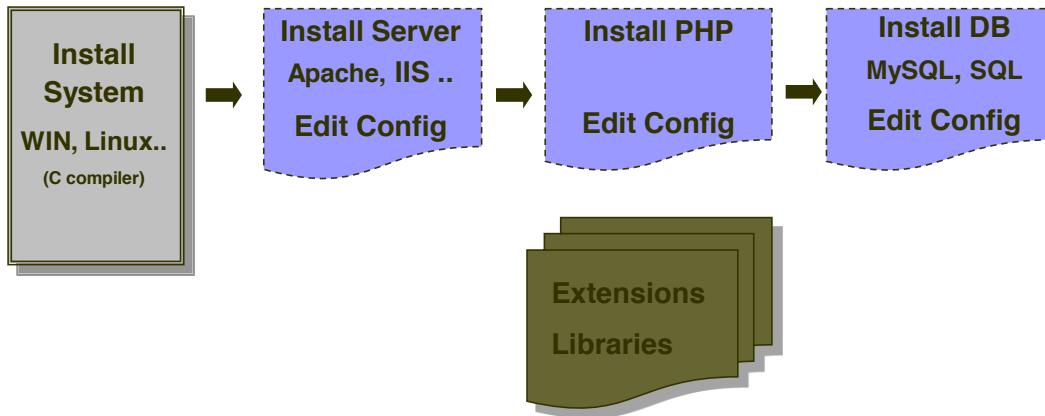
- http://articles.techrepublic.com.com/5100-22_11-5268948.html

Setting Up a Development Environment

- **For web sites and web applications (server-side scripting), you need three things:**
 - PHP itself
 - A web server (typically Apache; IIS)
 - A web browser (IE, Firefox, etc.)
- **And the little essential things...**
 - Text editor
 - Browser / plug-ins
 - IDE (recommended)
 - Source Control (recommended - Subversion or CVS)
- **For command line scripting, you only need the command line executable (no web server or browser needed)**

Slide 87

Flow Diagram for Setting Up Dev. Envt.



To learn more:

<http://us2.php.net/ini.core>

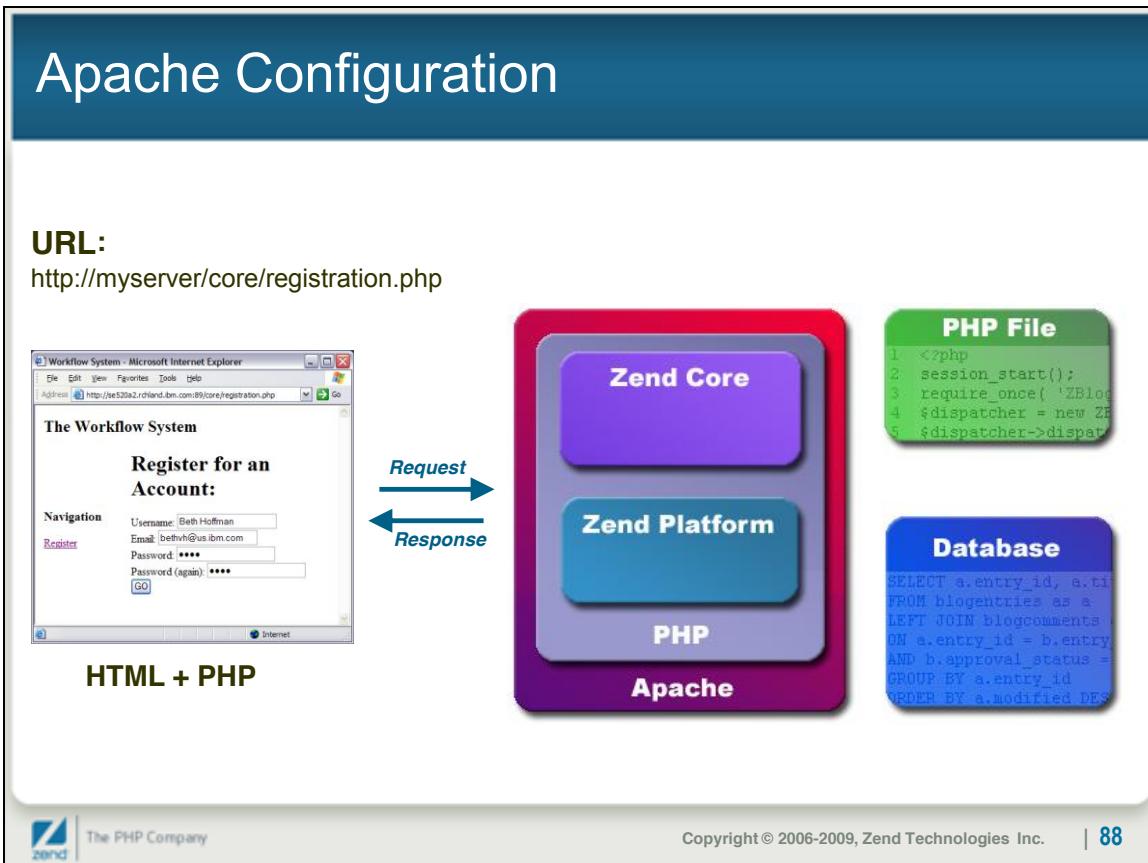
http://www.onlamp.com/pub/a/php/2001/01/11/php_admin.html



Copyright © 2006-2009, Zend Technologies Inc.

| 87

Slide 88



Slide 89

IIS Configuration

URL:
<http://myserver/core/registration.php>

The Workflow System
Register for an Account:
Navigation
Username: Beth Hoffman
Email: bethv@us.ibm.com
Password: ****
Password (again): ****
GO

IIS

PHP File

```
1 <?php
2 session_start();
3 require_once( 'Zend/Loader.php' );
4 $dispatcher = new Zend_Controller_Dispatcher_Fast_CGI();
5 $dispatcher->dispatch();
```

Database

```
SELECT a.entry_id, a.title
FROM blogentries as a
LEFT JOIN blogcomments
ON a.entry_id = b.entry_id
AND b.approval_status = 1
GROUP BY a.entry_id
ORDER BY a.modified DESC;
```

HTML + PHP

The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 89

Slide 90

Demonstration: Zend Core

- **Tour of Zend Core**
 - PHP Config
 - Extensions



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 90

Configure a Development Environment

Now that we have looked at setting up the environment, we need to...

- **Configure the various components:**
 - System configurations
 - Server configurations
 - DB configurations
 - PHP configurations (`php.ini`)
- **Build a configuration manager that understands the difference between the development and production environments**
 - An environment variable that is set, or a configuration option in a config file or config array

Configuring PHP Review

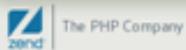
Review: `php.ini` contains settings to control how the PHP processor functions

- Provides a central place for all language-based run-time configurations
- Use the `php.ini` file to specify paths needed when using PHP, such as where libraries are located
- System looks for `php.ini` file in the path defined when PHP is compiled (the default `php.ini` file)
 - *NIX, typically
 - Zend Core `/etc/php.ini or /usr/local/lib/php.ini`
 - Windows,
 - Zend Core `C:\program files\PHP\php.ini`
 - Zend Core `C:\program files\Zend\Core\etc\php.ini`
- Settings can often be overridden by directives within an Apache `.htaccess` file or within a script



PHP.INI Structure Review

- **Called an ini file because follows the convention of WIN ini (ASCII) files**
- **Follows rules of PHP syntax and naming conventions**
- **; ... denotes a comment (💡TIP: use to enable/disable functions)**
- **If using command-line version of PHP, `php.ini` will be read every time the PHP binary program is run**



Copyright © 2006-2009, Zend Technologies Inc.

| 93

Slide 94

PHP.INI Set Paths

- **Review: Set a search path with `include_path`; becomes default if specific path name is not specified for a file**
- **Use path to add libraries (... :/usr/local/lib/php/pear:)**
 - Use colons for UNIX; semicolons for WIN



The PHP Company

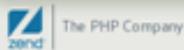
Copyright © 2006-2009, Zend Technologies Inc.

| 94

Slide 95

PHP.INI Select Settings

- **engine = On**
 - Means embedded PHP code is parsed by web server - this should almost always be on
 - Can be used on a per-directory or per-virtual server basis
 - This allows certain directories to not process PHP code
- **short_open_tag = On**
 - Allows use of <? and ?> tags, <?php and ?> tags
 - May need to disable if using languages other than PHP, such as XML
- **output_buffering = Off**
 - Sets whether or not data is sent directly to the web server
 - Turning output buffering on will cause the output to be sent in fewer calls to the web server
 - Turning output buffering on also allows you to add HTTP headers to the response after data has been sent



Copyright © 2006-2009, Zend Technologies Inc.

| 95

PHP.INI - Error Reporting

E_ALL	All errors and warnings
E_ERROR	Fatal run-time errors
E_WARNING	Run-warnings (non-fatal errors)
E_PARSE	Compile-parse errors
E_NOTICE	Run-notices (often result of bug in code; possibly intentional (e.g., rely on fact uninitialized variable automatically initialized to empty string))
E_STRICT	Run-notices (if enabled, PHP suggests changes to code which will ensure best interoperability and forward compatibility)
E_CORE_ERROR	Fatal errors that occur during PHP initial startup
E_CORE_WARNING	Warnings (non-fatal errors) occurring during PHP initial startup
E_COMPILE_ERROR	Fatal compile-errors
E_COMPILE_WARNING	Compile-warnings (non-fatal errors)
E_USER_ERROR	User-generated error message
E_USER_WARNING	User-generated warning message
E_USER_NOTICE	User-generated notice message

PHP.INI Select Settings

- **display_errors = On**
 - Should be turned on in a development environment
 - Should be turned off in production
- **display_startup_errors = On**
 - Use only in development environments; disable for production, using logs instead... Security issue
- **max_execution_time = # (_input_)**
 - Allows you to forcibly terminate a script, based on exceeding some set time limit
 - Useful in protecting against infinite or excessive looping



PHP.INI Select Settings: Security

- **safe_mode = On**
 - Safe mode controls user access searching for directories and files
 - With `safe_mode_exec_dir` set, user restricted to own document root
 - Should always be used when PHP is used as a CGI
 - Helps to increase system security, but does not ensure system is secure
- **register_globals = Off**
 - Generally, this should be set to off for Form security
 - If dealing with older (3.x) scripts, need to enable
- **upload_tmp_dir**
 - This setting relates more to data security than network...
 - Indicates location for temporarily storing form data - otherwise, default is for data to be housed on the server, with global access

Slide 99

PHP.INI Select Settings: Performance

- **register_long_arrays = Off**
 - \$HTTP_GET_VARS and \$HTTP_POST_VARS
 - Disabling these arrays can improve performance; not likely to be used (requires PHP 5)
 - \$_GET and \$_POST should be used instead
- **ini_set()**
 - Overrides initial php configuration set by `php.ini` on an individual script basis
 - Especially useful if you have no access to the central `php.ini` file (e.g., with shared servers)
 - Some settings **cannot** be changed using `ini_set`. Examples include
 - Extension_dir
 - Sendmail_path



Copyright © 2006-2009, Zend Technologies Inc.

| 99

Slide 100

Exercise: Configuring an Environment

- **Convert the Environment designated within the Z Blog application from PRODUCTION to DEVELOPMENT**
 - The application is accessible on the VMWare image
 - Your instructor will provide details
 - HINT: think configuration



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 100

Slide 101



V: Regular Expressions

Slide 102

Introduction to Regular Expressions

- **In this class we will discuss...**
 - What regular expressions are
 - Common syntax for creating regular expressions
 - Functions used with regular expressions
 - How to create a regular expression to check that a valid email address has been entered



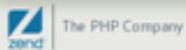
The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 102

Regular Expressions

- A **regular expression** is a string used to describe, parse or search text within a string, according to certain syntax rules
- Here are a few examples:
 - "Bill|William": matches a string that has either "Bill" or "William" in it
 - "^ [a-zA-Z]": searches for a string that starts with a letter
- You commonly use them to
 - check for a valid email address, URLs, complex patterns
 - search and replace a string
 - Explode a string on complex patterns



Copyright © 2006-2009, Zend Technologies Inc. | 103

- <http://us3.php.net/pcre> PERL compatible regular expressions

<http://www.regular-expressions.info/php.html> mentions three sets of regular expressions

Characters & Symbols

- In addition to its regular search and modification function, certain characters denote a special meaning with Regular Expressions (literals vs. metacharacters)
- Partial list of character symbols used (complete list in Appendix) :
 - / indicates a delimiter
 - ^ indicates the start of the line matches
 - \$ indicates the end of the line matches
 - \ is used as a general escape character
 - { } Indicates number of s to match
 - {n} n s; {n,} at least n s; {n,m} min/max s
 - () or [] encloses a pattern (group of characters as one unit)
 - | separates alternative patterns
 - * indicates a single character *and more*

Character Classes

- There are several pre-defined character classes you can use:

\d ... any decimal digit
\D ... any character that is not a decimal digit
\s ... any whitespace character
\S ... any character that is not a whitespace character
\w ... any "word" character
\W ... any "non-word" character
\b ... word boundary
\B ... not a word boundary

Character Classes (continued)

- \A... start of subject (independent of multiline mode)
- \Z ... end of subject or newline at end (independent of multiline mode)
- \z... end of subject (independent of multiline mode)
- \G ... first matching position in subject



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 106

RegEx Functions

There are three basic types of regular expression functions:

`ereg` functions:

PHP's standard regular expression syntax; implement POSIX extended RegEx

- + aspect part of codebase, supported by all PHP versions (3, 4, 5)
- aspect does not support some recent RegEx elements

`mb_ereg` functions:

multi-byte version of `ereg`; useful for character-based languages

`preg` functions:

Perl-compatible regular expression syntax

- + aspect wider function set than `ereg`
- aspect can be complex to use



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 107

preg RegEx Functions

- Here is a partial list of Perl-compatible functions

`preg_grep`: Return array entries that match the pattern

`preg_match_all`: Perform a global regular expression match

`preg_match`: Perform a regular expression match

`preg_quote`: Quote regular expression characters

`preg_replace`: Perform a regular expression search and replace

`preg_split`: Split string by a regular expression

`preg_replace_callback`: Almost identical to `preg_replace` except that instead of a replace string a callback function is used



Example: Validating an Email Address

- Let's look at Regular Expressions in action, using an email example
- Email addresses are typically in the format
username @ domain . ext

First, we'll look for the username characters up to the @

- Start the expression with ^ and then use [] to contain the possible values, which are upper and lower case A-Z and the numbers 0-9
$$^ [A-Za-z0-9] + @$$
- The +@ means there must be something found in the first part before the @ to be a valid email address



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 109

Validating an Email Address (continued)

Next, we'll look for the domain name characters up to the “.”

- Again use [] to contain the possible values

[A-Za-z0-9]+\\.

- The + means there must be something found in the first part before the “.” to be a valid domain

- Why use “\\.” versus “.” ?

- “.” means any character in regular expressions
 - “\\.” escapes the regular meaning so that the period can be used as a symbol

RULE: To match a metacharacter in a pattern literally, you have to escape it with a backslash



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 110

Validating an Email Address (continued)

Finally, we'll look at the extension characters

- Again use [] to contain the possible values
[A-Za-z] {2,4} \z
- The {2, 4} means there must be at least 2 characters and no more than four (com, net, org, biz, info)
- The \z ends the string so additional characters are not part of the validation



Copyright © 2006-2009, Zend Technologies Inc. | 111

- For your information: regular expression tester <http://www.quanetic.com/regex.php>

Slide 112

Exercise: Validating a URL

Write the regular expression to search for the international format of a URL address - example:

`www.bbc.co.uk`

- **Would this RegEx also return `news.bbc.co.uk` as a result?**
- **How would you modify it to allow for an extension with at least 3 characters and no more than 20 characters - example:**

`www.bbc.co.uk/worldservice/`



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 112

Validating an Email Address (continued)

Now we'll look at a code example that uses our statement:

- The subject is a variable that stores an email address - so, we are checking to see if that is a valid email address
- The expression we created is used as a function parameter - \$pattern - in the following function

```
int preg_match ( string $pattern, string $subject  
[, array &$matches [, int $flags [, int $offset]]] )
```

- This function searches \$subject for a match to the regular expression given in \$pattern



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 113

Syntax

- **The most popular use for Regular Expressions is validation**
- **Examples - but need to modify to use...**
 - Check to see if data include SQL injection attempts
 - `/ (SELECT | INSERT | UPDATE | DELETE) /i`
 - Validate an email address
 - `/^[\^@\s]+@[\\(\\[?)([-\\w]+\.\.)+([a-zA-Z]{2,6})/`



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 114

Greediness

- **Regular expressions will do their best to guarantee a match**
 - `/<.*>/` applied against "`<p>PHP</p>`" will cause the entire string to be matched
 - Patterns can be made un-greedy by using the ? meta character
 - `/<.*?>/` will match "`<p>`" in "`<p>PHP</p>`"



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 115

Pattern Modifiers

- PCRE can be modified by appending various modifiers at the end of the declarative statement
 - Regular PCRE `preg_match('abc/', $data);`
 - PCRE with a pattern modifier `preg_match('/abc/i', $data);`
 - Available modifiers

i - caseless	Alphabetical case is ignored for any alphabetical comparisons made
m - multiline	By default, PCRE treats a subject string as a single line of characters even if there are newline characters present. Setting the "m" modifier causes the ^ and \$ to match the start and end of a line, not the start and end of the string.
s - dot = all	Typically the "." meta-character will not match newline characters. With "s" set it will match newline characters

Pattern Modifiers (continued)

x - extended	Whitespace data characters are ignored except when escaped allowing you to enter in comments
e - eval	Evaluates the replacement string as PHP code. Use sparingly and with care
A - anchored	Force the pattern to match only from the beginning of the string. From a conceptual standpoint it functions as though \A were automatically added to the pattern.
D - Dollar Endonly	\$ will match the end of the subject string. Without this modifier \$ matches immediately before the final character. This modifier is ignored if "m" is set
S - Analyze	If a pattern is going to be used several times setting "S" will cause the pattern to have some extra analysis done on it to speed up matching



Regular Expressions (continued)

U – Ungreedy	Inverts the greediness inherent in regular expressions. Expressions become not-greedy by default and the ? meta-character causes the regular expressions to become greedy
X – Extra	Turns on extra functionality that is not compatible with Perl. Any backslash in a pattern followed by a letter that does not have special meaning (\ [AZwWbB . .]) would cause an error
U – UTF-8	Causes pattern strings to be treated as UTF-8



Pitfalls

- **Regular expressions are relatively expensive in terms of performance**
 - Often the `ctype_*` or `str*` () functions are sufficient
 - `ctype_alnum`
 - `ctype_digit`
 - `strpos($var, 'abc') !== false`
- **Beware of ^ and \$**
 - When using the multiline pattern modifier (`/m`) they may not match from the beginning of the string, allowing data to be injected that was not filtered. Using `\A` and `\Z` are much safer when matching the entire string



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 119

Slide 120

Exercise: Creating an Expression

- **Create a regular expression to search for “C++” or “Java” in a file named “myfile.txt” and replace those strings with “PHP”**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 120

Regular Expression Resources

- **There are several variations supported in PHP:**
 - Perl-compatible (<http://us3.php.net/pcre>)
 - Differences from Perl and pattern syntax are at
<http://us3.php.net/manual/en/reference.pcre.pattern.syntax.php>



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 121

Slide 122

Module Summary

- Q & A



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 122

Slide 123



VI: PHP Web Concepts

Slide 124

Web Concepts Introduction

- **In this class we will cover:**
 - Review of last class
 - Differences between local applications and web-based
 - How servers communicate (an introduction)
 - GET
 - POST
 - HTTP Headers
 - Cookies and Sessions
 - Buffers, Email, and Browser Caching
 - Forms and Form Validation



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 124

Slide 125

Review from Last Class

- **Review**



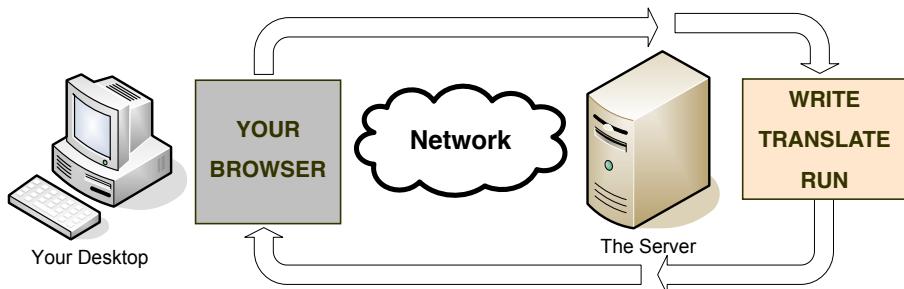
The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 125

Slide 126

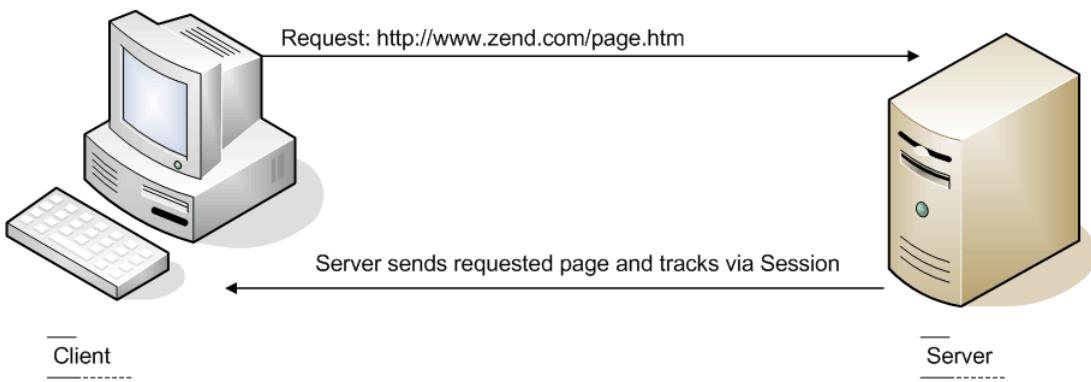
Client/Server Programs



- In Client/Server development (as with PHP), the program doesn't run on your desktop
- A request is made to a computer through a network, which runs the program and returns a response

Slide 127

How Servers Communicate (continued)



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 127

Review: How Servers Communicate

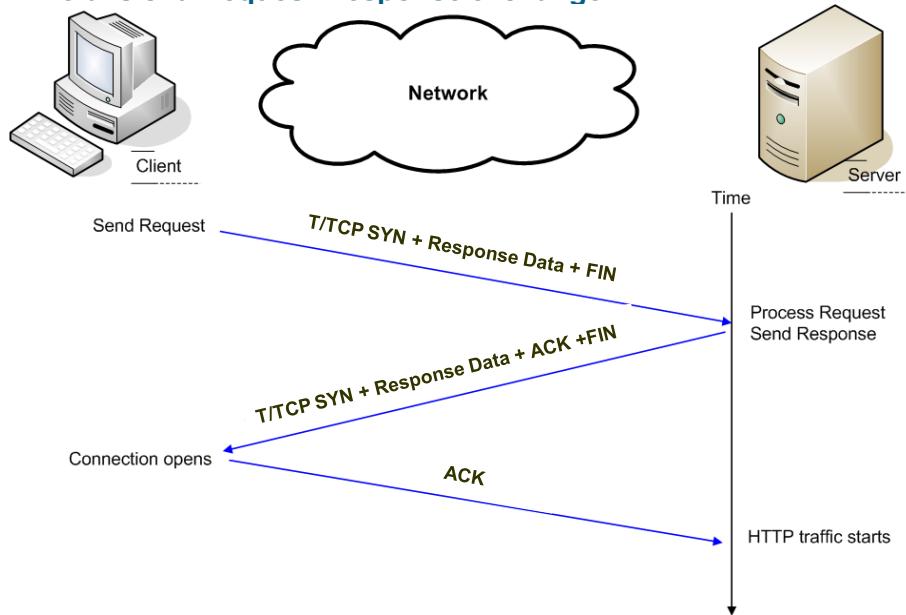
- **HTTP (HyperText Transfer Protocol) is the way messages get transferred on the Internet**
 - That's why you see <http://> in your browser when you type in web site names
- **HTTP doesn't “remember” the identity of each client who connects to a web site and treats every request for a web page as an independent request**
 - This is called ‘stateless’
- **To get around this, you can use a session or cookies**
- **Servers communicate using sessions and the GET and POST commands**



Slide 129

Server Communication Illustrated

- Details of a Request-Response exchange



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 129

Slide 130

Server Communication

- For any connection that is made from a browser to PHP there must be an IP address and a port for both the client and the server
- Typically HTTP-based communication is on port 80 or port 443 for SSL



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 130

HTTP Headers Overview

- An HTTP Header is a stream of text that is used by both client and server
- If you've ever seen code that looks something like the following, then you've seen a HTTP header:

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.27 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT Etag: "3f80f-1b6-
3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 131

- Do you recall what HTTP stands for?

Slide 132

What is Special About HTTP?

- **HyperText Transfer Protocol**
 - Governs most Internet traffic
 - Many other protocols are dependant on it such as:
 - HTTPS
 - Soap
 - Ajax
 - XML-RPC
 - WEBDAV – Extends HTTP 1.1
 - Uses TCP/IP as its transport
 - HTTP requests and responses both have required headers and optional bodies



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 132

Why Are Headers Important?

- **Response (`header()`)**

- Content-type (text/plain, text/html, text/xml, image/jpeg)
- Setting the response code (200, 403, 404, 500, 401, 301, 302 (Location))
- Setting client cache settings
- Determining charset
- http://en.wikipedia.org/wiki/P3P_Setting_p3p
- <http://www.fargs.org/rfcs/rfc2616.html>

- **Request (`$_SERVER`)**

- Accept
- User-Agent
- Accept-Language
- Accept-Encoding
- Host



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 133

Slide 134

HTTP Request Methods

- **Request methods**

- OPTIONS: GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT
- Most common are GET and POST
- GET and POST are essentially the same except that POST allows for a request body that is zero or more bytes in size



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 134

GET and POST Review

- **GET and POST functions**

- GET retrieves an array of variable names and values from a form
 - Values are shown in the URL and should not be used with sensitive information
 - They do allow you to bookmark that page
 - Different browsers have different limitations on GET variable size
- POST submits an array of variable names and values to be processed
 - Data is less visible to others and has no size limits

- **`$_GET` and `$_POST` variables**

- `$_GET` is an array of variable names sent by GET
- `$_POST` is an array of variable names and values sent by POST



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 135

HTTP GET Details

- **GET**

```
GET /test.php?name=John&20Smith HTTP/1.1
Host: zend.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1;
en-US; rv:1.8.1.9) Gecko/20071025 Firefox/2.0.0.9
Accept:
    text/xml,application/xml,application/xhtml+xml,text/
    html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```



HTTP POST Details

- **POST**

```
POST /test.php HTTP/1.1
Host: zend.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.8.1.9) Gecko/20071025 Firefox/2.0.0.9
Accept:
    text/xml,application/xml,application/xhtml+xml,text/html;q=
    0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://zend.com/test.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 54

name=John+Smith&myfriend=on&Submit+Button=Submit+Query
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 137

HTTP POST Using MIME Encoding

```
POST /test.php HTTP/1.1
Host: www zend com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.9)
Gecko/20071025 Firefox/2.0.0.9
Accept:
    text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;
    q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www zend com/test.php
Content-Type: multipart/form-data; boundary=
-----156982912616038
Content-Length: 370

-----156982912616038
Content-Disposition: form-data; name="name"
John Smith
-----156982912616038
Content-Disposition: form-data; name="myfriend"
on
-----156982912616038
Content-Disposition: form-data; name="Submit Button"
Submit Query
-----156982912616038--
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 138

Exercise: GET and POST

- Write a form that submits both GET and POST data at the same time
- Have it `var_dump($_GET, $_POST)` when it's submitted



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 139

Slide 140

Other Web Concepts

- **Here are some other web-related concepts we will examine:**
 - Buffer
 - Cache
 - Cookies and Sessions (review and further examination)
 - Email



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 140

Buffers Overview

- **Output buffering means no output will be sent to the web server until either the PHP script ends or the buffer is filled up**
 - This allows PHP scripts to side-step an HTTP limitation - that is, once content-output has been sent, no more headers (new sessions, cookies) can be sent
- **Output buffering can be used to not only improve performance but make handling of errors, cookies, etc., easier**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 141

- Buffers are used to improve performance by collecting data and then sending it to the client in the most efficient way
 - Sending one 1024-byte chunk is more efficient, and gives a smoother perceived performance, than 1,024 one-byte chunks

Output Buffering Functions

- Earlier, we looked at how to turn output buffering on and off in the PHP.INI; these functions can be used in conjunction with output buffering
 - `ob_clean`: Cleans (erases) the output buffer
 - `ob_end_clean`: Cleans (erases) the output buffer and turns off output buffering
 - `ob_end_flush`: Flushes (sends) the output buffer and turns off output buffering
 - `ob_flush`: Flushes (sends) the output buffer
 - `ob_get_clean`: Gets current buffer contents and deletes current output buffer; useful for full-page caching
 - `ob_get_contents`: Returns the contents of the output buffer



Example: Buffers

```
ob_start();
echo 'I want this in the buffer';
$val = ob_get_clean();
echo strlen($val);

// echos 25, the number of characters in the output // buffer
// "I want this in the buffer" is never echoed
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 143

Exercise: Buffers

- Use output buffering in `index.php` to make all content upper case
(Note that this will break HTML markup)

```
// Start code
$dispatcher = new ZBlog_Dispatcher();
$dispatcher->dispatch();
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 144

Browser Caching

- **HTTP provides several methods for caching information on the browser or a proxy**
 - Etag – Used to provide a token associated with a given resource.
 - For example, on October 20th the ETag value for /index.php is abc123
 - Browser sends back the ETag abc123 to the browser and responses to the subsequent requests to the page cause a status of **304 Not Modified** to be sent back to the browser with an empty HTTP response body
 - Overnight the content changes and so the new ETag value is abc124
 - Browser makes a request but the ETag does not match the current one so the web server creates a new page for the browser



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 145

Example: Browser Caching

- **ETag Example**

```
$time = 1195248471;
$eTag = md5($);

$noneMatch =
isset($_SERVER['HTTP_IF_NONE_MATCH']) ? $_SERVER['HTTP_IF_NONE_MATCH'] : '';

if ($noneMatch == $eTag) {
    header('304 Not Modified', true, 304);
    exit;
}
header("ETag: $eTag");
```



Browser Caching: Last-Modified

- **Last-Modified**

- Browsers will typically store the last request of a page in the browser cache
- When the browser makes a request for a page that it already has in the cache it will send a Last-Modified HTTP header which notes what the date was on that cached version
- If the date is after the last change to that page send a 304 header response back to the browser with an empty body



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 147

Example: Last-Modified

- **Last-Modified example**

```
$headerTime =  
    isset($_SERVER['HTTP_IF_MODIFIED_SINCE']) ? strtotime($_  
    SERVER['HTTP_IF_MODIFIED_SINCE']) : time();  
    $time = 1195248471;  
  
    if ($headerTime > $time) {  
        header('304 Not Modified', true, 304);  
        exit;  
    }  
    header('Last-Modified: '.gmdate('D, d M Y H:i:s  
    \G\M\T', time()));
```



Browser Caching

- Default headers may be sent, negating your attempt at browser caching so the following code is suggested when using browser caching

```
header('Expires: '.gmdate("D, d M Y H:i:s",
    $entry->modified+31536000) . " GMT");
header("Pragma: cache");
header("Cache-Control: public, must-revalidate,
max-age=0");
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 149

Home Work: Browser Caching

- Change the article controller to use browser caching when viewing an article.
- Verify this by tailing the
`/usr/local/Zend/apache/logs/access_log`
for 304 return codes when requesting a page

```
// Starting Code
if ($entryId != 0) {
    $entry = ZBlog_Model_Blogs::getEntry($entryId);
}
```



Slide 151

Browser Detection

- **Uses the `$_SERVER['HTTP_USER_AGENT']` variable**
- **Examples include:**
 - Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.9) Gecko/20071025 Firefox/2.0.0.9
 - Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; EmbeddedWB 14.52 from: <http://www.bsalsa.com/> EmbeddedWB 14.52; .NET CLR 1.1.4322; InfoPath.1; .NET CLR 1.0.3705; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30



Copyright © 2006-2009, Zend Technologies Inc. | 151

- Note `get_browser()` on php.net

Easy Checking Examples

- Easy browser detection checking for Internet Explorer

```
$isIE = strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== false;
```

- Easy checking for multiple browsers

```
$ua = $_SERVER['HTTP_USER_AGENT'];
switch (true) {
    case strpos($ua, 'Safari') !== false: $browser = 'Safari';
    break;
    case strpos($ua, 'Firefox') !== false: $browser = 'Firefox';
    break;
    case strpos($ua, 'MSIE') !== false: $browser = 'IE';
    break;
    default:           $browser = 'Unknown';
    break;
}
```



Cookies Review

- **What is a cookie?**
- **When you log in to a web site that says “Hello, Lynn” (and your name is Lynn), the site is using data stored from a previous session in a cookie (persistent data between sessions)**
 - Usually cookies are set to expire after a set period; for highly secure sites, cookies will not persist
 - Cookies are often used by vendors to track your behavior and preferences
 - **Syntax for setting a cookie:**
`setcookie(name, value, expire, path, domain);`
 - **Example of setting a cookie:**
`setcookie("user", "Tom Tobin", ()+3600);`
 - **Retrieving a cookie with the \$_COOKIE global variable:**
`echo $_COOKIE["user"];`
 - To delete a cookie, set the expiration date to the past



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 153

What is Special About Cookies?

- Used to store persistent data on the browser
- Set in an HTTP response using the header Set-Cookie

```
200 OK
Date: Fri, 16 Nov 2007 22:06:02 GMT
Server: Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.8d DAV/2
          SVN/1.4.3 Zend Core/2.0.1 PHP/5.2.1
Content-Type: text/html
Set-Cookie: cookie=value
```

- Cookie values are retrieved from the `$_COOKIE` superglobal as an associative array, including the PHP session ID

```
array(1) {
    ["cookie"]=> string(5) "value"
}
```



Setting Cookies

- Set cookies using the `setcookie()` function

```
bool setcookie (
    string $name
    [, string $value
    [, int $expire
    [, string $path
    [, string $domain
    [, bool $secure
    [, bool $httponly]]]]])
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 155

Cookies and Security

- **Security warnings**
 - Do not depend on data in the cookie

```
if ($_COOKIE['isAdmin']) {  
    echo 'Welcome Admin';  
}
```

- As cookie information is stored on the browser, it is easy to forge
- Do not store sensitive data in a cookie - use a session instead
 - If you must store sensitive information on the browser, at the very least use some kind of synchronous encryption on the value

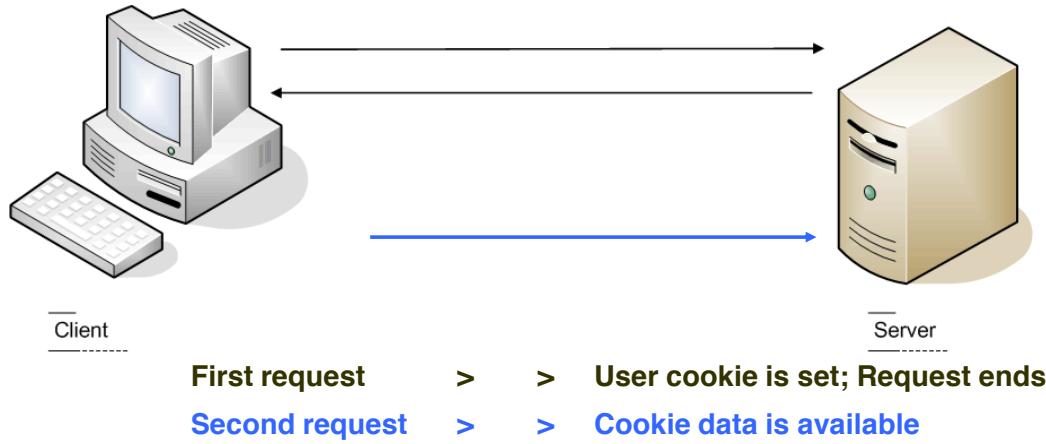


- **Example:**
 - Forged HTTP request

```
GET /admin HTTP/1.0  
Cookie: isAdmin=1
```

Cookies Data Availability

- **Do *not* expect to have your cookie values available after the first request... this is a very important point**
- **Cookie values are not available to PHP until the *second* request by the user**



Slide 158

Home Work: Cookies

- Write code that will hide the preview text on the front page and store the setting in a cookie
- Make sure to note the path
- Instructor will get you started... indicate where you need to make changes in the code...



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 158

Sessions Review

- **What is a session?**
- **Sessions track and maintain client-specific information during a single visit to a web site. That information is stored in a session variable `$_SESSION`**
 - `$_SESSION` is a *superglobal*, so you can use it inside and outside functions without needing to declare it as global first
 - `session_start()` checks if a session exists or creates one if it doesn't
 - `session_destroy()` destroys the session, call it when you are finished
 - `unset($_SESSION['variable'])` frees a specified session variable



Copyright © 2006-2009, Zend Technologies Inc. | 159

You can use a session to store connection-specific data; this session data is preserved on the server for the duration of the visit, and is destroyed on its conclusion. Sessions work by associating every session with a session ID (a unique identifier for the session) that is automatically generated by PHP. This session ID is stored in two places: on the client using a temporary cookie, and on the server in a flat file or a database. By using the session ID to put a name to every request received, a developer can identify which client initiated which request, and track and maintain client-specific information in session variables (variable-value pairs which remain alive for the duration of the session and which can store textual or numeric information).

- The most common method of tracking a customer through a web site is by assigning a unique session ID - and having this information transmitted back to the web server with every request. Unfortunately, should an attacker guess or steal this session ID information, it is normally a trivial exercise to hijack and manipulate another user's active session. So you need to set up authentication processes as well. (which is a more advanced topic)
- Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.
- `session_start()` function. This function checks to see whether a session already exists, and either restores it (if it does) or creates a new one (if it doesn't). Session variables can then be registered by adding keys and values to the special `$_SESSION` superglobal array, and can be accessed at any time during the session using standard array notation
- <?php

```
// initialize a session
session_start();

// print session ID
echo "I'm tracking you with session ID " . session_id();
// then destroy the session
session_destroy();
?>
```

- `$_SESSION` is a superglobal, so you can use it inside and outside functions without needing to declare it as global first
↳ this is the definition of a superglobal...

Slide 160

What is Special About Sessions?

- **Handled internally using the `ext/session` handler**
 - 3rd party session handlers can be written to handle session data storage
 - Can be done as a compiled C extension set using `session.save_handler` in `PHP.INI`
 - Can be done using PHP code by setting `session_set_save_handler()` prior to calling `session_start()`
- **Sessions are referenced using either a cookie or URL parameter**
- **Default cookie and parameter name ‘`PHPSESSID`’ can be changed in `PHP.INI` via the `session.name` configuration option**
- **Sessions are enabled by default and must be explicitly disabled**

About Sessions – continued

- Data to be kept persistent between HTTP requests is serialized and stored on the hard drive by default
- During the course of an HTTP request data can be accessed by referencing the `$_SESSION` variable



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 161

Sessions – PHP.INI

- **session.save_path**
 - Typically points to /tmp as the directory to store session data
- **session.name**
 - The name of the cookie or URL parameter that contains the session ID
- **session.save_handler**
 - Sets the handler that manages session data storage
 - Included in PHP is
 - files (default)
 - mm – shared memory
 - mysql
 - memcache
 - cluster (Zend Platform)
 - and many more...



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 162

Starting and Ending a Session

- Here is the basic code for starting and ending a session

```
<?php  
  
    // begin a session  
    session_start();  
  
    // print session ID  
    echo "The session ID is " . session_id();  
  
    // then end the session  
    session_destroy();  
  
?>
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 163

Slide 164

Example – Sessions

- Now, we'll take a look at the Sessions code in the following files:
 - loginController.php
 - logoutController.php



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 164

Exercise: Sessions

- Write some code so that the last article that was read is displayed on the left hand side
 - Starting code is the `Columnleft.php` helper and the article controller



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 165

Email Overview

- **PHP contains rudimentary methods of sending emails**
 - For more advanced email creation, such as MIME email and attachments consider Zend Framework
- **To send email from PHP use the `mail()` function**

```
bool mail (
    string $to,
    string $subject,
    string $message
    [, string $additional_headers
    [, string
    $additional_parameters]]
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 166

mail() Parameters

- **mail() parameters explained**
 - *to*
 - Receiver, or receivers of the mail
 - The formatting of this string must comply with [» RFC 2822](#). Some examples are:
 - user@zend.com
 - "User" user@zend.com
 - "User" <user@zend.com>
 - *subject*
 - Subject of the email to be sent
 - Newline characters should be removed from the subject. Not doing so could cause poorly formatted emails or create emails with injected content such as adding recipients
 - *message*
 - The message to be sent



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 167

mail() Parameters – continued

- ***additional_headers* (optional)**
 - String to be inserted at the end of the email header
 - This is typically used to add extra headers (From, Cc, and Bcc). Multiple extra headers should be separated with a CRLF or carriage return line feed (\r\n)
 - **Note:** When sending mail, the mail *must* contain a *From* header. This can be set with the *additional_headers* parameter, or a default can be set in *PHP.INI*



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 168

Additional mail() Parameters

- ***additional_parameters* (optional)**
 - The *additional_parameters* parameter can be used to pass an additional parameter to the program configured to use when sending mail using the *sendmail_path* configuration setting. For example, this can be used to set the envelope sender address when using sendmail with the *-f* sendmail option.



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 169

Configuration Email Delivery Options

- **Mail delivery options in PHP.INI**
 - *SMTP* string
 - Used under Windows only: host name or IP address of the SMTP server PHP should use for mail sent with the `mail()` function
 - *smtp_port* int
 - Used under Windows only: Number of the port to connect to the server specified with the *SMTP* setting when sending mail with `mail()`; defaults to 25
 - *sendmail_from* string
 - Which "From:" mail address should be used in mail sent from PHP under Windows. This directive also sets the "Return-Path:" header
 - *sendmail_path* string
 - Where the sendmail program can be found, usually `/usr/sbin/sendmail` or `/usr/lib/sendmail`. configure does an honest attempt of locating this one for you and set a default, but if it fails, you can set it here



Internet Message Access Protocol (IMAP)

- **IMAP**
 - Internet Message Access Protocol
 - Typically used when email needs to be stored in a central location
 - Uses the imap extension which is not compiled by default
- **Example code:**

```
$ds = imap_open("{localhost:143}INBOX", 'user', 'password') ;  
echo imap_num_msg($ds) ;  
imap_close($ds) ;
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 171

Post Office Protocol (POP)

- **POP**
 - Post Office Protocol
 - Typically used to temporarily store email in a central location prior to being downloaded to an end user's computer
 - Uses the `imap` extension
- **Example code**

```
$ds = imap_open("{localhost/pop3:110}INBOX",
    'user', 'password');
echo imap_num_msg($ds);
imap_close($ds);
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 172

Email

- You can add email functionality to your web application
 - Mail function: `mail(to, subject, message, headers, parameters)`
 - The last two parameters are optional and sending e-mail is dependent on your server configuration
 - if `sendmail` is not installed on the server when PHP is compiled, the `mail()` function won't compile and can't be used

```
<?php
$to = "lucky_person@zend.com";
$subject = "Test mail";
$message = "This is sent from a PHP script.";
$from = "employee@zend.com";
$headers = "From: $from";
mail($to,$subject,$message,$headers);
?>
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 173

Note: if you don't have sendmail installed on your server when you compile PHP the `mail()` function isn't even compiled at all and you can't use it.

- You can also send email via a form.
- <?php
- if (isset(\$_REQUEST['email']))
 - //if "email" form is filled out, send email
 - {
 - //send email
 - \$email = \$_REQUEST['email'];
 - \$subject = str_replace("\n", "", \$_REQUEST['subject']);
 - \$message = \$_REQUEST['message'];
 - mail("lucky_person@zend.com", "Subject: \$subject",
 - \$message, "From: \$email");
 - echo "Sent via mail form";
 - }
 - else
 - //if "email" is not filled out, display the form
 - {
 - echo "<form method='post' action='mailform.php'>
 - Email: <input name='email' type='text' />

 - Subject: <input name='subject' type='text' />

 - Message:

 - <textarea name='message' rows='15' cols='40'>
 - </textarea>

 - <input type='submit' />
 - </form>";
 - }

Slide 174

Exercise: Email

- Write some code in `articleController.php` that emails an administrator, defined in the config file, when a new comment is made



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 174

Forms

- If you are new to web programming, we'll briefly talk about HTML tables and working with HTML forms
- Then we'll spend some time discussing
 - Form submission
 - Form validation



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 175

HTML Tables Review

- Here is a simple HTML table with PHP code in it
- The data contained in the variables can come from a database, or it may be created by running a PHP script

```
//Put the results in an HTML table
print("<table>\n");
print("<tr bgcolor=#ffffff>\n");
print("<td><b>$custid</b></td>\n");
print("<td><i>$custname</i></td>\n");
print("<td><i>$custaddr</i></td>\n");
print("</tr>\n");
print("</table>\n");
```



Creating HTML Forms

- An **HTML form contains HTML content along with fields and checkboxes, radio buttons, menus, and other controls**
- People enter data, then submit the form to a web or mail server

```
<html>
<body>

<form action="users.php" method="post">
    First Name: <input type="text" name="fname" />
    Last Name: <input type="text" name="lname" />
    <input type="submit" />
</form>
</body>
</html>
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 177

Form Submission Overview

- Form variables are passed to PHP scripts by several superglobal associative arrays
 - `$_GET`
 - Data passed through the query string in a URL
 - <http://zend.com/test.php?querystring=1>
 - Can be set for both GET and POST requests
 - `$_POST`
 - Data passed during a POST request
 - `$_REQUEST`
 - Contains data from both the GET and POST request
-  **SECURITY TIP :** Often by retrieving data through `$_POST` you can help ward off *cross site request forgeries* and *cross site scripting*



Submitting Numerical Arrays

- **Numerical arrays**

```
<form method="post">
<input type="checkbox" name="option[]" value="opt1" checked>
<input type="checkbox" name="option[]" value="opt1" checked>
<input type="submit">
</form>

array(1) {
    ["option"]=> array(2)
    {
        [0]=> string(4) "opt1"
        [1]=> string(4) "opt1"
    }
}
```

Submitting Associative Arrays

- **Associative Arrays**

```
<form method="post">
<input type="checkbox" name="option[key1]"
value="opt1" checked>
<input type="checkbox" name="option[key2]"
value="opt1" checked>
<input type="submit">
array(1) {
    ["option"]=> array(2) {
        ["key1"]=> string(4) "opt1"
        ["key2"]=> string(4) "opt1"
    }
}
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 180

Submitting Multidimensional Arrays

- **Multidimensional Arrays**

```
<form method="post">
<input type="checkbox" name="option[key1][]" value="opt1" checked>
<input type="checkbox" name="option[key1][]" value="opt2" checked>
<input type="submit">
</form>
array(1) {
    ["option"]=> array(1) {
        ["key1"]=> array(2) {
            [0]=> string(4) "opt1"
            [1]=> string(4) "opt2"
        }
    }
}
```



Form Validation

- Client-based validation is faster and reduces server load, but it can be by-passed, causing potential security and data validation problems
- You can use regular expressions to validate code, but what are some other ways to validate data?
 - `preg_*`
 - `ctype_*`
 - String functions (`strpos`)



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 182

- "Why use client side validation at all?" The reason is, that high traffic web sites, should seize the opportunity to take off the load of the server and distribute it to the client browser. This means that if you can verify the content of a field before it is submitted and processed by the server, it makes sense to do so. And there is a user friendly side of it as well. Since most people assume that once they have clicked the submit button on a form, the process is over. A nifty popup explaining what is missing or incorrect, improves their chance of entering correct data into the form. Who wants to miss out on that lottery jackpot just because he or she forgot to verify the data they entered on an online entry form

Examples: Form Validation

- **Checking to see if a form variable is set**
 - `isset($_POST['name'])`
- **Simple validation should be done using Character Type checks**
 - Validating a username
 - `ctype_alnum($_POST['username'])`
 - Check a year
 - `ctype_digit($_POST['year'])`
- **Complex validation of an email address**
 - `preg_match('/^ [A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/, $_POST['username'])`

Exercise: Form Validation

- What type of validation should be done for each of the following types (`preg_*`, `ctype_*`, `str*`)

1) Age

`ctype_digit`

2) Phone number

`preg_match`

3) Text contains the word "Zend"

`strpos`

4) Text must be less than 255 characters

`strlen`

5) Contains a first and last name

`strpos(' ')`



Slide 185

Module Summary

- Q & A



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 185

Slide 186



VII: PHP Object-Oriented Programming

OOP Introduction

- **In this class we will cover:**
 - Review of last class
 - Object-Oriented Programming
 - Classes



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 187

Slide 188

Review from Last Class

- **Review**
 - Server Communication
 - GET and POST
 - Working with forms and form validation



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 188

What is an Object?

- **An instance of a given class**
- **Generally represents data and/or functionality that can be logically grouped**
 - Mobile phone
 - Calling (method)
 - Receiving calls (method)
 - Mute key (method)
 - Phone numbers (properties)
 - Photos (properties)



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 189

What is a Class?

- **It is a blueprint of properties and methods that pertain to a given concept**
 - Could include, but is not limited to, concepts such as
 - Users
 - Pages
 - Permissions
 - Roles
 - Groups of Users
 - Groups of Pages
 - Groups of Permissions
 - Groups of Roles
- **Objects are instances of a given class that represent real data or functionality**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 190

Defining Classes

- Defining a class is done using the ***class*** keyword

```
class User {}
```

- That is a basic class that can now be instantiated with

```
$user = new User()
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 191

Creating Objects

- **Objects are created using a constructor**
 - In PHP 4, the constructor was given the same name as the class
 - In PHP 5, you can still name a constructor according to the class but it is preferred that you use the keyword `__construct()`
- **To create an object based on the class use the `new` keyword**
 - `$user = new User();`



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 192

Defining Classes `__construct()`

- If the class has some default logic that needs to be executed, it is placed in the constructor
 - This default information could be an XML document that needs to be parsed or a database connection that needs to be queried to populate the object

```
class User {  
    function __construct() {  
        // Connect to database  
    }  
}
```



Defining Classes `__construct($param)`

- Constructors, like regular functions, are able to take parameters to customize behavior
 - For example, loading up data from a database row that the object will represent

```
class User {  
    function __construct($id = null) {  
        if ($id == null) { return; }  
        // load data from the database  
    }  
}
```



Destroying Classes `__destruct()`

- **A destructor is called when an object is**
 - Out of scope
 - Explicitly unset
- **During cleanup after the script has been completed**
- **Can be used to close database connections or remove temporary files**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 195

Slide 196

Exercise: Creating Classes

- **Which of the following constructors are valid in PHP for the class myClass?**
 - a) void myClass()
 - b) function __construct()
 - c) const()
 - d) function myClass()
 - e) sub myClass()



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 196

Defining Classes: Properties

- **Classes can have data attached to the individual instance of the class**

```
class User {  
    $name = 'My Name';  
}
```

- **That data is accessible using the `->` characters**

```
$user = new User();  
echo $user->name;
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 197

Defining Classes: Methods

- **Classes can have individual functions available on that instance called methods**

```
class User {  
    function getName() {  
        return 'My Name';  
    }  
}
```

- **Methods are accessible in a similar manner as properties**

```
$user = new User();  
echo $user->getName();
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 198

Static Context

- **Class members may be defined as static**
 - The word “static” means a fixed or stationary condition
- **In OOP, this refers to using a *class’s* functionality as opposed to an *object’s* functionality**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 199

Static Context (continued)

- Denoted by the `static` keyword

```
class Functions {  
    static function getCurrent() {  
        return date('m');  
    }  
}  
  
echo Functions::getCurrent();
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 200

Exercise: Static Context

- Create a static method called `changeCase()` in a class called `Functions` that will change a given string to lowercase or uppercase based on a parameter

```
<?php
$string = "Hello, World\n";

echo Functions::changeCase($string, true);
echo Functions::changeCase($string, false);

// Insert the code necessary to make this work
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 201

Class Constants

- PHP constants are typically declared using the `define()` function

```
define('COMPANY', 'Zend');
```
- This can lead to naming collisions and a pollution of the global namespace
- Keep related constants within a class definition

```
class User {  
    const TYPE_GUEST = 1;  
    const TYPE_USER = 2;  
    const TYPE_ADMIN = 3;  
}  
$user = new User(User::TYPE_GUEST);  
If ($user->type === User::TYPE_GUEST) {  
    echo 'User is a guest';  
}
```



Visibility: PPP

- In object-oriented programming, every property and method can have a specific visibility declared for it
 - Public
 - Protected
 - Private
- Visibility is determined on the code-level, not application level
 - Determines which code can access which class members



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 203

Visibility: Public

- **Public methods can be called by any code in your application**
- **Public properties can be modified by any code in your application**

```
$user = new User();
$user->name = ' Bob' ;
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 204

Visibility: Protected

- **Protected class members are accessible only to classes that extend that class and itself**

```
class User {  
    protected function userAge() { }  
}  
  
class Person extends User {  
    public function getUserAge() {  
        return $this->userAge();  
    }  
}  
  
$user = new Person();  
echo $user->getUserAge(); // Succeeds  
echo $user-> userAge(); //Throws a fatal error
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 205

Visibility: Private

- **Private class members are accessible only to the class itself**
- **Neither child classes nor external code can access them**
 - If calling a private member owned by the class (not extended) a fatal error will be displayed
 - If calling a private member on an extended class an “undefined property” notice will be displayed

```
class myClass {  
    private $var = 1;  
}  
class myClass2 extends myClass {}  
$c1 = new myClass();  
$c2 = new myClass2();  
echo $c1->var; // Fatal error  
echo $c2->var; // Undefined property notice
```



Exercise: Visibility

- Look at the `adminController.php`
- determine which methods are Public and which are Private



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 207

Inheritance

- One of the cornerstones of OOP is the ability for classes to build upon other classes

```
class myClass {  
    public $value = 'Inherited';  
}  
  
class newClass extends myClass {}  
  
$obj = new myClass();  
echo $obj->value; // Inherited
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 208

Exercise: Inheritance

- What is the output of the following code:

```
<?php
class myClass1 {
    private $var = 1;
}
class myClass2 extends myClass1 {
    public function getVar()
    {
        return $this->var;
    }
}
$obj = new myClass2();
echo $obj->getVar();
?>
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 209

Overriding Functions

- If the class you're writing has more pertinent logic, then you can override the function defined in the parent class

```
class User {  
    public function doUserStuff() {  
        // Do something  
    }  
}  
  
class AdminUser extends User {  
    public function doUserStuff() {  
        // Do something better  
    }  
}
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 210

Overriding Functions (continued)

- What if the parent function still has pertinent functionality in it?

```
class User {  
    public function doUserStuff() {  
        // Do something  
    }  
  
    class AdminUser extends User {  
        public function doUserStuff() {  
            // Do something different  
            // Then do parent thing  
            parent::doUserStuf();  
        }  
    }  
}
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 211

Overriding Functions (continued)

- **parent:: will call parent method but \$this-> will call object's method**

```
class test1 {
    public function f2() {
        echo 'f2';
    }
}
class test2 extends test1 {
    public function f1() {
        echo $this->f2();
        echo parent::f2();
    }
    public function f2() {
        echo 'f3';
    }
}
$test = new test2();
$test->f1(); // f3f2
```



Exercise: Overriding Functions

- **What is the result of this code?**

```
<?php
class myClass1 {
    public function test() {
        return 1;
    }
    public function test2() {
        return $this->test();
    }
}
class myClass2 extends myClass1 {
    protected function test(){
        return 2;
    }
}
$obj = new myClass2();
echo $obj->test2();
?>
```



Type Hinting

- When dealing with functions or methods you can explicitly request object types as specific parameters
- Passing incompatible variables causes a fatal error

```
function myFunction(Countable $i) {  
}  
$a = new ArrayObject();  
myFunction($a);
```



Polymorphism

- Child objects can be acted upon like parent classes

```
function doSomething(User $obj)
{
    echo $obj->getName();
}
class User{
    public function getName() {
        //... do something
    }
}
class AdminUser extends User {}

$user = new AdminUser();
doSomething($user);
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 215

Interfaces Overview

- **Provides a blueprint for a class**
- **Useful for ensuring that specific functions will be available**
- **Can be used to define interaction with other code, without implementing any of that functionality**
- **Classes can be derived from more than one interface**
 - **Note:** Classes derived from more than one interface cannot have a method naming collision between the interfaces
- **If a class implementing an interface does not implement all the methods in the interface, it must be declared abstract and is not instantiable**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 216

Example: Interfaces

- Implemented with the `implements` keyword
- Contains no executable code

```
interface UserInterface {  
    function getName();  
}  
  
class User implements UserInterface  
{  
    function getName() {  
        return 'My Name';  
    }  
}
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 217

Interfaces (continued)

- Can be used to type-hint variables when passing to functions

```
interface UserInterface {
    function getName();
}

class User implements UserInterface {
    function getName() {
        return 'My Name';
    }
}
function getUserName(UserInterface
$user) {
    return $user->getName();
}
$user = new User();
echo getUserName($user);
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 218

Exercise: Interfaces

- What is the result of this code

```
<?php
interface myInterface {
    function test();
}
class myClass extends myInterface {
    public function test() {
        echo 'test';
    }
}
function myFunction(myInterface $obj) {
    $obj->test();
}
$obj = new myClass();
myFunction($obj);
?>
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 219

- <http://www.php.net/~helly/php/ext/spl/>

Abstract Classes Overview

- Contain a skeleton, like an Interface, but also contain code
- Cannot be instantiated
- Classes that extend an abstract class need to be instantiable or must be declared abstract as well

```
abstract class UserAbstract {  
    public function getName() {  
        return 'My Name';  
    }  
    abstract public function getPermissions();  
}  
class User extends UserAbstract {  
    public function getPermissions() {}  
}
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 220

Preventing Overriding

- Use the **final** keyword
- Can be used on a class or a method

```
final class User {}  
class AdminUser extends User{} // Throws a fatal  
error  
  
class User {  
    final public function getName() {}  
}  
class AdminUser extends User{  
    public function getName() {}  
} // Throws a fatal error
```



How an Object Calls Itself

- **Object context**
 - `$this->`
- **Static context**

```
self:::  
    class myClass {  
        private $me = null;  
        private static $obj = null;  
        public function __construct() {  
            $this->me = 'My Name';  
        }  
        public static function getObject() {  
            self::$obj = new myClass();  
            return self::$obj;  
        }  
    }  
}
```



References

- All objects are passed by reference

```
$user1 = new User();
$user2 = $user1;
// $user1 and $user2 are references to the
// same object

function doSomething(User $obj) {
// $obj is $user1
}
doSomething($user2);
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 223

Cloning

- Since objects are passed as references by default, if you want to make a copy of the object use the `clone` keyword

```
$user1 = new User();
$user2 = clone $user1;
// $user1 and $user2 are now different objects
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 224

Cloning (continued)

- When clone is executed PHP checks for the `__clone()` magic function

```
class User {  
    public function __clone() {  
        throw new Exception("Don't clone me!");  
    }  
  
}  
  
$user1 = new User();  
$user2 = clone $user1; // Exception thrown
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 225

Comparing Objects

- **==** Checks to see if the objects are the same type and properties have the same values
- **===** Checks to see if 2 variables are the same object

```
$user1 = new User();
$user2 = clone $user1;

echo ($user1 == $user2)?'They are =' : 'They are !=';
echo ($user1 === $user2)?'The same' : 'Not the same';

// They are ==
// They are not the same
```



Slide 227

Module Summary

- Q & A



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 227

Slide 228



VIII: PHP Database Basics

Database Introduction

- **In this class we will cover:**
 - Review of last class
 - Review of relational databases
 - SQL and MySQL
 - Connecting PHP and MySQL
 - Database functions
 - Where to learn more



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 229

Slide 230

Review from Last Class

- **Review**
 - Classes
 - Objects



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 230

Review: Databases

- **Databases are sets of data organized in such a way that the information can be easily accessed, managed, and updated**
- **Data is stored in increasing precision**
 - Database
 - Table
 - Row
 - Column
- **Database terms common to most databases**
 - Table: contains data in structured rows/columns
 - Key: a unique value to locate/relate information in a table
 - Join: associates data from more than one table together
 - Query: statement that gathers information from a database
 - Schema: how the various tables map/relate to each other



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 231

Slide 232

Relational Databases

- Relational databases are a method of storing data where you can 'relate' or match pieces of data from different sources (data tables)
- Relational Database Management Systems (RDBMS)
 - Faster than flat files
 - Relate multiple pieces of data together
 - Produce robust reports (queries)
 - Designed to eliminate or minimize redundant data



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 232

Introduction to RDBMS (continued)

- **Data stored in individual columns in individual tables can be related to data in other tables in other columns using a JOIN**
- **For example, 2 tables:**
 - User table
 - Permission table
 - Data in both may be joined in a single query allowing you to retrieve all the pertinent



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 233

SQL Essentials

- Some commands (e.g., **SELECT**) within SQL are specific to running queries on data
- Other commands are used to add, delete, or edit data records
 - Known as SQL Data Manipulation Language (DML)
 - Include **INSERT INTO**, **DELETE**, **UPDATE**
- There are also commands to work specifically with tables (as opposed to data)
 - Known as SQL Data Definition Language (DDL)
 - Include **CREATE TABLE**, **DROP TABLE**, **ALTER TABLE**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 234

SQL Code Examples

- **Examples of writing SQL code to create a database and table:**

```
mysql> CREATE  
DATABASE fundamentals;
```

```
mysql> CREATE  
TABLE employees(  
empl_id INT AUTO_INCREMENT PRIMARY KEY,  
last_name VARCHAR(255) ;
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 235

- To learn about creating SQL queries in MySQL, see <http://dev.mysql.com/doc/>

In addition to writing queries you can also use the visual query builder in Studio.

Exercise: SQL Essentials

- Lynn and Dave are students in a Calculus I class, and achieved the following test scores for the semester:

| Lynn | Dave |
|--------|------|
| ▪ 80 | 80 |
| ▪ 87 | 87 |
| ▪ 90 | 90 |
| ▪ Null | 0 |
| ▪ 88 | 88 |
- The teacher records these scores in a DB and uses SQL to create a semester average.
- Will the two students have the same average? Why or why not?



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 236

Connecting PHP and MySQL

- **PHP provides built-in database functions to work with MySQL databases, as demonstrated below**

```
mysql_connect(servername,username,password);  
  
mysql_select_db("db_name", $connection_name);  
  
mysql_query("CREATE DATABASE db_name",$connection);  
  
mysql_query("INSERT INTO databasename (field1, field2)  
VALUES ('Tika', 'Gelsey')");  
  
mysql_fetch_array($result);  
  
mysql_close($connection_name);
```

- **Best practice is to use PDO, however; covered shortly**



- Connect, connects to a data source
- Select, specifies the database
- Query-allows you to query data or in these examples, create a database or insert data into a table. You can also update or delete data
- Fetch gets data from an array, usually the results of a query
- Close, closes the connection

Slide 238

SQLite

- **SQLite is a file based, embeddable database**
- **Extremely fast for simple operations**
- **Does not require user authentication to be used**
 - Benefit in small environments
 - Drawback in larger environments
- **Has OO, PDO and functional interface**



Copyright © 2006-2009, Zend Technologies Inc. | 238

- Also because of the lack of user authentication it is possible that SQL injection attacks will be more serious because you cannot limit actions based on the account

PDO in Detail

- **Lightweight, consistent interface for accessing databases in PHP**
- **Fully object-oriented interface**
- **Does not provide database abstraction, but rather database access abstraction**
- **Integrated transaction (ACID-compliant) handling**
 - An exception will be thrown if you begin a transaction on a database that does not support transactions
 - Initially runs in “auto-commit” mode
 - All queries are committed to the database
 - Overridden with the beginTransaction() method
- **Features the use of Prepared Statements**
 - Prepared Statements are emulated for databases that do not support them



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 239

Slide 240

PDO in Detail (continued)

- Large object support in `bindParam()` and `bindColumn()` by passing file resources
- Support for *input* and/or *output* parameters in stored procedures



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 240

PDO Drivers

- **Supported databases:**
 - MySQL 3-5
 - PostgreSQL
 - Microsoft SQL Server/Sybase
 - IBM DB2
 - Oracle
 - Informix
 - Firebird
 - ODBC
 - SQLite



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 241

Slide 242

Classes Available in PDO

- **PDO**
 - Represents the connection
- **PDOStatement**
 - Represents a prepared statement and an associated result set
- **PDOException**
 - Thrown on a database error



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 242

How It *Used to be* Done (outdated)

```
$db = sqlite_open('dbname.db');  
$id = sqlite_escape_string($_GET['id']);  
$result = sqlite_query($db, 'SELECT * FROM users WHERE  
user_key = '.$id);  
while (($row = sqlite_fetch_assoc($result)) !== false) {  
}
```

- Fine for simple scenarios, but complexity can increase quite quickly



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 243

How It *is Now* Done (Latest way)

```
$db = new PDO('sqlite:dbname.db');
$statement = $db->prepare('SELECT * FROM users WHERE
user_key = ?');
if ($statement->execute(array($_GET['id']))) {
    while (($row = $statement->fetch()) !== false) {

    }
}
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 244

Prepared Statements

- **Queries are compiled once on the server level and then parameters are passed to the server instead of building the query each**
- **Benefits of using prepared statements:**
 - Higher performance
 - SQL queries are compiled once on the server and re-used
 - Higher Security
 - Generally not vulnerable to SQL injection attacks
 - Still potential vulnerabilities but significantly lower probability
 - Data is passed using placeholders in the query instead of directly in the query
 - More portable
 - Different databases have different methods of escaping data



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 245

Prepared Statements – Usage Abstract

- **Prepared statements are dealt with in a specific order**
 1. Compile (prepare) the statement
 - The compilation step is the only addition to the steps needed to access a database
 2. Execute the query with or without parameters
 3. Retrieve the results
 - Previously, results were retrieved from a resource; now they are retrieved from an object



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 246

Prepared Statements – Actual Usage

1. Open a connection

```
$db = new PDO('sqlite:./mydb.db');
```

2. Compile (prepare) the statement

```
$stmt = $db->prepare('SELECT * FROM users WHERE user_id = ?')
```

3. Execute the query, filling in optional parameters

```
$stmt->execute(array($id)); // Returns true or false
```

4. Retrieve the results

```
$row = $stmt->fetch();
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 247

Retrieving Data from a Statement

- **\$row = \$pdo->fetch();**
 - `fetch()` can have multiple options (`$pdo->fetch(PDO::FETCH_LAZY)`)
 - `PDO::FETCH_ASSOC:` Returns an associative array
 - `PDO::FETCH_BOTH:` Returns associative and numerical array
 - `PDO::FETCH_BOUND:` Returns true and binds the values in your `bindColumn()` statement to the return values
 - `PDO::FETCH_CLASS:` Returns a new instance of the specified class with properties populated that map to the result set
 - `PDO::FETCH_INTO:` Takes an object and populates the properties with values mapped from column names
 - `PDO::FETCH_LAZY:` Combines `FETCH_BOTH` and `FETCH_OBJ`
 - `PDO::FETCH_NUM:` Returns a numerical array for the result set
 - `PDO::FETCH_OBJ:` Returns an anonymous object (`stdClass`) with property names corresponding to column names

Stored Procedures

- Precompiled sets of database queries stored on the server
- Stored procedures can be much faster than regular database queries
- Can put logic in the database, closer to the data



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 249

Stored Procedures – exec()

- Executing procedures without returned data

```
CREATE PROCEDURE 'proc' ()  
BEGIN  
    INSERT INTO somewhere VALUES (NOW());  
END  
  
$pdo->exec('CALL proc()');
```

- Note that when calling a stored procedure like this it cannot return data



Stored Procedures (continued)

- **Retrieving result sets from a stored procedure**

```
CREATE PROCEDURE 'proc' ()
BEGIN
    SELECT * FROM users
END

foreach ($pdo->query('CALL proc()') as
$row) {

}
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 251

Stored Procedures (continued)

- Handling input/output parameters

```
CREATE PROCEDURE proc (in gt int, out cnt int)
BEGIN
    SELECT COUNT(*) INTO cnt FROM users WHERE id > w
END

$gt = 500;
$stmt = $pdo->prepare('CALL proc(?, ?)');
$stmt->bindParam(1, $gt);
$stmt->bindParam(2, $count);
$stmt->execute();
echo $count; // The value from out "cnt int"
```



Stored Procedures (continued)

- Note: Output parameter binding may not work on some platforms (example: MySQL) due to MySQL API constraints

```
$gt = 500;  
$stmt = $pdo->prepare('CALL proc(?, @var)');  
$stmt->bindParam(1, $gt);  
$stmt->execute();  
  
echo $pdo->query('SELECT @var')->fetchColumn();  
// The value from out "cnt int"
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 253

Transactions

- Many databases have the ability to do ACID-compliant transactions
- ACID defined
 - Atomicity
 - Each transaction, or set of queries wrapped in a transaction, must either succeed or all fail (Commit vs. Rollback)
 - Consistency
 - If any data is invalid the entire transaction is rolled back
 - Isolation
 - Changes are not available in the main database until the entire transaction has been committed, though changes are available in the transaction
 - Durability
 - Any transactions committed to the database will not be lost
 - Ensured through the use of backups and transaction logs



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 254

Transactions (continued)

- Given `$pdo = new PDO()`
 - Transactions are started using the `beginTransaction()` method
 - `$pdo->beginTransaction();`
 - Transactions are ended via `commit()`, `rollback()` or doing nothing
 - `$pdo->commit()`
 - All changes are committed to the database as they appears in the transaction
 - `$pdo->rollback()`
 - All changes are discarded
 - `$pdo = null`
 - Database driver automatically rolls back any un-committed changes



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 255

PDO Tricks

- Easy, configurable persistence

```
$params = array(  
    PDO::ATTR_PERSISTENCE => true  
) ;  
  
$db = new PDO('mysql:...', $user, $password,  
$params)
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 256

PDO Tricks (continued)

- **Quick information retrieval with no parameters**

```
foreach ($db->query('SELECT * FROM users') as  
$row) {  
    // Do Stuff  
}
```

- **There are several optional parameters for this method:**

```
$pdo-> query ($statement )  
$pdo-> query ($statement, PDO::FETCH_COLUMN, $colno )  
$pdo-> ($statement, PDO::FETCH_CLASS, $classname, $ctorargs )  
$pdo-> query ($statement, PDO::FETCH_INTO, $object )
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 257

Slide 258

Exercise: Database Connections

- **The Z Blog application is currently coded to connect to a SQLite database**
- **Convert the application to connect to a MySQL DB instead**
- **Answer: Change one line of code**
 - One of the many benefits of PDO
 - In lib/ZBlog/Data.php change



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

Where to Learn More

- See the following resources to learn more about databases
 - <http://devzone.zend.com/article/641-PHP-101-part-8-Databases-and-Other-Animals---Part-1>
 - http://www.onlamp.com/pub/a/php/2003/12/23/php_foundations.html
 - [MySQL Crash Course by a Zend employee](#)



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 259

Slide 260

Module Summary

- Q&A
- **The next class is devoted to working on a comprehensive class project**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 260

Slide 261



IX: Building Applications - Critical Aspects

Slide 262

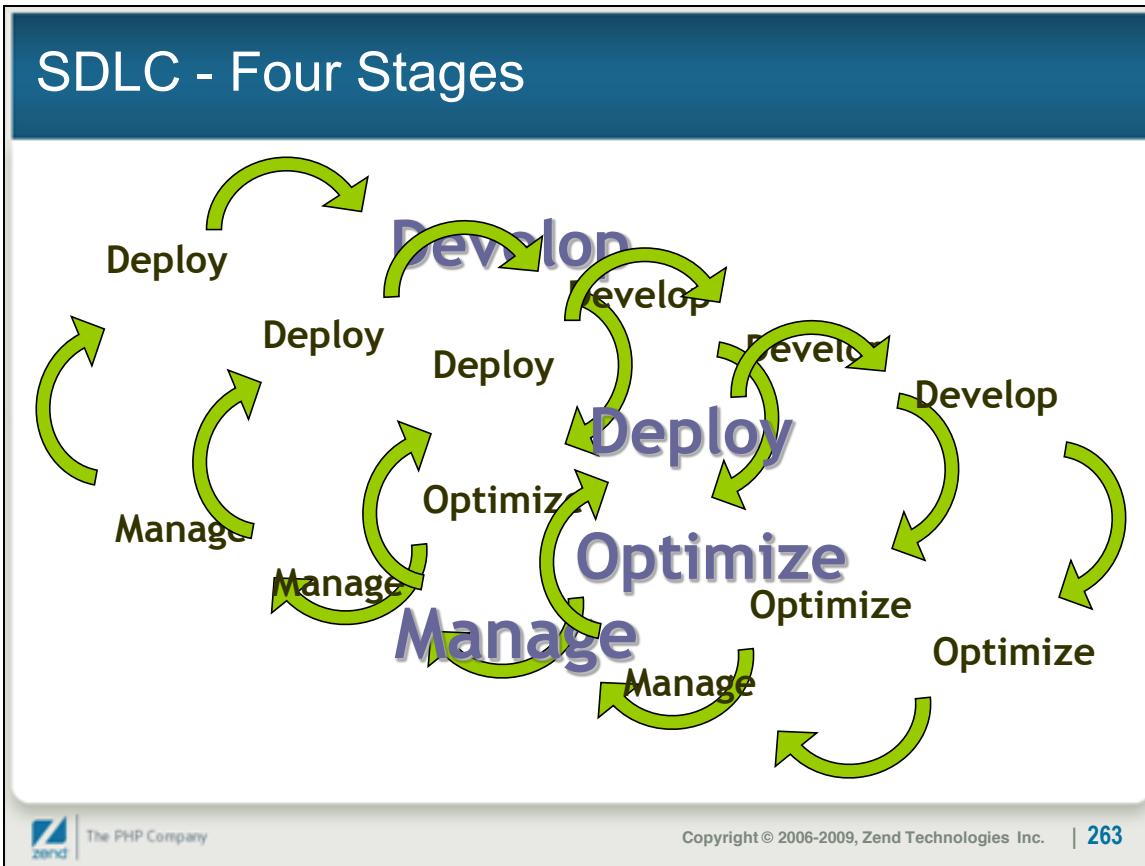
Building Applications Introduction

- **In this class we will cover:**
 - Software Development Life Cycle
 - Debugging Techniques
 - Security Considerations
 - Validation
 - Web Services
 - Performance enhancements



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 262



Slide 264

Testing and Debugging Techniques

- **Testing is an essential part of the Software Development Life Cycle, but is often not given enough time and focus**
 - Should involve testing beyond the developer – code buddy, simulated user
 - Best practices would include an alpha and beta testing stage
- **Within Debugging, there are a range of programming errors to look for:**
 - Syntax
 - Run-time
 - Logic

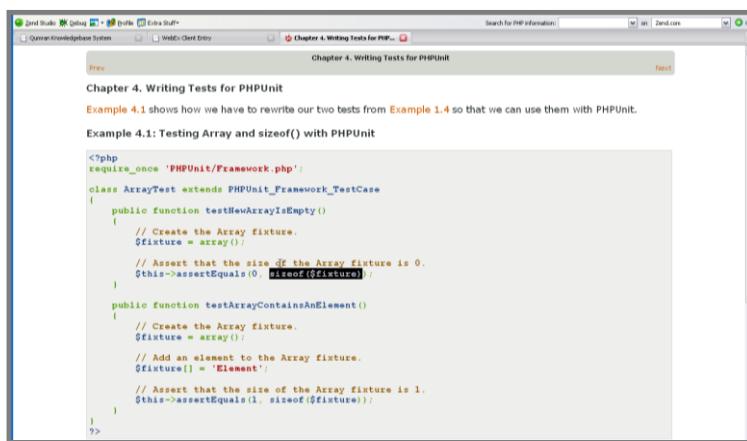


The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 264

Unit Testing

- **Unit Testing as a Best Practice**
 - Writing tests
 - Negative results indicate changes in application logic that may affect how your application runs



The screenshot shows a web browser window displaying a Zend Knowledgebase article. The title of the article is "Chapter 4. Writing Tests for PHPUnit". The content of the article includes a section titled "Example 4.1: Testing Array and sizeof() with PHPUnit" which contains the following PHP code:

```
<?php
require_once 'PHPUnit/Framework.php';
class ArrayTest extends PHPUnit_Framework_TestCase
{
    public function testNewArrayIsEmpty()
    {
        // Create the Array fixture.
        $fixture = array();
        // Assert that the size of the Array fixture is 0.
        $this->assertEquals(0, sizeof($fixture));
    }

    public function testArrayListContainsAnElement()
    {
        // Create the Array fixture.
        $fixture = array();
        // Add an element to the Array fixture.
        $fixture[0] = 'Element';
        // Assert that the size of the Array fixture is 1.
        $this->assertEquals(1, sizeof($fixture));
    }
}
?>
```

Debugging Tips Review

- When you debug, the compiler stops where it thinks the error is but the actual error may happen earlier (a line or two above)
 - An easy example is if you forget the ";" at the end of a line - look at prior line of code
- Check that you have spelled function, variable, or argument names correctly
 - You might declare a new item rather than use the existing one you intended
- Check that variables have a \$ in front of their names



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 266

Debugging Tips Review

- **Error message saying 'file/class cannot be found' - look for `get_include_path`'**
- **As you work on each part of your code ask:**
 - Did you get the results you expected?
 - Did you account for all things a user might do?
 - You might need to add a conditional statement to handle all conditions

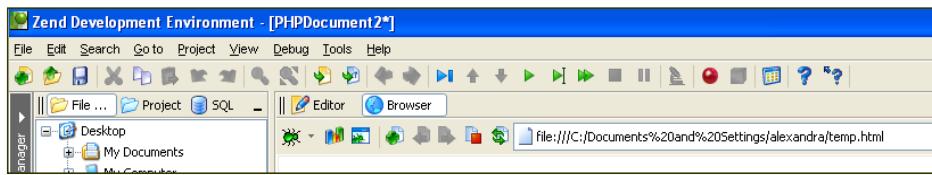


The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 267

Debugging Demonstration

- Now we'll look at some of the debugging features in Zend Studio, as an example of what capabilities an IDE can provide, including:
 - Set Breakpoints
 - Step into
 - Step over



Debugging Explanation

- **Debugging Ajax/RPC requests**
 - Both are done the same by way of adding addition query parameters to the remote request
 - It is not as easy as direct HTML but it can be done (needs to be worded better)
 - Append a query string to any Ajax or RPC request with the following key/value pairs
 - start_debug=1
 - debug_port=10100
 - Change this to the port that your Zend Studio is listening on
 - debug_fastfile=1
 - debug_host=192.168.0.91
 - Change to the IP address of your machine

Debugging Explanation

- **Debugging Ajax/RPC requests (continued)**
 - `debug_no_cache=1196256875125`
 - Pseudo randomize the value
 - `debug_stop=1`
 - `debug_url=1`
 - `debug_new_session=1`



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 270

Slide 271

Troubleshooting Procedures: Development

- **Common Troubleshooting Procedures - Development**
 - Setting `display_errors` to `On`
 - Causes errors or notices to be displayed on a Web page
 - Setting `E_Noteice` (`uninitialized variables`)
 - `E_Strict` forces you to use PHP 5 standards captured in log files with error reporting level set to `E_All`
 - Proper use of `var_dump`



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 271

Slide 272

Troubleshooting Procedures: Error Messages

- Compiler error messages
- Run-time error messages
- Log Files
- File Locations



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 272

Troubleshooting Procedures: Development

- Include paths: place files into directories based on some criteria (e.g., functional grouping, as in remote requests)
- PHP will look for `include_path()` in the `php.ini` file, or for the on-the-fly `set_include_path()`
- Can also access the `include_path` by using `get_include_path()` (gets directory that includes file)
 - Make sure to properly set the `include_path()` to include all required 3rd party libraries
 - Normally, refrain from using `set_include_path()`, which is a PHP function that will require changing your code (creates differences between the development code and production code)
 - Try to always properly set the `include_path = '...'` directive in `php.ini`, which defines global settings for the PHP environment without changing the code



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 273

Slide 274

Troubleshooting Procedures: Development

- **During Deployment:**

- `display_errors` always set to Off but `E_NOTICE` is set to On in the `error_reporting` `php.ini` setting
- Need to check for overall good code, as well as security (e.g., the use of `register_globals`)



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 274

phpDocumentor and PHPDoc

- **phpDocumentor is the current standard auto-documentation tool for the PHP language; open source**
- **PHPDoc is the protocol**
 - Similar to Javadoc
 - Can be used to create professional documentation from PHP source code
- **To use it, you need to comment your code carefully**
- **<http://www.phpdoc.org/> for more information**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 275

phpDoc

- Helps provide automatic documentation for your source code in a similar fashion as Javadoc
- Since function return types are dynamic phpDoc allows the developer to specify the return type for the IDE
- Denoted using the Javadoc syntax and is understood by the Zend Engine (T_DOC_COMMENT)

```
/**
```

```
 */
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 276

phpDoc Tags

- **Much of the benefit of phpDoc comes from the multitude of tags that you can embed in your documentation**
- **Some of note are:**
 - @author denotes the author of any item in a page
 - @category used with the @package tag to organize the element within a given package
 - @deprecated notes that a given element may be removed in a future version
 - @link links to a given element or URL
 - @package denotes a group that classes may belong to



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 277

phpDoc Tags (continued)

- `@param` documents the type and description of a function/method parameter
- `@property` a `__magic()` property that is not explicitly defined
- `@return` denotes the return type and description of a function/method
- `@todo` document changes that need to be done



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 278

Example – phpDoc

An example of a phpDoc block for a delete method in a Page class

```
/**  
 * Deletes the page associated with this object. Checks for  
 * super user permissions ahead. Will delete the page and all  
 * page content for all languages for this page.  
 *  
 * @param $handleTransaction Sets whether or not to handle  
 * the transaction  
 * @return boolean True or false depending on if the delete  
 * was successful  
 * @throws Page_Exception if page deletion not successful  
 */
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 279

Slide 280

Exercise: phpDoc

- Enter in phpDoc comments for the `model/comments.php` file and run the phpDocumenter from the Tools menu in Zend Studio



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 280

Error Handling Functions

- **Use standard error checking functions or define your own**
 - `die()`
 - Use this function to check for something and if it doesn't exist, stop the script running
 - `custom_errorHandler($error_code, $error_msg)`
 - Define your own, but there is a standard to use—there are other optional arguments available, too



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 281

- <http://www.php.net/manual/en/ref.errorfunc.php> error handling and reporting functions
- <http://us2.php.net/manual/en/security.errors.php> also discusses error reporting

Exceptions

- PHP 5 has an exception model similar to that of other programming languages
- An exception can be *thrown*, and caught ("*caught*") within PHP
- Code may be surrounded in a *try* block, to facilitate the catching of potential exceptions
- Each *try* must have at least one corresponding *catch* block
- Multiple *catch* blocks can be used to catch different classes of exceptions
- Normal execution (when no exception is thrown within the *try* block, or when a *catch* matching the thrown exception's class is not present) will continue after that last *catch* block defined in sequence



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 282

Exceptions (continued)

- Exceptions can be *thrown* (or re-thrown) within a *catch* block
- When an exception is thrown, code following the statement will not be executed, and PHP will attempt to find the first matching *catch* block
- If an exception is not caught, a PHP Fatal Error will be issued with an "*Uncaught Exception ...*" message, unless a handler has been defined with `set_exception_handler()`



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 283

Exception Example

```
<?php
try
{
    $connection = mysql_connect();
    if ($connection === false)
    {
        throw new Exception('Cannot connect to mysql');
    }

    /* ... do whatever you need with database; may throw      exceptions,
       too ... */

    mysql_close($connection);
}
catch (Exception $e)
{
    /* ... add logging stuff here if you need to ... */

    echo "This page cannot be displayed";
}
?>
```



Applying Error Checking

- **Be careful when checking return values of functions**
 - Typically a PHP function will return false
 - Because zero can be interpreted as a false, always use strict type checking
 - `if (preg_match_all('.../', $str, $matches) {`
 - `preg_match_all()` returns 0 or more results or false on error
- **When building PHP 5 applications Exceptions are much better at detecting and handling error conditions**
 - Errors are not handled by response code but by try/catch blocks
 - Separates return logic and error handling logic



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 285

Web Services

- **Web services are application interfaces that operate via the World Wide Web and provide a standardized way of allowing applications to interface and share data across a network**
- **Web service messages are written in XML, allowing for different applications in different programming languages to interface with each other**
- **Two most important protocols for implementing Web Services are Soap and WSDL**
 - A Web Service and a client communicate using Soap messages, which encapsulate the 'in' and 'out' messages as XML
 - The proxy class handles mapping parameters to XML elements and then sends them via a Soap message over the network

Web Services: Proxy Class

- **The generation of a proxy class depends upon the existence of a service description that conforms to the WSDL prescripts, which defines how to communicate with the Web Service**
- **Once the proxy class is created, the Web Service client can call methods from the proxy class, which processes the Soap messages to communicate**
- **The primary reason to use Web Services is the ability to build a loosely-coupled architecture**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 287

- Why use a Proxy Class?
- Loosely-coupled in the sense that the application is not required to know much about the components it communicates with
- A Web Service and a client may communicate using Soap (most common) messages, which encapsulate the 'in' and 'out' messages as XML
- The proxy class handles mapping parameters to XML elements and then sends them via a Soap message over the network

Web Services: XML-RPC

- The XML-RPC protocol is a predecessor of Soap
 - While Soap is an integral part of most Web Services implementations, in some instances it can be “overkill” - makes sense to use the lighter version provided by XML-RPC
 - Applications that contain relatively simple requests, or situations when both the client and server sides are controlled together, are examples of when it would be appropriate to use this ‘lighter’ protocol
 - The XML-RPC protocol uses only a few data types and does not require namespaces
 - It allows array types to contain other values, providing flexibility
- * Extensible Markup Language-Remote Procedure Call



Here is a list of XML-RPC data types:

- Array: An array of values with no keys
- Base64: A base64 encoded piece of data, usually binary data
- Boolean: Boolean value
- Date/: ISO rendered date
- Double: Double precision floating point number
- Integer: Whole number
- String: String of characters
- Struct: Associative array
- Nil: null

Web Services: Soap Client

- **Soap, a messaging protocol, is a simple XML document sent over HTTP, that facilitates a request and response interaction**
- **Composed of: The Declaration; The Body; The Envelope**
- **When the Web service client makes a request, the Soap client API builds a corresponding XML message containing the remote method name and value for its parameters, then sends the XML message over HTTP protocol to the server hosting the Web services**
- **The server receives the XML message, executes the business logic (may be written in another language), and sends the response back to the client**

* Simple Object Access Protocol (formerly)

Slide 290

Web Services: WSDL

- A WSDL is an XML-derived language for describing the methods available for a given web service
- WSDL service definitions provide documentation for distributed systems and serve as a framework for automating the details around site communication
- A WSDL document defines services as collections of network endpoints, or ports - this allows the reuse of abstract definitions
- The server receives the XML message, executes the business logic (may be written in another language), and sends the response back to the client

* Web Services Description Language



Copyright © 2006-2009, Zend Technologies Inc. | 290

Create a Web Service

- **Class with business logic**

```
class WSMETHODS {  
    function myMethod($param) {  
        return $param;  
    }  
}
```

- **Register with server**

```
$soap = new SoapServer('file.wsdl');  
$soap->setClass('WSMETHODS');  
$soap->handle();
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 291

Slide 292

Consume a Web Service

- **Instantiate (using WSDL)**

```
$soap = new SoapClient('file.wsdl');
```

- **Then call methods**

```
$soap->myMethod('Hello!'); or  
$soap->__soapCall('myMethod', array('Hello!'));
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 292

Debugging a Web Service

| | |
|--|--|
| <code>_getLastRequest()</code> | returns the body of the last web service HTTP request |
| <code>_getLastRequestHeaders()</code> | returns headers of the last web service HTTP request |
| <code>_getLastResponse()</code> | returns the body of the last web service HTTP response |
| <code>_getLastResponseHeaders()</code> | returns headers of the last web service HTTP response |

- **Must construct SoapClient object as**

```
$soap = new SoapClient('wizdle.wsdl', array('trace' => true));
```



Error Handling - Web Service

- **Using exceptions**

```
try {
    $soap = new SoapClient('file.wsdl');
    $res = $soap->myOtherMethod('xyz');
} catch (SoapFault $ex) {
}
```

- **Using a function**

```
if (is_soap_fault($res)) { ... }
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 294

Example: WSDL Generator

- **Using Studio's internal WSDL generator, let's take a look at how to build a WSDL from the blogs.php file**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 295

Exercise: Web Services

- **Create a new controller that handles Soap requests for listing all of the items in the blog.**
- **You will need to create:**
 - A new controller for handling the request
 - A library item that can be exposed using Soap that returns an array of all the items in the Blog
 - A Soap client in a separate file for making the Soap requests, which uses `var_dump` to display the results



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 296

Solution: Web Services

```
application/model/soap.php
WSDL should be created from this file

<?php
require_once('model/blogs.php');
class SoapArticle {
    public function getArticles()
    {
        $return = array();
        foreach (ZBlog_Model_Blogs::getEntries() as $entry) {
            $ent = array();
            foreach ($entry as $key => $val) {
                $ent[$key] = utf8_encode($val);
            }
            $return[] = $ent;
        }
        return $return;
    }
}
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 297

Instructor Slide 298

Solution: Web Services

```
application/controllers/soapController.php
<?php
require_once('model/soap.php');
class soapController {
    public function indexAction()
    {
        $soapServer = new SoapServer('../html/Articles.wsdl');
        $soapServer->setClass('SoapArticle');
        $soapServer->handle();
    }
}
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 298

Solution: Web Services

```
html/soap.php
```

```
This is the soap client program
```

```
$soapClient = new SoapClient('http://<IP ADDR>/Articles.wsdl');  
var_dump($soapClient->getArticles());
```



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 299

Slide 300

Performance Enhancements

- **Caching**
 - Dynamic Content Caching
 - Full Page Caching
 - Code Acceleration/Opcode caching **
- **File Compression ****

**** Generally, under control of System Administrators, but developers should be aware of process and benefits**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 300

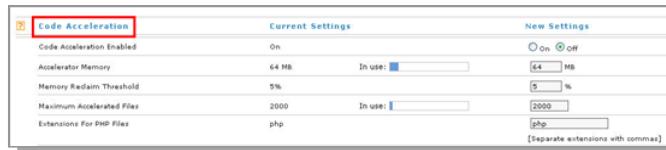
Dynamic Content Caching (Partial Page)

- **Partial Page (Dynamic) Caching**
 - Dynamic Content caching is the process of running code once and saving the output on the server for reuse
 - Performance is improved by using the cached output instead of generating the same output each
 - Files should be cached only when their content is relatively stable and do not require constant change
 - Caching needs to be defined specifically for each file or directory – no files are cached by default

- Web pages that contain sections that continuously change can also be cached
- Partial Page Caching is accomplished by applying caching APIs to portions of code that do not change
 - Changes in the application code are required
 - The more effort put into smart usage of the API, the bigger is the performance gain
 - Both data items (variables, SQL result sets) and output can be cached and reused

Code Acceleration (Op Code Caching)

- **Code Acceleration (Op Code Caching) is the process of gaining a performance boost by eliminating the code compilation**
- **Once PHP code is compiled for the first time, it is saved in the server's memory**
- **Each time the code is called, the pre-compiled version is used instead of incurring a compilation lag each time the code is used**

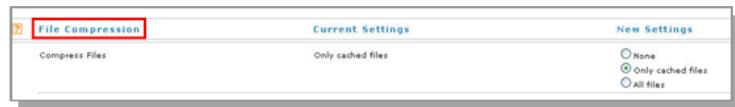


Example: Zend Platform

- Normally, when a PHP file is requested, the sequence of procedures are: read - parse - compile - execute. Code Acceleration works by saving the PHP code, once it is called for the first , into the server's memory (RAM; configurable). Then, each subsequent time the code is called, the saved, pre-compiled version is used instead of the original script
- Note: Acceleration is not the same thing as Caching
 - Acceleration saves the *actual compiled script* in the server's memory, while Caching saves the *output* of that script.
 - From a development view, it is recommended that this feature be set to Off during development, then to On for deployment, with the developer selectively excluding ("blacklist", user_blacklist_filename) those files that should not be cached. Common reasons *not* to cache a page are: files that have high memory consumption or long execution

Slide 303

File (Code) Compression



Example: Zend Platform

- **Code compression is the process of using less bandwidth and increasing performance by compressing code before it is sent**
- **The majority of browsers can receive compressed data (gzip format)**
- **If using full page caching, your application could benefit tremendously, as most files can be compressed up to 90%**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 303

Profiling

- Profiling summarizes the data that makes up the PHP application, such as
 - Function Statistics: list of files constructing the URL and detailed information on functions in the files
 - Call Trace: provides a hierarchical display of functions according to process order, enabling you to jump to the function, and then view the function call, declaration, details and more
- By placing timers within the code and running them over and over, the profiling tool is able to build a "profile" of how fast or slow specific areas of the application will run
- Designed to help discover bottlenecks and other areas that need to be optimized to improve the program's performance
 - An extensive library of profiling benchmarks is included within Zend Studio



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 304

What is Security?

- **Broadly defined, security is the set of measures and procedures, taken to prevent unauthorized access to hardware or software**
- **Steps need to be taken at all stages of the software development life cycle (SDLC) to avoid creating vulnerabilities**
- **When web application security is built into an application –**
 - in the code itself
 - in the regulation of data input/output
 - in proper error handling and logging

it reduces the threat that attackers will be able to manipulate applications and access, steal, modify, or delete sensitive data



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 305

Slide 306

Defense In Depth Concept

- It is critical to realize that no one step will ensure security. In fact, no combination can *ensure* it either
- There is a best strategy – defense in depth – which means employing a broad range of overlapping security tactics to present a defense to unwanted attacks
- The concept is to make it so difficult for an attacker to break through all the security measures in place that they are likely to give up and attack a site that is easier to assault



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 306

Basic Security Rules

- **Never trust data from the outside... NEVER**

Remember that outside data includes DB content...

Anything that employs user input, if the data was not sanitized, is suspicious

- GET/POST data
- Cookies
- Environment Variables
- Server Variables
- HTTP headers

What is your user name?

Set to alphanumeric, <=8 chars

- **Steps You Must Code Into Your Application**

- Filter/validate input
- Escape output



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | **307**

Validation

- You've already learned that you can validate on the client side or the server side and how to use regular expressions to validate email addresses
- There are several different methods you can use to do validation
 - Regex is an expensive operation in terms of performance
 - Functions that can do validation more efficiently
 - Casting - \$valid = \$id == (int)\$_GET['id']
 - ctype_* functions
 - str* functions
 - strpos
 - stripos



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 308

Validation Rules

- **What do you remember about form validation?**
- **Don't make assumptions - check that the results you get are what you are asking for**
 - Is it a valid email address?
 - Is it a phone number or zip code?
- **Specific areas to check (vulnerable to security breaches)**
 - SQL injections, etc.
 - XSS <script...
 - XSRF
 - Checking HTML input



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 309

Secure Web Applications Guidelines

Again, the key to security is employing a range of security measures to reduce the risk. Here are some best practices to take throughout the SDLC when creating web applications:

**Principle: Never assume that user input received follows the rules and
ALWAYS re-create (initialize) or verify it**

- Set `register_globals` to Off
 - If you have to use this setting, take great caution and make sure you have security steps in place
- Initialize all variables
 - Not required by PHP, but instills security into your application
 - Set `error_reporting` to `E_ALL` as a good way to remember this step

Principle of Least Privileges

- Grant permissions to users only to the level needed



Copyright © 2006-2009, Zend Technologies Inc. | 310

- **register_globals:** “poster boy” for making something so simple it becomes dangerous
 - At one, was defaulted as ‘ON’, but not anymore

“What do you do if you have to have this on?” some companies have this built in, depending on when applications were built
 - Not dead in the water, but have to be especially careful
 - **Every variable HAS TO BE INITIALIZED!**
 - Use tools; Zend Studio, e.g., has Code Analyzer to identify when you are using variables before they have been initialized
 - could use `isset()` to ensure that all variables are defined, especially useful if dealing with data of uncertain quality

Least Privileges:

older versions of OS, like Windows, used to require too many people to be an Admin... not any more

- Limit permissions to minimum necessary - running as an Admin supersedes all other security measures, so any breach now extends over entire system, not just individual user
 - e.g., no sending world-writable ‘777’ file directories
- Take the time to make sure groups, directories, and related permissions are established correctly

Secure Web Applications Guidelines (continued)

- **Cookies:**

- Only pass an identifier in a cookie, not actual data
 - Any data inside of a cookie can potentially be seen by others – restrict to a minimum
- Check the host returning the cookie is the same as to whom it was sent
 - will not work for group of computers behind single IP address (e.g., NAT)

- **Errors: covered earlier...**



Cookies: (can be set only for the domain and sub-domains the user is on... Parent to child only direction...)

- Any information sent out should be considered known to the user and potentially to other agents - so it should be restricted to minimum and verified on input, not trusted (e.g., cookie saying "logged in" is bad idea);
- Cookies and Check Host: will certainly not work for groups of computers hidden behind a single IP address (i.e. NAT). Since most corporate networks access this internet this way, this form of check is of limited value

Error Logging: Probably most under-rated and under-utilized strategy step for security

- In order for malicious users to take advantage of your programs, they must first probe your application to find possible attack vectors, so they often probe web applications to gather as much information as possible about them, esp. by trying to cause errors.

Secure Web Applications Guidelines (continued)

You are releasing a new Guest Book software program as an open source application, so you had to place all the files into an `htdocs` directory, for ease of installation and use by a large community. Within the program, you have an `index.php` and `user.inc` file. How could you make this application more secure?

Make sure that the extensions on all the documents, whether they are `/includes` or actual php scripts, have the same extension.

So, the user file would be renamed as: `user.inc.php`

If you keep all the extensions the same, then Apache cannot accidentally mishandle the file, and the attacker cannot download the file, even if they know the name of the `user.inc.php` file



Security Tips Review

- **Don't trust outside data**
- **Check to see if the results you get are what you expect**
 - Use Debugging techniques instead of `echo` or `print` statements during development
- **Here are some useful articles on writing secure code**
 - <http://devzone.zend.com/node/view/id/661>
 - http://www.onlamp.com/pub/a/php/2003/07/31/php_foundations.html
(part 1 of a 3-part article on writing secure code by a Zend employee)
- **Zend offers a class about writing secure code**
 - http://www.zend.com/education/php_training_courses/courses



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 313

Slide 314

Module Summary

- Q&A



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 314

Slide 315



X: PHP Application Summary Project

Slide 316

Application Introduction

- **In this class we will cover:**
 - Develop a final project
 - Review of entire course
 - Where to go from here



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 316

Slide 317

Review from Last Class

- **Review**
 - Building Application concepts



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 317

Slide 318

Z Blog Project: Create Search Controller

- **Code the Search controller in the Z Blog project application**

In creating this controller, you will be using/reviewing the following:

- Database
- Quotes
- Loops
- IF
- Comments
- Variables, Constants
- Arrays
- Internal Functions



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 318

Class Review

- **In this class we have covered:**
 - PHP Syntax
 - PHP Language Concepts
 - Configuring PHP
 - Regular Expressions
 - Object-Oriented Programming
 - Web concepts
 - Database concepts
 - Building PHP Application considerations



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 319

Class Review: Basic Syntax

- **In this class we have covered:**
 - Syntax elements
 - Data types: integer, float, string, Boolean
 - Operators
 - Assignment, math, string manipulation, logical, incrementing/decrementing
 - Order of precedence of operators
 - Variables, constants
 - Variables start with \$ and contain letters, numbers and underscores, typically start with lower case
 - Constants contain letters, numbers, and underscores and typically use capital letters



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 320

Slide 321

Class Review: PHP Language Concepts

- **In this class we have covered:**

- Including and requiring files
- Scope
- Globals
- References
- `List()`
- Working with files



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 321

Slide 322

Class Review: Configuring PHP

- **In this class we covered**
 - Setting up a PHP environment
 - More about `php.ini`



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 322

Class Review: Regular Expressions

- **In this class we discussed**
 - What regular expressions are
 - Common syntax for creating regular expressions
 - Functions used with regular expressions
 - How to create a regular expression to check that a valid email address has been entered



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 323

Slide 324

Class Review: Web Concepts

- **In this class we have covered:**
 - Differences between local applications and web-based
 - How servers communicate (an introduction)
 - GET
 - POST
 - HTTP Headers
 - Cookies and Sessions
 - Buffers, Email, and Browser Caching
 - Forms and Form Validation



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 324

Class Review: Object-Oriented Programming

- **In this class we covered:**
 - Object-Oriented Programming
 - Classes
 - Inheritance
 - Polymorphism
 - Abstract Classes
 - Interfaces
 - Type Hinting
 - Overriding Functions



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 325

Slide 326

Class Review: Database Concepts

- **In this class we have covered:**
 - Introduction to relational databases
 - SQL and MySQL
 - Connecting PHP and MySQL



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 326

Class Review: Building Applications

- **In this class we have covered:**
 - Software Development Life Cycle
 - Debugging Techniques
 - Security Considerations
 - Validation
 - Web Services
 - Performance enhancements



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 327

Slide 328

Where to Go From Here

- **Other courses from Zend:**
 - Building Security into your PHP Application
 - Zend Studio with Platform for Developers (must be experienced PHP developer and familiar with OOP)
- **To learn more see Additional Resources slide for more links**
 - <http://devzone.zend.com> and <http://zend.com>
- **Professional Services**
 - Zend Support Services
 - Zend consulting: packaged services, custom consulting
 - Zend certification



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 328

Additional Resources

- **PHP Development:** <http://devzone.zend.com>; <http://www.zend.com>
- **Free HTML cheat sheet:** <http://www.psacake.com/web/dy.asp>
- **Free online class to strengthen your PHP:**
<http://www.w3schools.com/php/default.asp>
- **Useful KnowledgeBase articles:**
<http://www.zend.com/support/knowledgebase.php>
- **PHP Collaboration Project** <http://framework.zend.com>
- **Open Source Applications:**
<http://sourceforge.net>
<http://freshmeat.net>
<http://www.hotscripts.com>
<http://www.phpjunkyard.com>
<http://www.phpfreaks.com>



Copyright © 2006-2009, Zend Technologies Inc. | 329

Tip: Google for “php cheat sheet” to find a bunch more

Slide 330

Before We End

- **Q&A**
- **Final class poll**



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc. | 330

Slide 331

Acceptance of Conditions for Use of Code

ZEND grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

THE ZEND COURSES' MATERIALS ARE PROVIDED "AS IS" AND SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, ZEND, ITS OFFICERS, DIRECTORS, EMPLOYEES, PROGRAM DEVELOPERS AND TRAINING PARTNERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE COURSES, ZEND COURSES' MATERIALS OR PROGRAMS PROVIDED, IF ANY.

UNDER NO CIRCUMSTANCES IS ZEND, ITS OFFICERS, DIRECTORS, EMPLOYEES, PROGRAM DEVELOPERS OR TRAINING PARTNERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

LOSS OF, OR DAMAGE TO, DATA;

DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

ACCURACY OR COMPLETENESS OF THE ZEND COURSES' MATERIALS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.



The PHP Company

Copyright © 2006-2009, Zend Technologies Inc.

| 331

Slide 332



THANK YOU FOR PARTICIPATING
IN OUR COURSE!