PHP Crash Course

PHP

- PHP is a server scripting language built for making dynamic and interactive web pages.

- PHP is a widely-used, open-source, and free, making it a popular alternative to other scripting languages, such as ASP.NET

- PHP 7 is the latest major stable release.

PHP Crash Course

- Links to official documentation on certain functions mentioned in this presentation have been omitted.
- The official PHP website is the definitive, most up-to-date, and version-specific, source on PHP. https://www.php.net/manual/en/
- Additional documentation, examples, and tutorials can be found on W3Schools.com https://www.w3schools.com/php/default.asp

PHP Complementary Technologies

To learn PHP effectively, you should have a basic understanding of

- HTML
- CSS
- JavaScript
- Client / Server Model
- SQL

These skills are necessary for a full stack developer, who work on tasks from the front-end design to the back-end system architecture.

Brief HTML 5 Review

- **Hypertext Markup Language - HTML**

- Simple HTML5 DOCTYPE declaration:

```
<!DOCTYPE html>
```

- Simple character encoding (charset) declaration:

```
<meta charset='UTF-8' >
```

HTML5 Recent Features and APIs

- Increased accessibility for screen readers and other devices
- Multimedia support
- Geolocation
- Application cache
- Web workers
- Simplified syntax
- Mobile support
- Local storage is an improvement over cookies
- Improvements on the user interface tools

HTML 5 Markup Example

```
<!DOCTYPE html>
<html>
  <head>
  <meta charset='UTF-8' >
  <title> Document Title</title>
  </head>
  <body>
  Hello World!
  </body>
</html>
```

HTML Formatting

- Every container tag must have a corresponding closing tag, e.g., <p>…</p>
- Every in-line tag must be of the format <…>, e.g., <br>
- All elements must be properly nested
- All tag names and attribute names must be lower case
- All attributes must have a value
- All attribute values must be quoted (single or double allowed)

HTTP Review

- **Hypertext Transfer Protocol**

- A **client** makes a request to a **server** over a network for a resource
  - A web page (.html, .php files)
  - An image (.png, .jpg, .gif files)
  - A video (.mp4 files)

- Each resource has an associated **Uniform Resource Locator** or **URL**
  - http://www.google.com

URL

- A **URL** is used to request a remote resource on a server

- Syntax of a HTTP URL

```
scheme://host[:port]/path/…/[;url-params][?query-string][#anchor]
```

URL Breakdown

http://www.somesite.com:8080/somepath/path;id=1234?name=demo&type=url#marked

- **Scheme**: http://
- **Host**: www.somesite.com
- **Port**: 8080
- **Path**: /somepath/path
- **URL Params**: id=1234
- **Query String**: name=demo&type=url
- **Anchor**: marked

URL Scheme and Host

- **Scheme**
  - Specifies the underlying protocol
  - Examples include http, https, ftp, file, mailto, …
- **Host**
  - Identifies the server and domain on the Internet
  - Domain is made up of top-level **domain** (e.g., .com, .edu, .net, etc. or country code) and second-level domain, which is the name immediately to the left of the top-level domain
  - Can be either the name or the **Internet Protocol (IP) address**

URL Port

- **Port**
  - Optional URL element
  - Identifies the port at which the web server is listening
  - It follows a colon (:)
  - Optional value, web server default is 80 for HTTP but may be changed when configuring the server and 443 for HTTPS
  - Must be specified if an alternative port number is used

URL Path

- **Path**
  - Identifies how to locate the resource within the server’s file system
  - May refer to an actual file on the server
  - Includes the first forward slash after the host name and optional port

URL Params

- **Params**
  - Optional URL element
  - Used to deliver elements such as session identifiers
  - It follows a semi-colon (;)
  - Not common

URL Query String

- **Query String**
  - Optional URL element
  - Contains dynamic parameters associated with the resource request
  - It follows a question mark (?)
  - Name/value pairs are separated with the equal sign (=)
  - Multiple name/value pairs are delimited by the ampersand sign (&)
  - Used to deliver elements such as form data and state

URL Anchor

- **Anchor**
  - Optional URL element
  - Reference to a positional marker within the requested resource
  - It follows a hash or pound sign (#)

URL Encoding

- Some information that needs to be included in a URL either violates the syntax or goes beyond capabilities of a text field
- The following characters are safe to use
  - A to Z and a to z
  - 0 to 9
  - -_.+*’(),
- Most languages offer URL encoding functions
  - JavaScript: **encodeURI**()
  - PHP: **rawurlencode**()

URL Encoding

ASCII Character	**URL-encoding**
space	20%
!	21%
"	22%
#	23%
$	24%
%	25%
&	26%
<	%3C
=	%3D
>	%3E
@	40%

PHP Overview

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor"
- Web-specific server-side scripting language
- Widely used, open source, free
- Embedded within an HTML page using tags
- Executed when a client requests a page
- Hidden from the client, client only sees output
- Built to easily integrate with a database and other server apps
- Platform and O/S portable – PHP runs on a variety of different platforms and operating systems

PHP Functionality

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

PHP Files

- PHP files have a .php extension. Server needs this in order to know to run the file through the PHP script engine before sending output to client.
- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code is executed on the server, replaced with resulting output, and returned to the browser.
- A block of PHP script is embedded within an HTML file using the <?php and ?> tags. (Optionally can use <? and ?>)
- Just like Java, whitespace is ignored.
- Just like Java, end lines with semicolon (;).

PHP Hello World Example

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='UTF-8' />
    <title>Hello, World!</title>
  </head>
<body>
  <h1>
    <?php
      echo "Hello, World!";
    ?>
  </h1>
</body>
</html>
```

PHP Comments

- Block comments

```
/* This is a block comment
   that can continue

   for many lines.

*/
```

- Line comments

```
// C-style line comment
```

Echo statement

- The echo statement works like Java print without parentheses
- echo, however, can take on a sequence of arguments separated by commas (or a singular concatenated string, PHP is very flexible).
- Example:

```
$outputString = "The answer is ";
echo $outputString, 42, "!";
```

- This outputs "The answer is 42!"

Escape Characters

- Several reserved PHP characters require escaping inside strings
- Lack of care taken with special characters may create security issue
- Characters are escaped by preceding them with a backslash (\).
- Characters that need escaped include ', ", \, $, and ?.
- Whitespace including carriage returns are allowed as part of a string, but they are then output as part of the string.
- Single quotes can be used without escaping within double quoted strings and vice versa.

PHP Variables

- PHP interprets the dollar sign ($) followed by a letter or an underscore (_) to be a variable name.
- PHP variables do not need to be declared before you use them.
- Examples:

```
$var1 = 25;
$var2 = "Hello, World!";
$var3 = true;
```

Variable Interpolation

- A variable can be embedded directly in a string defined with double quotes
- Example: The following code sets $str to “Hello, Andrew!”

```
$name = "Andrew";
$str = "Hello, $name!";
```

- You can also use curly brackets to set off a variable identifier within a string defined with double quotes – this will become helpful when we start discussing arrays and objects
- These methods do not work for strings defined with single quotes

Variable Interpolation

```
$var1 = 25;
echo "Value is $var1.";   // Outputs "Value is 25."
echo "Value is {$var1}."; // Outputs "Value is 25."
echo 'Value is $var1.';   // Outputs "Value is $var1."
echo 'Value is ', $var1, '.’;  // Outputs "Value is 25."

$string1 = "Value is $var1.";
$string2 = 'Value is $var1.';
echo $string1;              // Outputs "Value is 25. ";
echo $string2;              // Outputs "Value is $var1."
```

PHP Data Types

- Scalar types
  - boolean
  - float
  - integer
  - string
- Compound types
  - array
  - object

PHP Scalar Types

- A **boolean** variable can be assigned only values of true or false.

```
$answer = false;
$finished = true;
```

- An **integer** is a whole number (no decimal point)

```
$age = 31;
```

PHP Scalar Types

- A float has a decimal point and may or may not have an exponent

```
$price = 12.34;
$avg_num = 6.02e23; //6.02x10^23
```

- A string is identified as a sequence of characters

```
$name = "John Smith";
```

- Strings can be concatenated using a dot (.)

```
$name = "John" . " Smith";
```

PHP Constants

- Constants associate a name with a scalar value.
- Constants are defined using the function define().

```
define("PI", 3.141593);
```

- There are several predefined constants. These include:
  - M_E = 2.718281828459
  - M_PI = 3.1415926535898
  - M_2_SQRTPI = 1.1283791670955 (Square root of pi)
  - M_1_PI = 0.31830988618379 (Square root of 1/pi)
  - M_SQRT2 = 1.4142135623731 (Square root of 2)
  - M_SQRT1_2 = 0.70710678118655 (Square root of ½)

PHP Arithmetic Operators

Operator	Operation	Example	Result
+	Addition	`$y = 2 + 2;`	$y will contain 4

–	Subtraction	`$y = 3;`
		`$y = $y – 1;`
			$y will contain 2

/	Division	`$y = 14 / 2;`	$y will contain 7

*	Multiplication	`$z = 4;`
		`$y = $z * 4;`
			$y will contain 16

%	Modulo	`$y = 14 % 3;`	$y will contain 2

++	Increment	`$y = 7;`
		`$y++;`
			$y will contain 8

--	Decrement	`$y = 7;`
		`$y--;`
			$y will contain 6

**	Exponentiation	`$y = 8 ** 8`	$y will contain 64

PHP Bitwise Logical Operators

~	Bitwise NOT operator: Inverts each bit of the single operand placed to the right of the symbol
&	Bitwise AND: Takes the logical-bitwise AND of two values
|	Bitwise OR operator: Takes the logical-bitwise OR of two values
^	Bitwise XOR: Takes the logical-bitwise exclusive-OR of two values

PHP Bitwise Shift Operators

<< Left shift: Shifts the left operand left by the number of places specified by the right operand filling in with zeros on the right side.

>> Sign-propagating right shift: Shifts the left operand right by the number of places specified by the right operand filling in with the sign bit on the left side.

>>> Zero-fill right shift operator: Shifts the left operand right by the number of places specified by the right operand filling in with zeros on the left side.

PHP Control Structures

- Control structures consist of several reserved words combined which allow the computer to decide which parts of code to execute, which to jump over, and which to execute multiple times.

- Keywords examples:
  - **if**, **if/else**, **elseif**, **switch**, **while**, **for**, **foreach**

PHP If Statement

- The code below represents the syntax of a typical if-statement:

```
if ($grade >= 90){
    echo "Student's grade is A";
}
```

- If grade was below 90, the computer would simply skip this instruction.

PHP If Else Statement

- The programmer can string together if-statements to allow the computer to select from one of several cases using elseif and else. (Note that Java allows **else if** while PHP uses **elseif**.)

```
if ($grade > 93)
  print "Student's grade is an A";
elseif ($grade > 89)
  print Student's grade is an A-";
else
  print "Student did not get an A";
```

PHP Comparison Operators

>	Returns true if the first value is greater than the second
>=	Returns true if the first value is greater than or equal to the second
<	Returns true the first value is less than the second
<=	Returns true if the first value is less than or equal to the second
==	Returns true if first value is equal to second
!=	Returns true if first value is not equal to second

PHP Logical Operators

! The **not operator**, returns true if its operand is zero or false

&& The **and operator**, returns false if either operand is zero or false

|| The **or operator**, returns false if both operands are zero or false

PHP Switch Statement

- **Switch** statements can be used as alternative to if, elseif, else

```
switch($user_input)
{
  case 1:
    echo "You picked 1";
    break;
  case 2:
    echo "You picked 2";
    break;
  default:
    echo "You did not pick 1 or 2";
    break;
}
```

PHP While Loop

- PHP uses the **while loop** just like Java.
- Like the if-statement, this format also uses **a Boolean expression** placed between two parenthesis
- If the Boolean expression evaluates to true, the program continues to repeatedly execute the code between the curly brackets.
- To avoid an infinite loop, there must be something in the while loop that changes the Boolean expression to false, thus causing the loop to stop, or a break statement must be called manually.
- Try to develop your logic around the use of Boolean flags rather than relying on break statements. This helps the flow of instruction logic to become clearer when maintaining code.

PHP While Loop Example

The while loop works by repeatedly evaluating the Boolean expression and then executing the code in the block repeatedly, until the Boolean expression evaluates to false. The Boolean expression is checked each time before a block executes.

```
//counts from 1 to 71, displaying each value
$count = 1;
while($count < 72){
  print "$count ";
     $count++;
   }
```

PHP Do While Loop

- The **do/while** loop works the same as a while loop (which is kind of like a while/do loop, without saying it) except that the condition is evaluated at the end of the loop rather than the beginning. T
- he code block of a do/while loop will always run at least once.

```
$count = 1;
do{
  print "$count ";
  $count++;
} while($count < 72);
```

PHP For Loop

- In the two previous cases, a counter was used to count our way through a loop. This task is suited much better to a **for loop** because you create the counter in the for loop declaration.

```
for ($count = 1; $count < 72; $count++)
{
  print "$count ";
}
```

PHP Type Conversion

- Different programming languages deal with variable types in different ways. Some are strict enforcing rules such as not allowing an integer value to be assigned to a float.
- The process of converting from one data type to another is called **casting**.
- To convert from one type to another, place the type name in parenthesis in front of the variable to convert from.
- In some cases, there are functions that perform the type conversion too.

PHP Example Type Conversion by Name

```
$ivar = (int) $var;
$ivar = (integer) $var;
$bvar = (bool) $var;
$bvar = (boolean) $var;
$fvar = (float) $var;
$svar = (string) $var;
```

PHP Example Type Conversion by Function

```
$ivar = intval($var);
$bvar = boolval($var);
$fvar = floatval($var);
$svar = stringval($var);
```

PHP Type Conversion

- PHP can also automatically convert types.
- If a variable is used as if it were a different type, the PHP script engine assumes a type conversion is needed and does it for you.
- Examples:

```
$var = "100" + 15;   // var$ set to integer = 115
$var = "100" + 15.0; // var$ set to float = 115
$var = 15 + " bugs"; // var$ set to integer = 15
$var = 15 . " bugs"; // var$ set to string = "15
bugs"
```

PHP Printing Strings

PHP programming primarily involves generating strings of HTML output to do consumed by the browser.

- **echo**
  - Can come with or without parenthesis
  - Can output multiple elements delimited by commas, but only without parenthesis
- **print()**
  - Same as echo except that it uses parenthesis and only outputs a single element, i.e., no comma delimited list of strings is allowed

PHP Printing Strings

- **printf()**
  - Let’s the programmer format a string to a detail specification
  - Derived from C library (many languages support the use of the formatted string)
  - First parameter of printf() is the formatted string with optional placeholders preceded by a %
  - More documentation online

PHP Cleaning Strings

- Users often enter string data without good formatting. PHP offers several functions to help us clean up the strings.
- There are three functions used to trim leading and/or trailing whitespace (whitespace includes spaces, tabs, newlines, and carriage returns)
  - **trim()** – returns string with leading and trailing whitespace removed
  - **rtrim()** – returns string with trailing (right) whitespace removed
  - **ltrim()** – returns string with leading (left) whitespace removed

PHP String Formatting

- Other string functions include:
    - **strtolower()** – change all characters to lower case
    - **strtoupper()** – change all characters to upper case
    - **ucfirst()** – capitalize first character of string
    - **ucwords()** – capitalize first character of each word

PHP Escape Characters

Escape sequence	**Character represented**
\"	Double quotes
\n	Newline
\r	Carriage return
\t	Tab
\\	Backslash
\$	Dollar sign
\{	Left brace
\}	Right brace
\[	Left bracket
\]	Right bracket
\0 through \777	ASCII character represented by octal value
\x0 through \xFF	ASCII character represented by hex value

PHP Encoding and Escaping

- Here are some functions that help to escape specific types of string input

- **htmlentities()** - replaces all characters with HTML entity equivalents (except for whitespace) into those equivalents, e.g., >, <, &

- **strip_tags()** to prevent users from inserting their own html tags

PHP Comparing Strings

- The following operators work on comparing strings
- **==, ===, <, <=, >, >=**
- The most reliable way to compare to strings is with the functions **strcmp()** and **strncmp().**
- The return values of **strcmp()** and **strncmp()** are:
  - 0 if the strings are equal
  - 1   if string2 comes alphabetically before string1
  - –1 if string1 comes alphabetically before string2

PHP Manipulating Strings

- The list below represents some of the string functions that operate on substrings. More documentation online.

Function Syntax	Function Description
string substr(source, start[, length])	Returns a substring of source starting at start of length length. If length is left out, substring ends at end of source.
int substr_count(source, substring[, offset[, length])	Returns the number of times a substring occurs in source starting at offset of length length. If offset is left out, it's assumed to be 0. If length is left out, substring ends at end of source.
integer strpos(source, substring[, offset])	Returns the index of the position where the substring first appears in the source. If offset is included, search starts from that index. Returns false if not found. (Remember === operator – is identical!)
substr_replace(source, replace, start[, length])	Starting at position start, inserts replace into source. length identifies the number of characters being replaced, and when omitted, replaces to end of source.

PHP Regular Expressions

- For advanced pattern matching, PHP supports regular expressions
- **preg_match()** – performs a regular expression match
- See PHP documentation for details

PHP Other Useful String Functions

- **join(delimiter, array)** takes an array of elements and returns one long string placing the delimiter between each array element.

- **explode(delimiter, string[, limit])** returns an array of strings, each of which is a substring of the original string divided at the string separator. If limit is used, the maximum number of elements will be set to limit, the last element of which will contain the rest of string.

PHP Arrays

- Most languages have **associative arrays** (sometimes called dictionary, mapping, or key-value pairs)
- Unlike most other languages which use add-ins or libraries, PHP's associative arrays are built in as their standard arrays
- Like a dictionary, the values stored in a PHP array may be of mixed type
- Unlike most dictionaries, the key (index) can be either a string or an integer.

PHP Creating Arrays

- Arrays are declared using array keyword

```
$monsterArray = array();
```

- Array elements may be initialized by using a list of items in parenthesis after array keyword

```
$monsterArray = array (“Frankenstein”, ”Dracula”);
```

- Arrays may contain mixed data types. This will be important when retrieving database records, as each row of data will be represented as an array.

PHP Adding Values to Arrays

- Array elements can be added to existing arrays by assigning values to new, unset indices/key.

```
$monsterArray[] = “Weresquirrel”;
```

- If no index is specified, the values is assigned to the next available index.

PHP Printing Arrays

- Individual elements from an array can be printed simply by referencing their index.

```
print $names[2];  // Outputs Gene
```

- To debug an array, you can also print out the entire array using the print_r() function.

```
// Following displays:
//  "Array ( [0] => Bob [1] => Linda [2] =>  Gene
[3] => Louise [4] => Tina)"
print_r($names);
```

PHP Associative Arrays

- A nice feature of PHP arrays is the ability to use array index values other than integers starting at 0.

- By using the => operator, a different index can be assigned to an array element.

```
$myArray = array(“first”=>”Hello”,“second”=>”World”);
```

PHP Associative Arrays

- For example, the following array uses indices equal to the first four powers of 2.

```
//Displays "Array ( [1] => 23 [2] => 45 [4] =>13
[8] => 96 )"

$p2 = array(1=>23, 2=>45, 4=>13, 8=>96);
print_r($p2);
```

PHP Associative Arrays

- This same method works if you want to use strings as indices to an array.
- For example, the following array uses indices equal to the first four powers of 2.

```
$si = array("first"=>23, "second"=>45,
    "third"=>13, "fourth"=>96);
$si["second"];  // Outputs "45"
print_r($si);
```

PHP More About Arrays

- IMPORTANT: Arrays are mappings, and as such do not force index values to have order.
- Don’t have to quote single-word strings, i.e., `$age['fred']` is the same as `$age[fred]`
- Considered good PHP style to use quotes to avoid confusion with constants.
- Quotes are necessary when using variable interpolation, i.e., `$age["Person{$number}"]`

PHP More About Arrays

- The stored order of elements in arrays corresponds to the order in which they are declared.

- When new elements are added without an index to an existing array, one is added to the largest existing integer index to create the new index.

- If no largest integer index exists, 0 is used.

PHP Multidimensional Arrays

- There are times when arrays must contain data in more than one dimension. Below is an example of a 2- dimensional array or matrix.

```
$row0 = array(23, 19, -4, 3);
$row1 = array(42, -9, 9, 5);
$row2 = array(51, 33, 1, -8);
$matrix = array(0 => $row0, 1 => $row1, 2 =>
      $row2);
```

PHP Arrays and Foreach Loops

- Because of the possibility for arrays with unusual indices, PHP provides a simple method for "visiting" each element of an array without needing an integer index.
- The "foreach" loop steps through an array one index at a time based on their stored order. Note that **indexname** is **optional**. You may create a foreach loop that simply goes through the values. Syntax:

```
foreach(arrayname as [indexname =>] varname) {
  // Code where current array element of arrayname
  // is referenced using varname with an index of indexname
}
```

PHP Advanced Array Traversal

- A **pointer** is associated with each PHP array. Programmer has ability to manually control this pointer with the following functions.
  - **each**($array) – Is used to loop over the elements of an array, processing elements according to their internal order.
  - **current**($array) – Returns the element currently pointed at by the iterator
  - **reset**($array) – Moves the iterator to the first element in the array and returns it
  - **next**($array) – Moves the iterator to the next element in the array and returns it
  - **prev**($array) – Moves the iterator to the previous element in the array and returns it
  - **end**($array) – Moves the iterator to the last element in the array and returns it
  - **each**($array) – Returns the key and value of the current element as an array and moves the iterator to the next element in the array
  - **key**($array) – Returns the key of the current element
  - **rewind**($array) – Moves the iterator to the first element in the array
  - **valid**($array) – Returns true if the iterator currently points at a valid element, false otherwise

PHP Some Array Functions

- boolean **array_key_exists**(index, array) - Checks for an array index within an array and returns true if it exists
- array **array_keys**(array) - Returns the keys of an array as the elements of a new array
- int **count**(array) - Returns the number of elements in an array
- array **array_fill**(start_index ,count, value) - Returns an array of size count, each element set to value, with indices starting at start_index
- array **range**(low, high [, step]) - Creates and returns an array filled with a sequence of values starting with a value low and ending at a value high with an optional step
- number **max**(array_of_numbers) - Returns the maximum element of an array
- number **min**(array_of_numbers) - Returns the minimum element of an array
- boolean **in_array**(value, array) - Returns a boolean true if it finds value in array
- mixed **array_search**(value, array) - Returns the array index instead of a boolean true if value is found and a false if the value is not found.

PHP Even More Array Functions

- array **array_merge**(array1, array2,…) - Returns an array which is the result of combining 2 or more arrays
- array **array_reverse**(array [,preserve]) - Reverses the order of the elements in an array. It can also be told to preserve the indices/keys which would result in the indices/keys also be reversed.
- number **array_sum**(array) - Returns sum of values in array as int or float. Can compute average by combining with count()
- array **array_diff**(array1 , array2 …) - Returns array of values that are in array1, but not any of the other arrays.
- bool **sort**(array) - Rearranges array elements in ascending order
- bool **rsort**(array) - Rearranges array elements in descending order
- bool **asort**(array) - Rearranges array elements in ascending order keeping keys associated with elements
- bool **arsort**(array) - Rearranges array elements in descending order keeping keys associated with elements
- bool **ksort**(array) - Rearranges array elements in ascending order of keys
- bool **krsort**(array) - Rearranges array elements in descending order of keys

PHP OOP

- PHP is an object-oriented programming (OOP) language
- Supports classes and objects, like Java
- The concept of objects is the same across many programming languages, OOP is a programming paradigm which many language support
- However, there are some syntactical differences between Java and PHP classes

PHP Class Files

- Unlike Java, PHP doesn't require a new file with each class definition. In fact, all classes could be defined in a single PHP file.
- It is good design practice to keep all class definitions in one or more separate “library” files.
- These files should be separate from your “view” files, i.e., the ones used to generate output for the client.

PHP Include Files

- Code from another file can be inserted into a PHP script using include() and require(). *Similar to Java's import*
- Can use to make class library files visible to the "view" files.
- Four versions:
  - **include**(URL_str) – includes the file referenced by URL_str
  - **require**(URL_str) – same as include() except that an error during the include is fatal and will stop processing
  - **include_once**(URL_str) – same as include() except that a 2nd include() is ignored
  - **require_once**(URL_str) – same as require() except that a 2nd require() is ignored

PHP Class Definition

- A class is the definition used to create an instance or object.
- A class definition defines the class' name, its variables, and functions, i.e., everything about the object.
- A class definition can also contain functions used to initialize objects, **constructors**, and remove them, **destructors**.

PHP Class Basic Format

```
<?php
class ClassName {
  // Member variables
  public $variable1 = 0;
  protected $variable2 = "String‟;
  private $variable3 = true;

  // Member functions
  function classFunction($_arg1 = 0, $_arg2){
    // Function code goes here
  }
}
?>
```

PHP Class Definition Format

- The keyword **class** followed by the class name is used to start the definition.
- Curly brackets are used to enclose all the elements of class definition.
- The keywords **public**, **protected**, and **private** are used to identify class variables. These keywords are access modifiers that define how class variables can be used.
- Variables can be initialized. Every time a new **object** is created, the variables for that object are initialized to these values.
- Functions are defined normally, but when contained within the curly brackets of the class, become member functions of the class.

PHP Access Modifiers

- Variables and functions can be declared as public, protected, or private
- Default is public
- Levels:
  - public – can be accessed everywhere
  - protected – can be accessed only within the class itself and by and classes that inherit from parent class
  - private – can be accessed only by the class itself

PHP Object Creation

- Just like Java, PHP uses the keyword **new** to create a new object from a class.

```
$_myobject = new ClassName(args);
```

- Syntax elements:
  - Just like variables, the name used to identify the object needs to begin with '$'.
  - Many classes need arguments (the "args" part of the above example) in order to create a new object. These are passed to a function called a constructor which initializes the object.

PHP Object Components

- In Java, we used periods to delimit the parts of an object hierarchy. For example:

```
System.out.println("Hello, World!");
```

- In PHP, the operator "->" is used to delimit the elements of an object hierarchy. For example:

```
$object_name->object_function();
```

- The above example refers to a function (note the parenthesis). The same format is used for properties too, i.e.,

```
$object_name->property;
```

PHP Private Variables

- It may be necessary to allow code external to the class access to private or protected variables
- Use **setters** (mutators) and **getters** (accessors)
- If a variable needs to be modified from outside the class, a function can be provided to do so. For example, instead of:

```
$_object -> variable = 25;
```

  use

```
$_object -> setVariable(25);
```

PHP Static Member Variables

- Each time an object of a class is created, a whole new set of variables and functions for that object is created along with it.
- It is possible to make it so that regardless of the number of objects of a class, only a single variable is created for that class.
- This allows **all objects to share a single variable.**
- To do this, follow the visibility keyword (public, protected, or private) with the keyword **static** in the variable declaration.

PHP Object Constructors

- When an object is created, it may be necessary to go through an initialization process.
- This initialization process might be based on arguments passed from the code creating the object.
- A function can be written for a class that is automatically called whenever an object for that class is created. This is called a **constructor**.

PHP Object Constructors

- A constructor has the same format as a regular function except for the name.
- The name of a constructor in PHP is `__construct()`.
- Two underscores before the keyword construct
- Example:

```
function __construct($_arg = 0)
{
  // Code to initialize class
}
```

PHP Destructors

- It is also possible that some housekeeping or cleanup needs to be performed when an object is removed such as closing database connections.
- In this case, a **destructor** function is automatically called to close the object.
- Unlike the constructor function, no arguments can be passed to the destructor function.
- The name of a destructor is always `__destruct()`.

PHP Self Referencing Variable

- Self referencing variable in PHP is **$this**
- **$this** is similar to most OO programming language)
- Refers to the object itself
- Uses '->' pointer to refer to the functions and properties of the class

PHP Class Example

```
class Person{
   public $full_name;
   public $birthday;
   private $month_list = array ("January", "February", "March",
                    "April", "May", "June", "July", "August",
                    "September", "October", "November", "December");
// Constructor
   function __construct($first_name, $last_name, $birth_month, $birth_day,
 $birth_year)
   {
      $this->full_name = $first_name." ".$last_name;
      $this->birthday = $month_list[$birth_month-1]." ". $birth_day.", ". $birth_year;
}
// Class continued on next slide
```

PHP Class Example

```
// Print function to output person's data in HTML
    public function toString()
    {
         $_returnString = $this->full_name. ” was born
        on ". $this->birthday}";
    }
}
```

PHP Class Example

- The code to create an object and call the class function getPersonString() looks like this:

```
$person_1 = new Person("John", "Doe", 3, 24, 1974);
echo $person_1->toString();
```

- The output then will be:

```
John Doe was born on March 24, 1974.
```

PHP Forms

To make useable forms, you need to know four things

- The use of HTML **form** elements
- The use of the PHP **superglobals $_POST, $_GET,** and **$_SERVER**
- The use of the PHP function **isset**()
- The use of string functions to clean and parse user input

HTTP Messages

- For two things to communicate, they need to understand each other, i.e., speak the same language
- HTTP uses the request/response paradigm
- A single HTTP transaction consists of
  - the HTTP client program sending an HTTP request message to an HTTP server (HTTP Request)
  - the HTTP server responding with an HTTP response message (HTTP Response)
- Can also view raw headers from Network tab on Google Developer Tools

HTTP Request Format

- Browser uses an HTTP request message to request a page from a web server. Request includes:
- Request line (first line) specifies
  - HTTP command/method
  - address of requested document
  - version of the HTTP protocol being used
- Headers giving server additional information about the request
- Blank line separating headers from body
- Body containing information such as post data

HTTP Request Contents

- The format of an HTTP request message is:

```
METHOD /path-to-resource HTTP/version-number
Header-Name-1: value
Header-Name-2: value
Header-Name-n: value
[optional request body]
```

HTTP Request Contents

- Every request begins with the request line including:
  - METHOD – options for the request method include
    - GET – retrieve a resource
    - POST – update a resource
    - PUT – store a resource
    - DELETE – remove a resource
    - HEAD – retrieve the headers for a resource
- The two most common HTTP methods are GET and POST

HTTP Request Contents

Other parts of the HTTP request include

- /path-to-resource – The path to the resource, e.g., /path/index.html
- HTTP/version-number – The HTTP version used, e.g., HTTP/1.1
- Can access headers using PHP function **getallheaders**()
- Example:

```
print_r (getallheaders());
```

HTTP Request Example

```
GET /~ada/action.php?firstName=Ada&lastName=Lovelace HTTP/1.1

Host: csphplinux.northeaststate.edu

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:28.0) Gecko/20100101
Firefox/28.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: http://csphplinux.northeaststate.edu/~ada/example/

Cookie: __utma=80014147.2017374380.1331909245.1407243143.1407261695.86;
_ga=GA1.2.2017374380.1331909245; __utmc=80014147;
__utmz=80014147.1407243143.85.1.utmcsr= csphplinux.northeaststate.edu |
utmccn=(referral)|utmcmd=referral|utmcct=/ada/; badprotocol=4;
Glink_session=!FM6tyHTiwm3cHBB4lyU++bZwQpIr+JZ/UNRQZ+o7NAI9+KEzrOQE9MQTsc
o10IfZfsc=; BANSSO=1WQI5c3m996h4Y320hDqWODj0o72DNUTHQBzTQC38NIjSa4S3QG=;
fos.web.server= csphplinux.northeaststate.edu Connection: keep-alive
```

HTTP GET Requests

GET Requests

- Simplest of the request methods
- Should be used when simply requesting resources from the server and not making changes
- Does not have a body
- Form values are passed to the processing script over the URL in a query string
- Pre HTTP version 1.1 did not require headers
- HTTP version 1.1 requires the host header

HTTP GET Form Example

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8" />
   <title>Basic Get Form</title>
</head>
<body>
   <form action="action.php" method="get">
      <p>Your name: <input type="text" name="name" /></p>
      <p>Your age: <input type="text" name="age" /></p>
      <p><input type="submit" /></p>
   </form>
</body>
</html>
```

HTTP Post Requests

POST Requests

- Should be used when the intent is to change a resource on the server
- A blank line separates the headers from the body
- Form values passed to processing scripts are in the body of the message

HTTP Post Form Example

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8" />
   <title>Basic Post Form</title>
</head>
<body>
   <form action="action.php" method="post">
      <p>Your name: <input type="text" name="name" /></p>
      <p>Your age: <input type="text" name="age" /></p>
      <p><input type="submit" /></p>
   </form>
</body>
</html>
```

HTTP Response

- Server responds to a request with an HTTP response message made up of:
  - Status line (first line) specifies:
  - protocol version
  - status code
  - human readable version of status code
  - Headers
  - Blank line separating headers from body
  - Body

PHP Superglobals

- **Superglobals** – built-in variables available in all scopes
- Majority are arrays containing information available to server
  - **$GLOBALS** – References all variables available in global scope
  - **$_SERVER** – Server and execution environment information
  - **$_POST** – HTTP POST variables
  - **$_GET** – HTTP GET variables
  - **$_COOKIE** – HTTP Cookies Declared by PHP, not your code
  - **$_FILES** – HTTP File Upload variables
  - **$_ENV** – Environment variables

PHP $_SERVER

This is a list of some of the $_SERVER indices. The index name is descriptive of the data contained at that location in the array.

- **PHP_SELF** - The name of the current script
- **SERVER_NAME** - The hostname, DNS alias, or IP address for self-referencing URLs
- **REQUEST_METHOD** - The method the client used to fetch the document (e.g., "GET")
- **QUERY_STRING** - Everything after the ? in the URL
- **REMOTE_ADDR** - A string containing the IP address of the machine that requested this page
- **CONTENT_LENGTH** - The length of the information attached to queries such as POST
- **HTTP_REFERER** - The page the browser said it came from to get to the current page
- **REQUEST_TIME** – The timestamp of the client request (in seconds since 01/01/1970).

PHP Form Example

This form explicitly defines its processing script (action), a literal.

```
<form method='get' action=‘processingScript.php'>
   Name: <input type='text' name='name'><br>
   Phone: <input type='text' name='phone'><br>
</form>
```

PHP Self Processing Form

- This form will process itself, somewhere else in the .php file. This script makes use of the $_SERVER[‘PHP_SELF’] variable to always know its own file name.

```
<form method='get' action=‘<?php echo $_SERVER['PHP_SELF']; ?>’>
   Name: <input type='text' name='name' /><br />
   Phone: <input type='text' name='phone' /><br />
   <input type='submit' name='submit' value='Click Here' />
</form>
```

PHP isset() Function

- Many times, values from your PHP script are used to populate form elements
- If not set, a warning or a blank field will be inserted into form element
- Remember that **isset**() is used to determine if a variable contains a non-NULL value
- isset() tests if a variable has a value.
- Unfortunately, isset() will return a true if the variable is set to an empty string, so you will want to combine isset() with the string function strlen()

PHP Form Example Using isset()

```
<form method='get' action='<?php echo $_SERVER['PHP_SELF']; ?
>'>
Name: <input type='text' name='name' value='<?php
if ((isset($_GET['name'])) && (strlen($_GET['name']) > 0))
   echo $_GET['name'];
else
   echo 'Enter name here';?>'><br>
Phone: <input type='text' name='phone' /><br>
<input type='submit' value='Click Here' /></form>
```

PHP Handling User Input

- You have no control over a user's form input
- Scripts must examine all input to prevent unintentional characters from causing erroneous execution
- malicious input from breaching security
- JavaScript on client side could do form validation, but…
  - Mostly for client's convenience
  - Client might have JavaScript disabled
  - Attackers could create forms simulating input from legitimate forms

PHP Handling User Input

- Always validate form input
    - Clean input with built-in functions for escaping control characters
    - Use form data to assign values to hard-coded variables
- To eliminate confusion, all forms should indicate to the client
    - which fields are required
    - the format and type of information a field is expecting where applicable

PHP Testing Input for Numbers

- To test if a submitted value is a number, use the **is_numeric**() function
- **is_numeric**() returns a boolean true if the value is a number
- Unsigned hexadecimal is considered valid, i.e., 0xe4
- There is also **is_integer**(), **is_double**(), and **is_float**()

PHP Other Verification Tips

- Name the submit button and test that name using isset()
- Hidden form elements can be used to pass data to a PHP script without being too obvious to client
- Hidden elements can also be used to identify the form that requested the page or passing constants to the server-side script
- Never use hidden elements to store secure information as the HTML can be viewed by the client.

PHP Methods to Verify Clients

- **$_SERVER['HTTP_REFERER']** returns the address of the page that referred the user to this script.
- **$_SERVER['REQUEST_METHOD']** returns the method of the form used to refer the user to this script.
- **$_SERVER['REMOTE_ADDR']** returns the IP address of machine originating request. Can use this to limit which machines have access to your script.

PHP Other String Sanitization Functions

- **urlencode**() - Returns a string encoded for use as a URL. Spaces are replaced with plus signs (+) and non-alphanumeric characters except dashes, underscores, and periods are replaced with a percent (%) sign followed by two hex digits.
- **urldecode**() - Replaces %## encoding with corresponding character and plus signs with a single space.
- **addslashes**() - escapes single quote ('), double quote ("), backslash (\) and NULL. Used to prevent PHP control characters from entering form data.
- **stripslashes**() - Replaces \' with ', \" with ", \\ with \, etc. Used in case injection of escape characters could pose a threat.

PHP Sticky Form

- Making your form “sticky” prevents making a user re-populate a form if they have forgotten a required field on a submitted form.
- Pre-populate form as much as possible with value attribute
- Example:

```
<input type='text' name='first_name' value='<?
 if(isset($_GET['first_name']))
    echo $_GET['first_name']; ?>' />
```

PHP Sticky Form Tips

- Radio buttons and check boxes are turned on using **checked="checked"**.
- An option from a select input is pre-selected using **selected="selected".**
- The text within a text area is pre-set by placing the text between the **textarea** tags.

PHP htmlspecialchars()

- Certain characters have special significance in HTML and should be represented by HTML entities if they are to preserve their meanings. This function returns a string with these conversions made.

Specifically, the function translates the following characters:

  - ‘&' (ampersand) becomes '&amp;'
  - " (double quote) becomes '&quot;' when ENT_NOQUOTES is not set.
  - ' (single quote) becomes '&#039;' only when ENT_QUOTES is set.
  - '<' (less than) becomes '&lt;'
  - '>' (greater than) becomes '&gt;'

PHP Populate Select Boxes

- Some form elements are tedious to generate, e.g., select boxes
- Use arrays and PHP to generate them for you

```
$months = array(1=>'January', 'February', 'March’…);

echo "<select name='month'>\n";

foreach ($months as $key=>$value)
  echo "<option value=\"$key\">$value</option>\n";

echo "</select>\n";
```

PHP Interacting with Databases

- PHP supports many databases including:
  - MySQL
  - SQLite
  - PostgreSQL
  - Oracle
- In this class, we will be using the built-in PHP Data Objects (PDO) system to access MySQL/MariaDB databases and handle errors
  - Abstract PDO allows simple switching between database engines
  - Abstraction takes away from performance a little

PHP Object Review

- PDO is object-based, and hence, you will need to know how to access objects, their methods, and their properties
- Use the new keyword to create an object from a class

```
$introToScripting = new Course;
$introToScripting = new Course("CITC", 1317);
```

- Use the -> notation to access methods and properties of an object

```
$introToScripting -> $setNumberOfStudents(31);
$enrolled = $ introToScripting -> $numberOfStudents;
```

Database Concepts

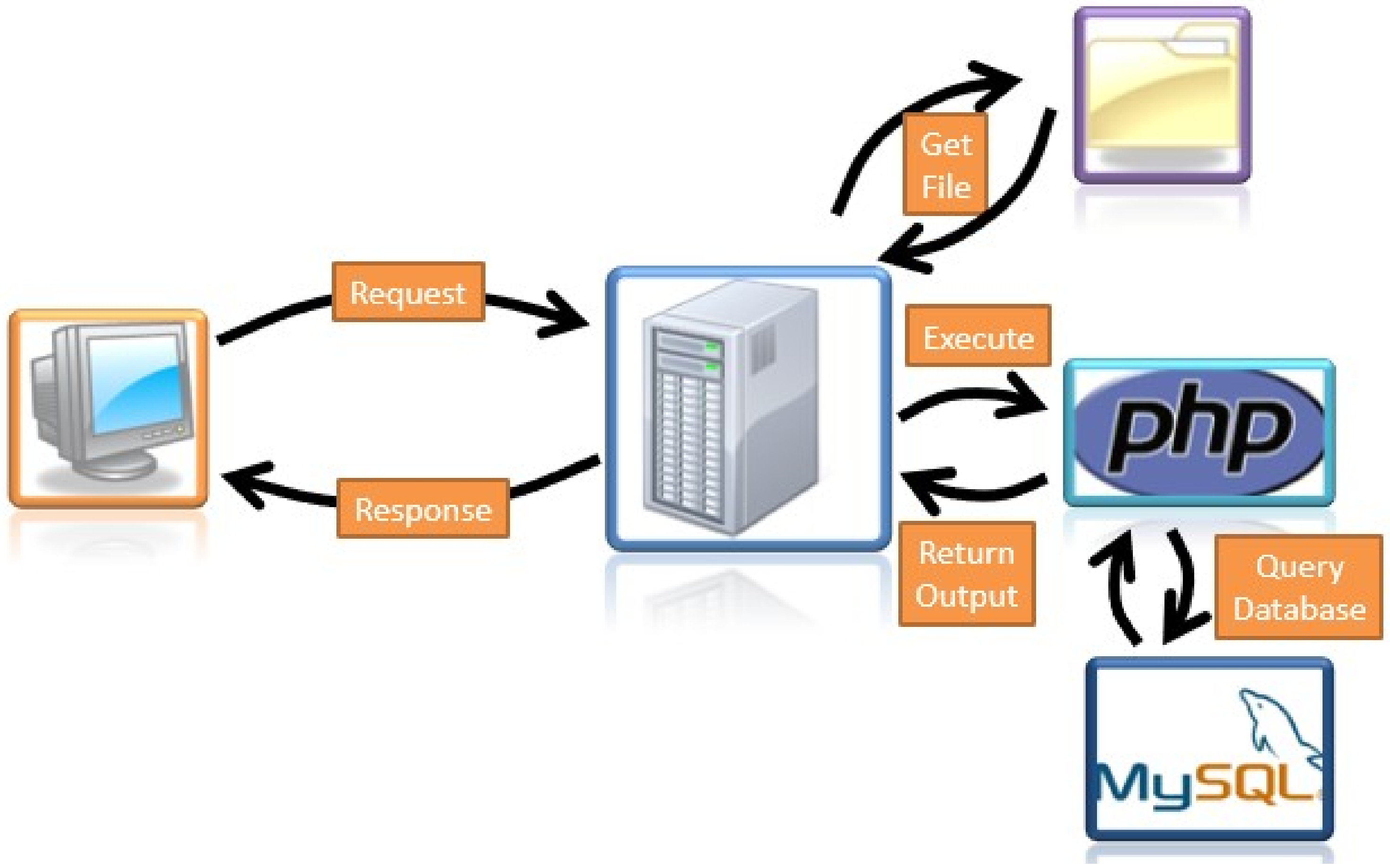
Why store data in a database?

- Let the database do the heavy lifting with well-designed queries
- Tables are grouped together into databases
- Databases hide storage details from programmer
    - Doesn’t care how the data is stored
    - Doesn’t care where the data is stored
- Supports multi-user environments
    - Transactions provide recovery for errors during database operations
    - Locking is used to gain exclusive access to tables or rows during periods of heavy traffic

Database Commands Refresher

```
INSERT INTO books VALUES (null, 4, 'I, Robot', '0-553-29438-5', 1950, 1);

INSERT INTO books (authorid, title, ISBN, pub_year, available)
        VALUES (4, 'I, Robot', '0-553-29438-5', 1950, 1);

DELETE FROM books WHERE pub_year = 1979;

UPDATE books SET pub_year=1983 WHERE title='Roots';

SELECT * FROM books WHERE pub_year > 1979 AND pub_year < 1990;

SELECT title, pub_year FROM books
        WHERE pub_year > 1979 AND pub_year < 1990;

SELECT authors.name, books.title FROM books, authors
        WHERE authors.authorid= books.authorid;
```

PHP and Database Data Flow



MySQL / MariaDB

- Remember the process for accessing data from a database using MySQL:
- Log onto MySQL:
  ```
  mysql -u username -p password
  ```
- Select a database to work with:
  ```
  use database;
  ```
- Send a query to one or more tables, e.g.,
  ```
  SELECT * FROM books WHERE pub_year > 1979;
  ```
- MySQL returns results (typically in the form of a text-based table)
- When you’re finished, exit MySQL using `exit`
- We will be using the same steps in PHP using PDOs

PHP Data Objects (PDO)

- PDO is a PHP class with properties and methods used to interact with databases
  - PDO takes advantage of the latest PHP 5 internals for performance
  - PDO uses buffered reading of result set also for performance
  - PDO has drivers for almost all databases
  - PDO provides access to common DB features
  - PDO also provides access to DB-specific functions
  - PDO supports transactions

PHP Connecting to Database

- The first thing to do is make a connection to the database using an instance of the PDO class

```
$db = new PDO ($dsn, $username, $password);
```

- **$dsn** represents the data source name and is a string that defines the database engine, the **hostname**, and the **database name**

```
"mysql:host=hostname;dbname=database"
```

- When accessing a database on the same server as the PHP script, the hostname should be “**localhost**”.

```
$db = new PDO("mysql:host=localhost;dbname=zxyx999",
"zxyx999", "12345");
```

PHP PDO Attribute Connections

- The PDO method setAttribute sets the attributes on a the database handle
- Some examples include:
  - PDO::CASE_LOWER: Force column names to lower case
  - PDO::CASE_NATURAL: Leave column names as returned by database driver
  - PDO::CASE_UPPER: Force column names to upper case
  - PDO::ERRMODE_SILENT: Just set error codes
  - PDO::ERRMODE_WARNING: Raise E_WARNING
  - PDO::ERRMODE_EXCEPTION: Throw exceptions
- Definition: bool PDO::setAttribute(int $attribute, mixed $value)
- Example:

```
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

PHP Basic PDO Interaction

- Querying the database is done through the PDO object

```
$myQuery = "SELECT * FROM tablename";
$results = $db->query($myQuery);
```

- Returns an array containing the results of the query
- **The following basic code should only be used for:**
  - *simple SQL statements*
  - *queries that do NOT contain user input*
- **For more complex queries, you should use prepared statements**

PHP Basic PDO Interaction

- Normally, **print_r**() won’t work with standard query because of “lazy load” (Improves performance by only getting next row when needed)
- The basic code

```
$myQuery = "SELECT * FROM book";
$results = $db->query($myQuery);
print_r($results);
```

outputs:

```
[queryString] => select * from book
```

PHP Basic PDO Interaction

- To get print_r() to work, use **fetchAll**()

```
$query = "SELECT * FROM book";
$results = $db->query($query)->fetchAll();
print_r($results);
```

- foreach, however, works fine without using fetchAll()

```
$results = $db->query("SELECT * FROM book");
foreach($results as $row)
  echo "$row[book_id]: $row[title]\n";
```

PHP PDO Prepared Statements

- The PDO class allows for what are known as **prepared statements** or parameterized queries.
- Prepared statements should be used for queries containing user input

```
$statement = $db->prepare("SELECT * FROM book");
$statement->execute();
while($row = $statement->fetch()) {
  echo "$row['book_id']: $row['title']\n";
}
$statement = null;
```

PHP PDO Prepared Statements

- Prepared statements use placeholders to pass data values through an array argument of the execute method. Notice the syntax of the execute stamen. Parameters to the query are passed in via an array.

```
$q = "SELECT * FROM book WHERE pub_yr > :year";
$stmnt = $db->prepare($q);
$stmnt->execute(array('year'=>$_GET['year']));
while($row = $stmnt->fetch())
{
   echo $row['title'].", pub date: ".$row['pub_yr'];
   echo "<br />";
}
$stmnt = null;
$db = null;
```

PHP PDO Prepared Statements

- Array argument of the execute method can have multiple elements

```
$q = "SELECT * FROM book WHERE pub_yr BETWEEN
           :beg AND :end";
$statement = $db->prepare($q);
$statement->execute(array(
'beg'=>$_GET['start'],
'end'=>$_GET['finish’])
);
```

PHP PDO Prepared Statements

- The benefits of PDO prepared statements include:
  - The same SQL command can be executed with different arrays
  - PDO cleanses user input to add layer of protection against malicious input
  - Question marks can be used as positional place holders

```
$q = "INSERT INTO books (author, title, pub_yr)“
. "VALUES (?,?,?)";
$statement = $db->prepare($q);
$statement->execute(array(“Stephen King",
        “The Shining", 1977));
```

PHP Prepared Statements w/ bindparam()

- Another “long hand” method for binding parameters to **named placeholders** is using **bindparam().**

```
$q = "SELECT * FROM book WHERE pub_yr BETWEEN :beg
   AND :end";

$statement = $db->prepare($q);

$statement->bindparam(":beg", $_GET['start']);

$statement->bindparam(":end", $_GET['finish']);

$statement->execute();
```

- notice no array passed into execute function because parameters have been set with bindParam function. Examine the differences in each way to use the execute method

PHP Prepared Statements w/ bindparam()

- Another way to use **bindparam()** is to binding parameters to **unnamed placeholders** (by order of ? appearance), as opposed to a named placeholder.

```
$q = "SELECT * FROM book WHERE pub_yr BETWEEN
   ? AND ?";
$statement = $db->prepare($q);
$statement->bindparam(1, $_GET['start']);
$statement->bindparam(2, $_GET['finish']);
$statement->execute();
```

Database Transactions

- If RDBMS supports transactions, attempted changes made with **commit()** can be discarded using **rollback()**
- Typically, this is due to a violation of one of the four properties such as simultaneous actions being performed on a table (violates isolation)
- If you call commit() or rollback() on a database that doesn’t support transactions, DB_ERROR is returned
- Requires try-catch programming construct
- Prepared statements made outside of transaction will not be visible inside transaction and vice versa

PHP Transaction Example

```
try {  // Attempt connection to database
  $db = new PDO("mysql:host=localhost;dbname=zxyx999", "zxyx999", "12345");
}
catch (Exception $error) {  // If attempt failed, send error
  die("Connection failed: " . $error->getMessage());
}
try {  // Try modifying table
  $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $db->beginTransaction();
  $db->exec("INSERT INTO book (book_id, title) VALUES ('KOO00', 'Odd Thomas')" );
  $db->exec("INSERT INTO book (book_id, title) VALUES ('KOO01', 'The Taking')" );
  $db->commit();
}
catch (Exception $error) {
  $db->rollback();
  echo "Transaction not completed: " . $error->getMessage();
}
```

PHP PDP Closing a Connection

- It is good practice to close all database connections when you are done using them
- This is done by destroying the PDO object by setting it to null

```
// Open the connection
$db = new PDO('mysql:host=localhost;dbname=zxyx999',

  'zxyx999', '12345');
// Use the connection (not shown)
// Now close the connection
$db = null;
```

PHP heredocs to Create Long Queries

```
$query = <<< QUERY
  select *
  from book
  where pub_year > :year
  and price > :price
QUERY;
$statement = $db->prepare($query);
echo $query;
$statement->execute(array(
  'year' => 1960,
  'price' => 5.00)
);
```

PHP Maintaining State

- HTTP is a stateless protocol, i.e., once a server provides a page to a client, the interaction is forgotten
- PHP resources exist to provide the client with state including
  - Sessions
  - Cookies
  - File storage

PHP Storing Application Data

- Multiple methods exist for storing client-specific data for a page/application
- On client’s machine
  - Cookies
  - Local storage (HTML5)
- On server
  - Database
  - Session variables
  - Client upload files

PHP Cookies

- Cookies allow a loaded page to read/write data to a client's computer
- Cookies are included as an HTTP header
- Cookies are stored as text on a client's machine
- The total size for all cookies from a single domain should be less than 4k
- A cookie can be set with expiration date (No expiration date forces cookie to be deleted as soon as client's browser is closed.)
- Possibly disabled – check if cookies are enabled by setting, then checking to see if set

PHP Using Cookies

- Because cookies are sent with HTTP header, cookies must be created before any output is generated including white space.
- Cookies created using:
  ```
  bool setcookie($name [, $value [, int
  $expire]])
  ```
- $**name** is the name to refer to the cookie
- $**value** is the value set to the name
- $expire is the expiration date in seconds since midnight of January 1, 1970

PHP Using Cookies

- **setcookie**() returns false if PHP script had output before cookie was set
- A return value of true from setcookie() only means code was executed, not that the cookie was successfully set at client's machine
- Once set, a cookie can be retrieved by the next page accessed by the client using the $_COOKIE array
- Delete cookies by setting them with setcookie() using an expiration date from the past.

PHP Using Cookies Example Code

```
<?php
 setcookie("firstname","David",time()+60*60*24*30);
 setcookie("value","37",time()+60*60*24*30);
?>
<!DOCTYPE html>
<html>
<head><meta charset='UTF-8' /><title>Set Cookies</title></head>
    <body>
        <h1>Set Cookies</h1>
        <a href="viewcookies.php">Click to view cookies</a>
    </body>
</html>
```

PHP Deleting Cookie Example Code

Set the cookie to expire

```
<?php
 setcookie("firstname","",time()-1);
 setcookie("value","",time()-1);
?>
<!DOCTYPE html>
<html>
<head><meta charset='UTF-8' /><title>Delete Cookies</title></head>
    <body>
        <h1>Delete Cookies</h1>
        <a href="viewcookies.php">Click to view cookies</a>
    </body>
</html>
```

PHP Viewing Cookie Example Code

```
<!DOCTYPE html>
<html>
<head><meta charset='UTF-8' /><title>View
 Cookies</title></head>
    <body>
        <h1>View Cookies</h1>
        <?php
            print_r ($_COOKIE);
        ?>
    </body>
</html>
```

PHP Sessions

- **Sessions** are identified using a session ID (32 digit hexadecimal value)
- The session ID is transmitted between the client and server with each HTTP request and response
- Client keeps track of a session through the use of a cookie
- If cookies are not enabled, session ID is appended to URL as "get" data

PHP Sessions

- Server keeps track of sessions through locally stored text files (low traffic) or a database (high traffic)
- The server also maintains the session variables in the text file or database (depending on PHP configuration).
- Server cleans up old sessions after a specified timeout period to prevent someone hijacking an old session and to avoid clogging the server with unused sessions.

PHP Implementing a Session

- **session_start**(), serves 2 purposes:
  - creates a new session or
  - locates an existing session
- If existing session found, it loads the session's variables to the superglobal array $_SESSION
- Because of dual role of session_start(), use isset() on a known session variable to see if session already exists
- Since session ID is sent as a cookie in an HTTP header, it must be sent before any HTML is generated, i.e., session_start() must be executed before any output is generated, even white space.

PHP Implementing a Session

- A variable can be removed from a session using the unset() function

```
unset($_SESSION['quantity']);
```

- All session variables can be removed by re-initializing $_SESSION array

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

PHP Session Variable Types

- A session variable can be of any type or object
- If a session variable is an object, be sure to define the object before running session_start()
- If an existing session that uses an object is opened before the object is defined, it will cause problems

PHP Sessions

```
<?php
    session_start();
    if(!isset($_SESSION['count']))
        $_SESSION['count'] = 0;
    $_SESSION['count']++;
?>
<!DOCTYPE html>
<html>
  <head>…</head>
<body>
    <?php
        echo "You've visited ".$_SESSION['count']." times.";
    ?>
</body></html>
```

PHP Ending a Session

- If you receive a signal from the client that they are ending the session, e.g., clicking on a link to log out, there is a method you can use to force a session to end.
- First, you must open the session so that it is identified as the one to be closed
- Next, you must end the session with the function **session_destroy**()
- Last, use the header function to redirect client’s browser

PHP Ending a Session

```
<?php
  // Begin by accessing the session
  session_start();
  // Close the session
  session_destroy();
  // Direct output to the client
  header("Location: logout.html");
?>
```

PHP Redirections

- The previous slide used a redirect, this is how it works.
- Use the location header to send client's browser to a new URL
- After header, nothing else should be generated, so use the exit() method to end script

```
header("Location:
            http://www.domain.com/index.html");
exit();
```

PHP Creating and Setting Response Headers

- Typically, PHP and Apache take care of headers
- PHP offers the opportunity to generate custom headers
- You generate headers using the header() function
- Example: `<?php header("Content-Type: text/plain"); ?>`
- All headers must be set before any of the body is generated – do this at the very top of your file, even before the <html> tag
- Setting headers after document has started results in a warning

PHP Content Type Header

- The Content-Type header identifies the type of document being returned
- Example: header("Content-Type: text/plain");
- Types include:
  - "text/html" (default) for HTML documents
  - "text/plain" forces browser to display raw (plain) text
  - "text/rtf" to be interpreted as Rich Text Format (RTF)
  - "video/avi" Windows-compatible video formats
  - "application/pdf" Portable Document Format (PDF)
  - "audio/mpeg" MP3 or other MPEG audio

PHP Session Variables or Cookies?

- Session variables
  - Unlimited size (based on server resources, of course)
  - In some cases, load balancing may lose session information
  - Depending on PHP setup, sessions expire usually within about 30 minutes
  - Depend on cookies to identify session
- Cookies
  - Information must be transmitted with each request
  - Cookies can be stored for days, months, or even years
  - Clients can access/edit information stored in cookies
  - Cookies have a limited size

PHP Session Variables or Cookies?

- Cookies should be used for short, non-sensitive, “throw away” information
- Session variables should be used for non-critical information or large-scale information pertaining to a single session
- Databases should be used for sensitive information and vital information that is to be maintained for long periods of time

PHP State Expiration

- To force browsers and proxy servers to reload page after a certain date/time, use the expires header

```
header("Expires: Thu, 05 Nov 2016 12:37:46  GMT");
```

- Can also be used to force expiration in a certain amount of time.

```
$expireDate = gmstrftime("%a, %d %b %Y %H:%M:%S GMT", time()
  + 60 * 60 * 24 * 31);
header("Expires: {$expireDate}");
```

- To mark a document as expired, use the current time or a time in the past:

```
$expireDate = gmstrftime("%a, %d %b %Y %H:%M:%S GMT");
header("Expires: {$expireDate}");
```

PHP Prevent Page Caching

- The following code is the best way to prevent a browser or proxy cache from caching your document:

```
header("Expires: 0");
header("Cache-Control: no-store, no-cache,
      must-revalidate");
header("Cache-Control: post-check=0, pre-
      check=0", false);
header("Pragma: no-cache");
```

PHP File Uploads

- Forms can be used to allow client to upload files
- Form method must be POST
- Additional form tag attribute: enctype='multipart/form-data'
- Additional form input: <input name='filetoLoad' type='file' />
- Two ways to prevent client from uploading large files
    - upload_max_filesize option in php.ini (2 MB by default)
    - Set hidden form parameter MAX_FILE_SIZE as soft upper limit (Clients could spoof this, so do not rely on it)

PHP File Uploads

```
<form enctype='multipart/form-data'
    action='<?= $_SERVER['PHP_SELF']; ?>'
    method='post'>
<input type='hidden' name='MAX_FILE_SIZE'
    value='1024'>
File name: <input name='file2load' type='file'>
<input type='submit' value='Upload'>
</form>
```

PHP File Uploads

- Elements in $_FILES provide information about uploaded file
  - **name** – Browser supplied name for the uploaded file. This might not be useful information as it is user and client O/S specific
  - **type** – MIME type of the uploaded file as guessed by browser
  - **tmp_name** – Name of the temporary file where the server is holding the uploaded file. If file was too large, the name is given as "none".
  - **error** – 0 indicates no error while 4 indicates no file uploaded (For more errors, see http://php.net/manual/en/features.file-upload.errors.php)
  - **size** – Size of the uploaded file in bytes. If file was too large, the size is reported as 0.

PHP File Uploads

- Uploaded file is in temporary location – to use it, need to move it somewhere less temporary
- Script must have appropriate write privileges for where the file is being moved to
- Functions needed:
  - bool **is_uploaded_file**($filename) – returns true if the file was from an HTTP POST operation, i.e., a malicious user hasn't tried to get the script to access a critical server file.
  - bool **move_uploaded_file**($filename, $destination) - returns a true if it was able to successfully move the file to the destination location. False is returned otherwise.

PHP File Upload Example Code

```
<?php
if ($_FILES['file2load']['error'] != 0)
    die("Upload Error.");
if (!is_uploaded_file($_FILES['file2load']['tmp_name']))
    die("Attempt at file upload attack suspected");
$destdir = "/home/jnmcmeen/public_html/uploads/";
$destfile =
    $destdir.basename($_FILES['file2load']['name']);
$srcfile = $_FILES['file2load']['tmp_name'];
if (!move_uploaded_file($srcfile, $destfile))
    die("Unable to move file");
else echo "Upload successful.";
?>
```

PHP Crash Course

Updates coming including

- Security Concerns
- Exception Handling
- Web API and RESTful web services in PHP
- JavaScript Object Notation (JSON)
- Asynchronous JavaScript and PHP

Sources

- w3schools.com
- php.net
- The initial slides were created by Mr. David Tarnoff, a brilliant educator, at East Tennessee State University for their Server-Side Programming course.
- The compilation is intended for classroom and online discussion purposes only.