University of Regina

# CS 215
# Web Oriented Programming

## PHP Crash Course

Dr. Orland Hoeber
orland.hoeber@uregina.ca
http://www.cs.uregina.ca/~hoeber/cs215/

# Readings

- Chapters 3 – 7, 12
- http://www.php.net/manual/en/

# PHP

- PHP is a server-side scripting language
  - scripts are embedded within HTML documents
  - designed to handle forms, file processing, and database access
  - numerous built-in functions for array and text processing
  - easy access to form data, session tracking, and cookies
  - simple function-call access to databases

# PHP Script Processing

- When the Web server encounters a PHP script, it processes it in the same way that embedded JavaScript is processed by a browser
  - the PHP processor examines the document
  - regular HTML is simply passed to the browser
  - when PHP code is encountered, it is executed
  - any output is placed in the location of the script, and then passed to the browser
    - in most cases, this means that the output must be valid HTML
  - the browser never sees the PHP code

# PHP Language

- PHP is an interpreted language
  - you don't need to compile it
  - some PHP-enabled Web servers will pre-compile commonly used or complex PHP web pages
- PHP syntax is similar to JavaScript
  - minor differences (e.g., string concatenation, classes)
- PHP is dynamically typed
  - the type of a variable is set every time it is assigned a value
  - like JavaScript, variables may be coerced to another type if they are used in a context for which they are not valid

# Embedding PHP in HTML

- PHP can be embedded directly in an HTML document, or via an external file
  - internally, it is encapsulated within special tags

```
<?php
…
?>
```

    - optionally, PHP can be configured to allow the “php” to be excluded (<? … ?>)
  - externally, using the include function

```
<?
include (“myincludedfile.php”);
?>
```

    - PHP code can be within this included file, but must be encapsulated within the special PHP tag

# Variables

- All variables must start with $
  - variable names follow the same rules as other common languages (C, C++, JavaScript)
  - can't use common reserved words
  - case sensitive
  - no type declarations
    - an unassigned (unbound) variable has the value NULL
    - the `unset` function removes the variable assignment, giving it a value of NULL
    - the `isset` function checks to see if a variable has been assigned (bound)
- There are a set of pre-defined variables that are used for form processing and for checking the PHP settings, which we'll discuss later

# Primitive Data Types

- There are eight primitive data types
  - integer
  - double
  - string
  - Boolean
  - array
  - object
  - resource
  - NULL

# Strings

- Strings can be assigned with either single or double quotes
  - double quotes
    - embedded variables are interpolated
      - any variable names found within the string are replaced with their values

```
print("The sum is $sum.");
```

    - can use escape sequences (e.g., \n)
  - single quotes
    - embedded variables are not interpolated
    - escape sequences are not processed

# String Operations and Functions

- The only string operator is concatenation (.)

```
$newString = "hello" . $oldString;
```

- We can treat strings as arrays, accessing a single character using {d}

```
$character = $newString{3};
```

- There are a large number of string processing functions
  - strlen – returns the number of characters in a string
  - strcmp – alphabetical comparison of two strings
  - trim – removes all white space before and after a string
  - strtolower – converts the string to lower case
  - strtoupper – converts the string to upper case
  - … and many more (check php.net for the details)

# Arithmetic Operators and Coercions

- PHP supports the usual arithmetic operations

- Coercions occur based on the context of the expression
  - if a number is used in the context of a string, it is converted to a string
  - if a string is used in the context of a number, it is converted to a number
    - if the string doesn't start with a number, zero is used
    - non-number characters at the end are ignored (except for e, where this is interpreted as the exponent if it is followed by a number)

# Explicit Type Conversions

- PHP supports explicit type conversions
  - casting
    - works as expected

```
$sumInt = (int)$sum;
```

  - type conversion functions
    - three specific functions: intval, doubleval, strval

```
$sumInt = intval($sum);
```

  - settype function
    - takes the variable as the first parameter, and a string that specifies the type as the second parameter

```
$sumInt = settype($sum, “integer”);
```

# Checking the Type of a Variable

- There are two ways to check the type of a variable
  - gettype function
    - returns the type of the variable as a string

```
$type = gettype($sumInt);
```

  - type-testing functions
    - takes the variable as the parameter, and returns true if the type matches the function
      - integer: is_int, is_integer, is_long
      - double: is_double, is_float, is_real
      - Boolean: is_bool
      - string: is_string

```
if (is_int($sumInt)) {
…
}
```

# Output from PHP

- The output of a PHP script is HTML that is sent to the browser
  - embedded at the location of the script
  - many different ways to produce output
    - print
      - simple unformatted output
      - can be used with or without brackets (behaves like a function)
    - printf
      - formatted output
      - works the same as with C
    - echo
      - same as print, but does not behave like a function
    - <?=$variable?>
      - shorthand syntax

# Output Style

- Although there are a wide range of PHP coding styles, my preference is to:
  - process all the data and accumulate dynamic content in variables
  - avoid including HTML within the dynamic content
  - include all of the HTML at the end of the document
  - use the shorthand notation for inserting the dynamic content within the page

# Example

```
<?
   $d = date(“l, F j, Y”);
?>
<!DOCTYPE html>
<html lang = "en">
  <head>
    <title> today.php </title>
    <meta charset = "utf-8" />
  </head>
  <body>
    <p>Welcome to my home page</p>
    <p>Today is <?=$d?>.</p>
  </body>
</html>
```

# Control Statements

- Control expressions
  - relational operators are the same as in JavaScript (including === and !==)
  - Boolean operators are as you would expect (&&, ||, !)
- Selection statements
  - all are the same as in C (and JavaScript)
    - if-else (can also use elseif)
    - switch
    - for
    - while
    - do-while
    - foreach (also includes special syntax for accessing keys)

# Control Statements

- In the standard loop statements (for, foreach, while, do-while), there are two special control statements
  - break
    - terminates the execution of the loop construct
  - continue
    - skips the remainder of the current iteration, but continues the execution of the loop construct beginning at the next iteration

# Arrays

- The arrays in PHP are unlike the arrays in any other programming language
  - can be used like normal arrays
  - can be used like associative arrays (hashes)
  - the value types can be mixed
  - the key types can be mixed
- Arrays are implemented as mappings of keys to values (with pointers to support sequential access)
  - keys are **numbers:** traditional array
  - keys are **strings:** hash
- There are a rich set of functions for manipulating and iterating through arrays

# Array Creation

- Arrays can be created in one of two ways
  - using a variable like an array

```
$myArray[0] = 17;
$myArray[“this one”] = 55;
$myArray[] = 45;
```

  - using the array creation function `array()`
    - takes a list of values and returns the corresponding array

```
$myArray = array(17, 24, 36, “smith”);
```

    - takes one or more `key=>value` pairs as parameters, and returns the corresponding array

```
$myArray = array(“Joe”=>15, “Mary”=>33, “Tom”=>44);
```

  - for all of these, you can mix the types for the keys and the values

# Accessing Array Elements

- Individual elements can be accessed via subscripting
  - subscript contained within square brackets
  - refers to the key of the value to be referenced

```
$ages[“Mary”] = 29;

$key = 3;
$myArray[$key] = “Hello”;

<?=$myArray[3]?>
```

# Functions for Dealing with Arrays

- There are two built-in functions for accessing the keys and values in an array

```
$highs = array (“Mon”=>22, “Tue”=>23, “Wed”=>18, “Thu”=>19,
“Fri”=>22, “Sat”=>23, “Sun”=>26);
$days = array_keys($highs);
$temps = array_values($highs);
```

- There is a function for testing whether an element exists

```
if (array_key_exists(“Wed”, $highs)) {
  …
}
```

- Whole arrays or just array elements can be deleted

```
unset($highs);
unset($highs[4]);
```

# More Array Functions

- `is_array($list)` returns true if `$list` is an array
- `in_array(17, $list)` returns true if 17 is an element (value) of `$list`
- `sizeof($list)` returns the number of elements in `$list`
- `explode(" ", $str)` returns an array with the values of the words from `$str`, split on a space
- `implode(" ", $list)` returns a string of the elements from `$list`, separated by a space

# Sequential Access to Array Elements

- A set of functions provide easy access to the array elements in a sequential manner
  - by default, a pointer exists at the first element of each array
    - access this value using `current`
    - access this key using `key`
    - move the pointer forward and access the value using `next`
    - move the pointer backward and access the value using `prev`
    - access the key/value pair and move the pointer forward using `each`
    - reset the pointer and return the first value using `reset`
  - moving the pointer returns false when it cannot be moved further

# Sequential Access

- Using current and next

```
$cityVal = current($cities);
print("$cityVal<br />");
while ($cityVal = next($cities)){
      print("$cityVal<br />");
}
```

- Using each

```
while ($city = each($cities)){
      print("$city['value']<br />");
}
```

- Using foreach (version 1)

```
foreach ($cities as $cityVal){
      print("$cityVal<br />");
}
```

- Using foreach (version 2)

```
foreach ($cities as $cityKey=>$cityVal){
      print("$cityVal<br />");
}
```

# Stack Functions

- There are two functions that provide stack functionality:
  - array_push puts a new element at the end and returns the number of elements in the stack

```
$num = array_push ($stack, $element);
```

  - array_pop takes the top element off the stack and returns it (or NULL if the array is empty)

```
$element = array_pop($stack);
```

# Sorting Arrays

- There is a built-in sort function that is intended for use with traditional array data
  - sorts the data in descending order (letters before numbers)
  - replaces keys with incremental numerical values
  - sort does not return a value

```
sort($list);
```

- Two related sort functions
  - `asort` – sorts the values, maintaining the original key-value associations
  - `ksort` – sorts the keys instead of the values
- Reverse sort alternatives
  - `rsort, rasort, rksort`

# Functions

- The method for defining functions is typical of C-type languages

```
function name ([parameters]) {
    …
}
```

- General characteristics:
  - functions need not be defined before they are called
  - no function overloading
  - functions can have a variable number of parameters
  - function names are not case sensitive (for now)

# Function Parameters

- Variable number of parameters
  - more actual parameters than formal parameters
    - extra actual parameters are ignored
  - fewer actual parameters than formal parameters
    - missing formal parameters are unbound (=NULL)

- Default parameter passing is pass-by-value
- Pass-by-reference can be specified in the function definition by prepending an ampersand (&) to the parameter

```
function update(&$thing){
  $thing++;
}
```

# Variable Scope

- The default scope of a variable in a function is local
  - a variable defined in a function is only visible in the function
  - variables defined outside of a function are not visible by default
    - such global variables can be only accessed if they are declared global within the function

```
global $sum;
```

  - variables can also be specified as static

```
function doIt () {
    static $count = 0;
    $count++;
    echo “this function has been called $count times <br
    />”;
}
```

# Regular Expressions

- PHP supports the same Perl-based regular expressions that JavaScript does
- Two common functions:
  - preg_match
    - returns true if the regular expression matches the string

```
$found = preg_match("/^T/", $str);
```

  - preg_split
    - returns an array containing the contents of the string split on the pattern

```
$elements = preg_split("/\W/", $str);
```

# PHP Objects

```
class User {
   /* member variables */
   var $username;
   var $password;
   public $lastAccess;

   /* constructor */
   function __construct($u, $p){
      $this->username = $u;
      $this->password = $p;
     $this->lastAccess = Date(DATE_RFC2822);
   }

   /* member functions */
   function checkLogin ($u, $p) {
       if ($this->username == $u && $this->password == $p) {
           return true;
       } else {
           return false;
       }
   }
}
```

# Objects, Properties, & Functions

```
$user = new User;
$user2 = new User (“john”, “secretpass”);

$lastLogin = $user2->lastAccess;

if ($user2->checkLogin(“tim”, “hello”)) {
   // successful login
} else {
   // failed login
}
```

# Objects & Pass-By-Reference

- As you would expect, object assignment is pass-by-reference

```
class Test {
    public $name;
}
$obj1 = new Test();
$obj1->name = "John";
$obj2 = obj1;
$obj2->name = "Bill";
```

- If you want a complete copy, you must use the clone keyword

```
$obj2 = clone $obj1;
```

# Built-in Objects

- There are a number of built-in objects available to the PHP script
  - some of the important ones are
    - $_SERVER
    - $_POST
    - $_GET
    - $_FILES
    - $_COOKIES
    - $_SESSION
    - $_ENV
  - these are all “superglobals”: available in all scopes throughout a script

# Form Handling

- When a form is submitted, its contents are encoded and transmitted to the server
  - PHP implicitly decodes this data and makes it available in the $_POST, $_GET, and/or $_FILES variables
    - arrays of the variables from the form
    - which one depends on the method of the form submission (get or post), and if there was a file upload
    - the array subscripts are the names of the form elements
    - the values are the contents of the submitted forms

# The Power of the GET Method

- The HTTP GET method can be exploited without an HTML form
  - GET encodes the form elements in the URL
  - You can encode these yourself in a link

```
<a href="setvar.php?x=3&y=3&z=7">click me</a>
```

  - There is a limit on how much data you can send this way, as well as the format
    - the URL cannot include spaces or other reserved characters
    - if your data includes things like ? or &, these will need to be encoded too
  - These variables can then be extracted using the $_GET superglobal in PHP

```
$x = $_GET["x"];
$y = $_GET["y"];
$z = $_GET["z"];
```

# Session Management

- The HTTP protocol is stateless
- But there are times where keeping track of state information is beneficial
  - previously browsed locations on a site
  - confirmation of prior login
  - contents of shopping baskets

- There are two mechanisms we can use for managing state information with PHP:
  - cookies
  - session variables

# Cookies

- A cookie is a name-value pair that is passed between the browser and server
  - cookies are sent as part of the HTTP header
  - present in both requests and responses
  - can be created/overwritten both on the client and server
  - the browser will only send the cookie to the server that created the cookie (or the server that the browser was talking to when it created the cookie)
  - data is stored on the browser, which may pose a security risk

# Cookies & PHP

- Creating a cookie
  - the setcookie function takes the name, value, and lifetime as parameters

```
setcookie("voted", "true", time() + 864000); // 10 days
```

  - cookies must be created before any content is sent to the browser
- Reading a cookie
  - the value of the cookies can be read from the `$_COOKIE` superglobal
  - it is a good idea to test for the existence of the cookie before using it (using the `isset` function)

# Session Variables

- PHP includes built-in support for managing session variables
  - a session is the time span during which a browser interacts with a particular server
  - when a session is started, PHP creates a session ID
  - only this session ID is stored on the browser (via a cookie)
  - all of the session data is stored on the server, making this method more secure
  - if cookies are disabled, there is a mechanism for passing this session ID as a GET parameter, but then the session can be moved from one computer to another (which may be a problem)

# PHP and Session Variables

- In order to use the session variables, the session has to explicitly be started

```
session_start();
```

- After this, the `$_SESSION` superglobal can be used
  - associative array
  - can add as many variables as needed
  - session variables can be destroyed using `unset`
- If the session is no longer needed, it can be explicitly destroyed

```
session_destroy();
```

- The lifetime of the session (the cookie that holds the session ID) is specified in the PHP configuration (default: 1440 seconds)

# Checking the PHP Configuration

- There is a simple function built into PHP that will output all of the configuration settings for the server:

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

- If you put this in a page on its own, you can use it to easily check to see if a particular feature is enabled on the server

# PHP Site Architecture

- As with any static website, we should give some thought to the maintainability of a PHP site
  - separate style (CSS) from content (HTML)
  - separate code (JavaScript) from content using implicit embedding (<script> tag links to external file)
  - whenever possible, we also want to separate the PHP code from the HTML content
    - put functions in one or more separate files and use the include function to add these to the pages that need them
    - do all of the data processing in one place (beginning of file) and then insert the results using the shorthand tag (<?=$var?>)
    - consider using a template HTML file that is also separated from the data processing PHP file
  - general philosophy: maintainability

# Homework

- Read Chapter 10 & 11
- Next topic: MySQL & PHP
- Upcoming deadlines:
  - Assignment 4: Nov 21 @ 11:55 PM