# Introduction to PHP

## PHP

- Server-side scripting language useful for writing CGI
- Began as online tools by Rasmus Lerfdorf
  - PHP = Personal Home Page tools
  - Bottom-up development, has become extremely popular today
    - Somewhat like C but much higher level, OOP model was added later
    - Especially with Apache/Linux/MySQL
    - Runs on both Unix and Windows platforms, with most web servers
    - Used today with many commercial sites
- Available for free
  - http://www.php.net
  - Documentation and many resources available online
    - I prefer the online documentation, easy to search (several Mb as a single large HTML file)

# Simple PHP Script

- Consider the following HTML file, example.html:

```
<html>
<head>
<title>My Page</title>
</head>
<body>
<p>Hello world!</p>
</body>
</html>
```

# Simple PHP Script

- Here is an equivalent PHP script. PHP files have the extension ".php" and may contain both HTML and PHP code, which is enclosed inside <?php code ?> tags, or alternately <? code ?> (if supported)

```
<html>
<head>
<title>My Page</title>
</head>
<body>
<?php
  print("<p>hello world!</p>");
?>
</body>
</html>
```

# Simple PHP Script

- More interesting version, displays the date as known by the server:

```
<html>
<head>
<title>My Page</title>
</head>
<body>
<?php
   print("<p>hello world!  Timestamp: " . time() . "</p>");
?>
</body>
</html>
```

Could also use "echo" and () not required

# PHP Time Stamp

- The "." is used to concatenate strings
- The server parses the document and interprets code within the <?php ?> tags instead of sending it directly to the client
  - i.e. you can write code to output the HTML you desire
- Output of previous:

hello world! Timestamp: 1050289182

Refresh two Seconds later: hello world! Timestamp: 1050289184

# PHP Script

- Sometimes everything is placed inside the PHP tags. The following is equivalent; the header function specifies the MIME Type; i.e. that the document is HTML (as opposed to graphics, etc.):

```
<?php
  header("Content-Type: text/html");
  print("<HTML><HEAD><TITLE>My Page</TITLE>");
  print("</HEAD>");
  print("<BODY>");
  print("hello world! Timestamp: " . time() . "<p>");
  print("</BODY></HTML>");
?>
```

# Identifiers and Data Types

- Identifiers
  - Case-sensitive
  - Same rules as Java
- Data Types
  - booleans
  - integer
  - double
  - string, surrounded by “ “ or by ‘ ‘
  - Weak typing; you do not declare variables, just use them and the value assigned is the type of the variable; any old value is gone
  - Can typecast just like Java
    - (int), (double), (string), etc.

# Variables

- A variable is an identifier prefaced by $
- Example:

```
$x = 1;
$y = 3.4;
$z = $x + $y;
$a = true;
$s = "hello!";
print ($z . " " . $a . " " . $s);
print "$z $a $s some other text here";
```

Output: 4.4 1 hello!
Note: true = non zero or not empty. False = 0 or the empty string “”

Common novice mistake: Forgetting the $

# Variables

- Interpreted; consider the following:

```
$x = 1;
$y = “x”;
print($$y);
```

Output: 1

- Often {} are used to denote variable boundaries:

```
$x = 1;
$y = “x”;
print(${$y});
```

# Form Variables

- If an HTML form invokes a PHP script, the PHP script can access all of the form variables by name
- Invoking FORM:

```
<form method=post action="scr.php">
<input type=text name="foo" value="bar">
<input type=submit value="Submit">
</form>
```

- Inside scr.php:

```
print($_REQUEST['foo']);                    // Outputs "bar"
```

# Sample PHP Form

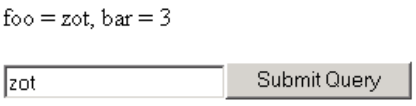
```
<?php
 header("Content-Type: text/html");
 print("<HTML><HEAD><TITLE>My Page</TITLE>");
 print("</HEAD>");
 print("<BODY>");
 print("foo = " . $_REQUEST['foo'] . ", bar = " .
        $_REQUEST['bar'] .  "<P>");
 print("<form method=post action=\"example.php\">");
 print("<input type=text name=\"foo\" value=\"zot\">");
 print("<input type=hidden name=\"bar\" value=3>");
 print("<input type=submit>");
 print("</form>");
 print("</BODY></HTML>");
?>
```

Note: \" escape character
Could also use ' instead

# Sample PHP Form

- First load:


- Upon submit:




# Webbrowser

- What the web browser receives after the first load. Note that we see no PHP code:

```
<HTML>
<HEAD><TITLE>My Page</TITLE>
</HEAD>
<BODY>foo = , bar =  <P>
<form method=post action="example.php">
<input type=text name="foo" value="zot">
<input type=hidden name="bar" value=3>
<input type=submit></form></BODY></HTML>
```

# Accessing Unset Variables

- Depending upon the configuration of PHP, you may or may not get error messages when trying to access variables that have not been set
- Can avoid this issue using isset:

```
if (isset($_REQUEST[‘foo’], $_REQUEST[‘bar’]))
{
  print("foo = " . $_REQUEST[‘foo’] . ", bar = " .
        $_REQUEST[‘bar’] .  "<P>");
}
```

# GET and POST

- Another way to hide the printing of variables when the code is first loaded is to detect if the program is invoked via GET or POST

```
<?php
 header("Content-Type: text/html");
 print("<HTML><HEAD><TITLE>My Page</TITLE>");
 print("</HEAD>");   print("<BODY>");
 if ($_SERVER['REQUEST_METHOD'] == ‘POST') {
   print("foo = " . $_REQUEST[‘foo’] . ", bar = " .
         $_REQUEST[‘bar’] .  "<P>");
 }
 print("<form method=post action=\"example.php\">");
 print("<input type=text name=\"foo\" value=\"zot\">");
 print("<input type=hidden name=\"bar\" value=3>");
 print("<input type=submit>");
 print("</form>");
 print("</BODY></HTML>");
?>
```

# Operators

- Same operators available as in Java:

  +, -, *, /, %, ++, --  (both pre/post)

  +=, -=, *=, etc.

  <, >, <=, >=, ==, !=, &&, ||, XOR, !

- Some new ones

  ===  Identical; same value and type

  !==  Not identical; not same value or type

# Assignments

- PHP will convert types for you to make assignments work
- Examples:

```
print(1 + "2");          // 3

print("3x" + 10.5);      // 13.5

$s = "hello" . 55;
print("$s<p>");          // hello55
```

# Arrays

- Arrays in PHP are more like hash tables, i.e. associative arrays
  - The key doesn't have to be an integer

- 1D arrays
  - Use [] to access each element, starting at 0
  - Ex:

```
$arr[0] = "hello";
$arr[1] = "there";
$arr[2] = "zot";
$i=0;
print("$arr[$i] whats up!<p>");    // Outputs  :  hello whats up!
```

# Arrays

- Often we just want to add data to the end of the array, we can do so by entering nothing in the brackets:

```
$arr[] = "hello";
$arr[] = "there";
$arr[] = "zot";
print("$arr[2]!<p>");    // Outputs  :  zot!
```

# Array Functions

- See the text or reference for a list of array functions; here are just a few:

```
count($arr);            // Returns # items in the array
sort($arr);             // Sorts array
array_unique($arr);     // Returns $arr without duplicates

print_r($var);          // Prints contents of a variable
                        // useful for outputting an entire array
                        // as HTML
in_array($val, $arr)    // Returns true if $val in $arr
```

# Multi-Dimensional Arrays

- To make multi-dimensional arrays just add more brackets:

```
$arr[0][0]=1;
$arr[0][1]=2;
$arr[1][0]=3;
..etc.
```

# Arrays with Strings as Key

- So far we've only seen arrays used with integers as the index
- PHP also allows us to use strings as the index, making the array more like a hash table
- Example:

```
$fat["big mac"] = 34;
$fat["quarter pounder"]=48;
$fat["filet o fish"]=26;
$fat["large fries"]=26;
print("Large fries have " . $fat["large fries"] . " grams of fat.");

// Output : Large fries have 26 grams of fat
```

Source: www.mcdonalds.com

# Iterating through Arrays with foreach

- PHP provides an easy way to iterate over an array with the foreach clause:
- Format: foreach ($arr as $key=>$value) { … }
- Previous example:

```
foreach($fat as $key=>$value)
{
    print("$key has $value grams of fat.<br/>");
}
```

Output:
big mac has 34 grams of fat.
quarter pounder has 48 grams of fat.
filet o fish has 26 grams of fat.
large fries has 26 grams of fat.

# Foreach

- Can use foreach on integer indices too:

```
$arr[]="foo";
$arr[]="bar";
$arr[]="zot";
foreach ($arr as $key=>$value)
{
    print("at $key the value is $value<br>");
}
```

Output:
at 0 the value is foo
at 1 the value is bar
at 2 the value is zot

If only want the value,
can ignore the $key variable

# Control Statements

- In addition to foreach, we have available our typical control statements
  - If
  - While
  - Break/continue
  - Do-while
  - For loop

# IF statement

- Format:

```
if (expression1)
{
        // Executed if expression1 true
}
elseif (expression2)
{
        // Executed if expression1 false expresson2 true
}
…
else
{
        // Executed if above expressions false
}
```

# While Loop

- Format:

```
while (expression)
{
    // executed as long as expression true
}
```

# Do-While

- Format:

```
do
{
    // executed as long as expression true
    // always executed at least once
}
while (expression);
```

# For Loop

- Format:

```
for (initialization; expression; increment)
{
    // Executed as long as expression true
}
```

# Control Example

Counts # of random numbers generated between 0-10

```
srand(time());        // Seed random # generator with time

for ($i=0; $i<100; $i++) {
    $arr[]=rand(0,10);      // Random number 0-10, inclusive
}
$i=0;
while ($i<=10) {      // Initialize array of counters to 0
    $count[$i++]=0;
}
// Count the number of times we see each value
foreach ($arr as $key=>$value) {
    $count[$value]++;
}
// Output results
foreach ($count as $key=>$value) {
    print("$key appeared $value times.<br>");
}
```

# Output

0 appeared 9 times.
1 appeared 9 times.
2 appeared 11 times.
3 appeared 14 times.
4 appeared 6 times.
5 appeared 7 times.
6 appeared 8 times.
7 appeared 11 times.
8 appeared 5 times.
9 appeared 9 times.
10 appeared 11 times.

# Functions

- To declare a function:

```
function function_name(arg1, arg2, …)
{
        // Code
        // Optional:   return (value);
 }
```

Unlike most languages, no need for a return type since PHP is weakly typed

# Function Example: Factorial

```
function fact($n)
{
   if ($n <= 1) return 1;
   return ($n * fact($n-1));
}

print(fact(5));                    // Outputs 120
```

# Scoping

- Variables defined in a function are local to that function only and by default variables are pass by value

```
function foo($x,$y)
{
   $z=1;
   $x=$y + $z;
   print($x);                          // Outputs 21
}

$x=10;
$y=20;
foo($x,$y);
print("$x $y<p>");                      // Outputs 10   20
```

# Arrays: Also Pass By Value

- Arrays also are passed by value!

```
function foo($x)
{
  $x[0]=10;
  print_r($x);                         Array ( [0] => 10 [1] => 2 [2] => 3 )
  print("<p>");
}

 $x[0]=1;
 $x[1]=2;
 $x[2]=3;
 print_r($x);                         Array ( [0] => 1 [1] => 2 [2] => 3 )
 print("<p>");
 foo($x);
 print_r($x);                         Array ( [0] => 1 [1] => 2 [2] => 3 )
 print("<p>");                        Not changed!
```

# Pass by Reference

- To pass a parameter by reference, use & in the parameter list

```
function foo(&$x,$y)
{
   $z=1;
   $x=$y + $z;
   print($x);                    // Outputs 21
}

$x=10;
$y=20;
foo($x,$y);
print(“$x $y<p>”);               // Outputs 21   20
```

# Dynamic Functions

- Functions can be invoked dynamically too, like we can do in Scheme
  - Useful for passing a function as an argument to be invoked later

```
function foo()
{
  print("Hi<p>");
}

$x="foo";
$x();                    // Outputs “Hi”
```

# Classes & Objects

- PHP supports classes and inheritance
- PHP 5 allows public, private, protected (all instance variables are public in PHP 4)
- Format for defining a class; the extends portion is optional

```
class Name extends base-class
{
    public varName;
    …
    function __construct() {
      //… code for constructor …  public if not specified; name of class is old style
    }
    private function methodName() { … code … }
    …
}
```

- To access a variable or function, use $obj->var (no $ in front of the var)
- To access instance variables inside the class, use $this->var
  needed to differentiate between instance var and a new local var

# Class Example

```
class User
{
  public $name;
  public $password;
  public function __construct($n, $p)
  {
     $this->name=$n;
     $this->password=$p;
  }
  public function getSalary()
  {
     // if this was real, we might
     // look this up in a database or something
     return 50000;
  }
}

$joe = new User("Joe Schmo","secret");
print($joe->name . " - " . $joe->password . "<p>");
print($joe->getSalary() . "<p>");
```

Output:

Joe Schmo - secret

50000

# Inheritance

- Operates like you would expect

```
class foo
{
   public function printItem($string)
   {
      echo 'Foo: ' . $string . '<P>';
   }

   public function printPHP()
   {
      echo 'PHP is great.' . '<P>';
   }
}
```

```
class bar extends foo
{
   public function printItem($string)
   {
      echo 'Bar: ' . $string . '<P>';
   }
}


$foo = new foo();
$bar = new bar();
$foo->printItem('baz'); // Output: 'Foo: baz'
$foo->printPHP();       // Output: 'PHP is great'
$bar->printItem('baz'); // Output: 'Bar: baz'
$bar->printPHP();       // Output: 'PHP is great'
```

# Dynamic Binding

```
class foo
{
   public function printItem($string)
   {
      echo 'Foo: ' . $string . '<P>';
   }
}

function myTest($o)
{
  print_r($o->printItem("mytest"));
}
```

```
class bar extends foo
{
   public function printItem($string)
   {
      echo 'Bar: ' . $string . '<P>';
   }
}


$foo = new foo();
$bar = new bar();
myTest($foo);          // Output: Foo: mytest
myTest($bar);          // Output: Bar: mytest
```

# Static Variables

```
class User
 {
   public static $masterPassword;

   public function foo()
   {
      print self::$masterPassword;
   }
 }


User::$masterPassword = “abc123”;
```

# Destructors

- Called when there are no more references to the object

```
function __destruct() {
    print "Destroying " . $this->name . "\n";
}
```

# Abstract Classes

- Abstract classes cannot be instantiated; abstract methods cannot be implemented in a subclass

```
abstract class AbstractClass
{
   // Force Extending class to define this method
   abstract protected function getValue();
   abstract protected function prefixValue($prefix);

   // Common method
   public function printOut() {
      print $this->getValue() . "\n";
   }
}
```

```
class ConcreteClass1 extends AbstractClass
{
   protected function getValue() {
      return "ConcreteClass1";
   }

   public function prefixValue($prefix) {
      return "{$prefix}ConcreteClass1";
   }
}
```

# Interfaces

- Could have made an interface if we leave out the common method

```
interface iFoo
{
   public function getValue();
   public function prefixValue($prefix);
}
```

```
class ConcreteClass1 implements iFoo
{
   public function getValue() {
      return "ConcreteClass1";
   }

   public function prefixValue($prefix) {
      return "{$prefix}ConcreteClass1";
   }
}
```

# Objects in PHP 5

- Assigning an object makes a reference to the existing object, like Java:

```
$joe = new User("Joe Schmo","secret");
$fred = $joe;
$joe->password = "a4j1%";
print_r($joe);          // user Object ( [name] => Joe Schmo [password] => a4j1% )
print("<p>");
print_r($fred);         // user Object ( [name] => Joe Schmo [password] => a4j1% )
print("<p>");
```

In PHP 4 it assigned a copy instead; use clone in PHP 5

# Other items in PHP

- Many others, but just a few… OOP features added in PHP 5
  - Final
  - instanceof
  - Reflection
  - Namespaces
  - Serialization