CSC 498: Web Programming

Haidar Harmanani
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010 Lebanon

Server-Side Alternatives

- CGI is the most common approach to server-side programming
  - Universal support: (almost) Every server supports CGI programming. A great deal of ready-to-use CGI code.
    - Most APIs (Application Programming Interface) also allow CGI programming.
  - Choice of languages: CGI is extremely general, so that programs may be written in nearly any language.
    - Perl is by far the most popular, with the result that many people think that CGI means Perl.
    - C, C++, Ruby or Python are also fine.
- CGI Drawbacks: A separate process is run every time the script is requested. A distinction is made between HTML pages and code.

Server-Side Alternatives

- Server-Side Includes (SSI):
  - Code is embedded in HTML pages, and evaluated on the server while the pages are being served. Add dynamically generated content to an existing HTML page, without having to serve the entire page via a CGI program.
- Active Server Pages (ASP, Microsoft) :
  - The ASP engine is integrated into the web server so it does not require an additional process. It allows programmers to mix code within HTML pages instead of writing separate programs. (Drawback(?) Must be run on a server using Microsoft server software.)
- Java Servlets (Sun):
  - As CGI scripts, they are code that creates documents. Must be compiled as classes which are dynamically loaded by the web server when they are run.
- Java Server Pages (JSP):
  - Like ASP, another technology that allows developers to embed Java in web pages

PHP

- developed in 1995 by Rasmus Lerdorf (member of the Apache Group)
  - originally designed as a tool for tracking visitors at Lerdorf's Web site
  - within 2 years, widely used in conjunction with the Apache server
  - developed into full-featured, scripting language for server-side programming
  - free, open-source
  - server plug-ins exist for various servers
  - fully integrated to work with mySQL databases

Introduction

- PHP
  - PHP: Hypertext Preprocessor
  - Originally called "Personal Home Page Tools"
  - Popular server-side scripting technology
  - Open-source
    - Anyone may view, modify and redistribute source code
    - Supported freely by community
  - Platform independent

5

PHP

- Basic application
  - Scripting delimiters
    - `<? php ?>`
    - Must enclose all script code
  - Variables preceded by `$` symbol
    - Case-sensitive
  - End statements with semicolon
  - Comments
    - `//` for single line
    - `/* */` for multiline
  - Filenames end with `.php` by convention

6

PHP

- PHP is similar to JavaScript, only it's a server-side language
  - PHP code is embedded in HTML using tags
  - when a page request arrives, the server recognizes PHP content via the file extension (.php or .phtml)
  - the server executes the PHP code, substitutes output into the HTML
  - the resulting page is then downloaded to the client
  - user never sees the PHP code, only the output in the page
- The acronym PHP means (in a slightly recursive definition)
  - PHP: Hypertext Preprocessor

What do You Need?

- Our server supports PHP
  - You don't need to do anything special!
  - You don't need to compile anything or install any extra tools!
  - Create some .php files in your web directory - and the server will parse them for you.
- Most servers support PHP
  - Download PHP for free here: http://www.php.net/downloads.php
  - Download MySQL for free here: http://www.mysql.com/downloads/index.html
  - Download Apache for free here: http://httpd.apache.org/download.cgi

Basic PHP syntax

A PHP scripting block always starts with <?php and ends with ?>. A PHP scripting block can be placed anywhere in an HTML document.

```
<html>
<!-- hello.php COMP519 -->
<head><title>Hello World</title></head>
<body>
 <p>This is going to be ignored by the PHP interpreter.</p>

 <?php echo '<p>While this is going to be parsed.</p>'; ?>

 <p>This will also be ignored by PHP.</p>

 <?php print('<p>Hello and welcome to <i>my</i> page!</p>');
  ?>

 <?php
  //This is a comment
 /*
  This is
  a comment
  block
 */
  ?>
</body>
</html>
```

view the output page

print and echo for output

a semicolon (;) at the end of each statement (can be omitted at the end of block/ file)

// for a single-line comment

/* and */ for a large comment block.

The server executes the print and echo statements, substitutes output.

PHP

- Variables
  - Can have different types at different times
  - Variable names inside strings replaced by their value
  - Type conversions
    - `settype` function
    - Type casting
  - Concatenation operator
    - . (period)
    - Combine strings

Scalars

All variables in PHP start with a $ sign symbol.
A variable's type is determined by the context in which that variable is used (i.e. there is no strong-typing in PHP).

```
<html><head></head>
<!-- scalars.php COMP519 -->
<body>  <p>
<?php
$foo = True; if ($foo) echo "It is TRUE! <br /> \n";
$txt='1234'; echo "$txt <br /> \n";
$a = 1234; echo "$a <br /> \n";
$a = -123;
echo "$a <br /> \n";
$a = 1.234;
echo "$a <br /> \n";
$a = 1.2e3;
echo "$a <br /> \n";
$a = 7E-10;
echo "$a <br /> \n";
echo 'Arnold once said: "I\'ll be back"', "<br /> \n";
$beer = 'Heineken';
echo "$beer's taste is great <br /> \n";
$str = <<<EOD
Example of string
spanning multiple lines
using "heredoc" syntax.
EOD;
echo $str;
?>  </p>
</body>
</html>
```

Four scalar types:
boolean
 TRUE or FALSE
integer,
 just numbers
float
 float point numbers
string
 single quoted
 double quoted

view the output page

PHP

Data type	Description
`int, integer`	Whole numbers (i.e., numbers without a decimal point).
`float, double`	Real numbers (i.e., numbers containing a decimal point).
`string`	Text enclosed in either single (' ') or double (" ") quotes.
`bool, Boolean`	True or false.
`array`	Group of elements of the same type.
`object`	Group of associated data and methods.
`Resource`	An external data source.
`NULL`	No value.

Fig. 26.2 PHP data types.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4  <!-- Fig. 26.3: data.php             -->
5  <!-- Demonstration of PHP data types -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>PHP data types</title>
10    </head>
11
12    <body>
13
14       <?php
15
16          // declare a string, double and integer
17          $testString = "3.5 seconds";
18          $testDouble = 79.2;
19          $testInteger = 12;
20       ?>
21
```

Assign a string to variable
$testString

Assign an integer to variable
$testInteger

Arrays

An array in PHP is actually an ordered map. A map is a type that maps values to keys.

```
<?php
$arr = array("foo" => "bar", 12 => true);
echo $arr["foo"]; // bar
echo $arr[12];    // 1
?>
```

array() = creates arrays

key = either an integer or a string.

value = any PHP type.

```
<?php
array(5 => 43, 32, 56, "b" => 12);
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

if no key, the maximum of the integer indices + 1.

if an existing key, its value will be overwritten.

```
<?php
$arr = array(5 => 1, 12 => 2);
$arr[] = 56;    // the same as $arr[13] = 56;
$arr["x"] = 42; // adds a new element
unset($arr[5]); // removes the element
unset($arr);    // deletes the whole array
$a = array(1 => 'one', 2 => 'two', 3 => 'three');
unset($a[2]);
$b = array_values($a);
?>
```

view the output page

can set values in an array

unset() removes a key/ value pair

array_values() makes reindex effect (indexing numerically)

*Find more on arrays

Constants

A constant is an identifier (name) for a simple value. A constant is case-sensitive by default. By convention, constant identifiers are always uppercase.

```
<?php

// Valid constant names
define("FOO",    "something");
define("FOO2",   "something else");
define("FOO_BAR", "something more");

// Invalid constant names
define("2FOO",   "something");

// This is valid, but should be avoided:
// PHP may one day provide a "magical" constant
// that will break your script
define("__FOO__", "something");

?>
```

You can access constants anywhere in your script without regard to scope.

Operators

- Arithmetic Operators: +, -, *,/ , %, ++, --
- Assignment Operators: =, +=, -=, *=, /=, %=

Example	Is the same as
x+=y	x=x +y
x-=y	x=x-y
x*=y	x=x*y
x/=y	x=x/y
x%=y	x=x%y

- Comparison Operators: ==, !=, >, <, >=, <=
- Logical Operators: &&, ||, !
- String Operators: . , .=

```
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"

$a = "Hello ";
$a .= "World!";
```

Conditionals: if else

- Can execute a set of code depending on a condition

```
<html><head></head>
<!-- if-cond.php COMP519 -->
<body>
<?php
$d=date("D");
if ($d=="Fri")
    echo "Have a nice weekend! <br/>";
else
    echo "Have a nice day! <br/>";
$x=10;
if ($x==10)
{
    echo "Hello<br />";
    echo "Good morning<br />";
}
?>
</body>
</html>
```

```
if (condition)
code to be executed if condition is
true;
else
code to be executed if condition is
false;
```

date() is a built-in function that can be called with many different parameters to return the date (and/or local time) in various formats

In this case we get a three letter string for the day of the week.

view the output page

Conditionals: switch

- Can select one of many sets of lines to execute

```
<html><head></head>
<body>
<!-- switch-cond.php COMP519 -->
<?php
$x = rand(1,5);  // random integer
echo "x = $x <br/><br/>";
switch ($x)
{
case 1:
 echo "Number 1";
 break;
case 2:
 echo "Number 2";
 break;
case 3:
 echo "Number 3";
 break;
default:
 echo "No number between 1 and 3";
}
?>
</body>
</html>
```

```
switch (expression)
{
case label1:
  code to be executed if expression =
label1;
  break;
case label2:
  code to be executed if expression =
label2;
  break;
default:
  code to be executed
  if expression is different
  from both label1 and label2;
}
```

view the output page

Looping: while and do-while

Can loop depending on a condition

```
<html><head></head>
<body>
<?php
$i=1;
while($i <= 5)
{
  echo "The number is $i <br />";
  $i++;
}
?>
</body>
</html>
```

loops through a block of code if and as long as a specified condition is true

view the output page

```
<html><head></head>
<body>
<?php
$i=0;
do
{
  $i++;
  echo "The number is $i <br />";
}
while($i <= 10);
?>
</body>
</html>
```

loops through a block of code once, and then repeats the loop as long as a special condition is true

view the output page

Looping: for and foreach

Can loop depending on a "counter"

```
<?php
for ($i=1; $i<=5; $i++)
{
echo "Hello World!<br />";
}
?>
```

loops through a block of code a specified number of times

view the output page

```
<?php
$a_array = array(1, 2, 3, 4);
foreach ($a_array as $value)
{
   $value = $value * 2;
   echo "$value <br/> \n";
}
?>
```

```
<?php
$a_array=array("a","b","c");
foreach ($a_array as $key=>$value)
{
  echo $key." = ".$value."\n";
}
?>
```

loops through a block of code for each element in an array

User Defined Functions

Can define a function using syntax such as the following:

```
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
   echo "Example function.\n";
   return $retval;
}
?>
```

Can also define conditional functions, functions within functions, and recursive functions.

Can return a value of any type

```
<?php
function square($num)
{
   return $num * $num;
}
echo square(4);
?>
```

```
<?php
function small_numbers()
{
   return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
echo $zero, $one, $two;
?>
```

```
<?php
function takes_array($input)
{
   echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
 takes_array(array(1,2));
?>
```

view the output page

Variable Scope

The scope of a variable is the context within which it is defined.

```
<?php
$a = 1; /* global scope */
function Test()
{
   echo $a;
  /* reference to local scope variable */
}
Test();
?>
```

The scope is local within functions.

```
<?php
$a = 1;
$b = 2;
function Sum()
{
   global $a, $b;
   $b = $a + $b;
}
Sum();
echo $b;
?>
```

global

refers to its global version.

view the output page

```
<?php
function Test()
{
   static $a = 0;
   echo $a;
   $a++;
}
Test1();
Test1();
Test1();
?>
```

static

does not lose its valu

Including Files

The *include()* statement includes and evaluates the specified file.

```
vars.php
<?php

$color = 'green';
$fruit = 'apple';

?>
test.php
<?php

echo "A $color $fruit"; // A

include 'vars.php';

echo "A $color $fruit"; // A green apple

?>
```

view the output page

```
<?php

function foo()
{
   global $color;

   include ('vars.php');

   echo "A $color $fruit";
}

/* vars.php is in the scope of foo() so     *
 * $fruit is NOT available outside of this  *
 * scope.  $color is because we declared it *
 * as global.                               */

foo();                  // A green apple
echo "A $color $fruit";  // A green

?>
```

view the output page

The scope of variables in “included” files depends on where the “include” file is added!

You can use the include_once, require, and require_once statements in similar ways.

PHP Information

The *phpinfo()* function is used to output PHP information about the version installed on the server, parameters selected when installed, etc.

```
<html><head></head>
<!– info.php COMP519
<body>
<?php
// Show all PHP information
phpinfo();
?>
<?php
// Show only the general information
phpinfo(INFO_GENERAL);
?>
</body>
</html>
```

view the output page

INFO_GENERAL	The configuration line, php.ini location, build date, Web Server, System and more
INFO_CREDITS	PHP 4 credits
INFO_CONFIGURATION	Local and master values for php directives
INFO_MODULES	Loaded modules
INFO_ENVIRONMENT	Environment variable information
INFO_VARIABLES	All predefined variables from EGPCS
INFO_LICENSE	PHP license information

Server Variables

The $_SERVER is a reserved variable that contains all server information.

```
<html><head></head>
<body>

<?php
echo "Referer: " . $_SERVER["HTTP_REFERER"] . "<br />";
echo "Browser: " . $_SERVER["HTTP_USER_AGENT"] . "<br />";
echo "User's IP address: " . $_SERVER["REMOTE_ADDR"];
?>
</body>
</html>
```

view the output page

The $_SERVER is a super global variable, i.e. it's available in all scopes of a PHP script.

File Open

The `fopen("file_name","mode")` function is used to open files in PHP.

r	Read only.	r+	Read/Write.
w	Write only.	w+	Read/Write.
a	Append.	a+	Read/Append.
x	Create and open for write only.	x+	Create and open for read/write.

```
<?php
$fh=fopen("welcome.txt","r");
?>
```

For w, and a, if no file exists, it tries to create it (use with caution, i.e. check that this is the case, otherwise you'll overwrite an existing file).

For x if a file exists, it returns an error.

```
<?php
if (!($fh=fopen("welcome.txt","r")))
exit("Unable to open file!");
?>
```

If the fopen() function is unable to open the specified file, it returns 0 (false).

File Workings

fclose() **closes a file.**
fgetc() **reads a single character**
fwrite(), fputs **writes a string with and without \n**

```
<?php
$myFile = "welcome.txt";
if (!($fh=fopen($myFile,'r')))
exit("Unable to open file.");
while (!feof($fh))
{
$x=fgetc($fh);
echo $x;
}
fclose($fh);
?>
```

view the output page

```
<?php
$lines = file('welcome.txt');
foreach ($lines as $l_num => $line)
{
 echo "Line #{$l_num}:“ .$line.”<br/>”;
}
?>
```

view the output page

feof() **determines if the end is true.**

fgets() **reads a line of data**

file() **reads entire file into an array**

```
<?php
$myFile = "welcome.txt";
$fh = fopen($myFile, 'r');
$theData = fgets($fh);
fclose($fh);
echo $theData;
?>
```

view the output page

```
<?php
$myFile = "testFile.txt";
$fh = fopen($myFile, 'a') or die("can't open file");
$stringData = "New Stuff 1\n";
fwrite($fh, $stringData);
$stringData = "New Stuff 2\n";
fwrite($fh, $stringData);
fclose($fh);
?>
```

view the output page

Form Handling

Any form element is automatically available via one of the built-in PHP variables.

```
<html>
<-- form.html COMP519 -->
<body>
<form action="welcome.php" method="POST">
Enter your name: <input type="text" name="name" /> <br/>
Enter your age: <input type="text" name="age" /> <br/>
<input type="submit" /> <input type="reset" />
</form>
</body>
</html>
```

```
<html>
<!-- welcome.php COMP 519 -->
<body>

Welcome <?php echo $_POST["name"].”.”; ?><br />
You are <?php echo $_POST["age"]; ?> years old!

</body>
</html>
```

$_POST
contains all POST data.

$_GET
contains all GET data.

view the output page

Cookie Workings

`setcookie(name,value,expire,path,domain)` **creates cookies.**

```
<?php
setcookie("uname", $_POST["name"], time()+36000);
?>
<html>
<body>
<p>
Dear <?php echo $_POST["name"] ?>, a cookie was set on this
page! The cookie will be active when the client has sent the
cookie back to the server.
</p>
</body>
</html>
```

view the output page

NOTE:
setcookie() must appear BEFORE <html> (or any output) as it's part of the header information sent with the page.

```
<html>
<body>
<?php
if (isset($_COOKIE["uname"]))
echo "Welcome " . $_COOKIE["uname"] . "!<br />";
else
echo "You are not logged in!<br />";
?>
</body>
</html>
```

view the output page

$_COOKIE
contains all COOKIE data.

isset()
finds out if a cookie is set

use the cookie name as a variable

Getting Time and Date

`date() and time ()` **formats a time or a date.**

```
<?php
//Prints something like: Monday
echo date("l");

//Like: Monday 15th of January 2003 05:51:38 AM
echo date("l dS of F Y h:i:s A");

//Like: Monday the 15th
echo date("l \\t\h\e jS");
?>
```

view the output page

date() returns a string formatted according to the specified format.

```
<?php
$nextWeek = time() + (7 * 24 * 60 * 60);
            // 7 days; 24 hours; 60 mins; 60secs
echo 'Now:      '. date('Y-m-d') ."\n";
echo 'Next Week: '. date('Y-m-d', $nextWeek) ."\n";
?>
```

view the output page

time() returns current Unix timestamp

*Here is more on date/time formats: http://uk.php.net/manual/en/function.date.php

Required Fields in User-Entered Data

A multipurpose script which asks users for some basic contact information and then checks to see that the required fields have been entered.

```
<html>
<!-- form_checker.php COMP519 -->
<head>
<title></title>
</head>
<body>
<?php
/*declare some functions*/
```

Print Function

```
function print_form($f_name, $l_name, $email, $os)
{
?>

 <form action="form_checker.php" method="POST">
 First Name: <input type="text" name="f_name" value="<?php echo $f_name?>" /> <br/>
 Last Name <b>*</b>:<input type="text" name="l_name" value="<?php echo $l_name?>" /> <br/>
 Email Address <b>*</b>:<input type="text" name="email" value="<?php echo $email?>" /> <br/>
 Operating System: <input type="text" name="os" value="<?php echo $os?>" /> <br/><br/>
 <input type="submit" name="submit" value="Submit" /> <input type="reset" />
 </form>

<?php
}
```


Check and Confirm Functions

```
function check_form($f_name, $l_name, $email, $os)
{
  if (!$l_name||!$email){
  echo "<h3>You are missing some required fields!</h3>";
  print_form($f_name, $l_name, $email, $os);
  }
  else{
   confirm_form($f_name, $l_name, $email, $os);
  }
}
```

```
function confirm_form($f_name, $l_name, $email, $os)
{
?>

<h2>Thanks! Below is the information you have sent to us.</h2>
<h3>Contact Info</h3>

<?php
echo "Name: $f_name $l_name <br/>";
echo "Email: $email <br/>";
echo "OS: $os";
}
```

Main Program

```
/*Main Program*/

if (!$_POST["submit"])
{
?>

 <h3>Please enter your infromation</h3>
 <p>Fields with a "<b>*</b>" are required.</p>

<?php
 print_form("","","","");
}
else{
 check_form($_POST["f_name"],$_POST["l_name"],$_POST["email"],$_POST["os"]);
}
?>


</body>
</html>
```

view the output page

More on PHP

- Objects
- Defining objects
- Inheritance
- Sessions and session variables

Object oriented programming in PHP

- PHP, like most modern programming languages (C++, Java, Perl, JavaScript, etc.), supports the creation of objects.
- Creating an object requires you to first define an object class (containing variables and/or function definitions) and then using the "new" keyword to create an instance of the object class. (Note that the object must be defined before you instantiate it.)

Object oriented programming in PHP

```
<?php
//  Assume that the "Person" object has been previously
  defined. . .

$x = new Person;  //  creates an instance of the Person class
  (*no* quotes)

//  The object type need not be "hardcoded" into the declaration.

$object_type = 'Person';
$y = new $object_type;  //  equivalent to $y = new Person;

$z = new Vehicle('Jaguar','green');  //  creating an object and
  passing
                                     // arguments to its constructor
?>
```

Declaring a class

- Use the "class" keyword which includes the class name (case-insensitive, but otherwise following the rules for PHP identifiers). Note: The name "stdClass" is reserved for use by the PHP interpreter.

```
<?php
class Person
{
   var $name;

    function set_name($new_name)  {
      $name = $this -> new_name;
   }

    function get_name() {
      return $this -> name;
   }
}
?>
```

Declaring a class (cont.)

- Properties and functions can be declared as "public" (accessible outside the object's scope), "private" (accessible only by methods within the same class), or "protected" (accessible only through the class methods and the class methods of classes inheriting from the class.
- Note that unless a property is going to be explicitly declared as public, private, or protected, it need not be declared before being used (like regular PHP variables).

```
<?php
class Person
{
   protected $name;
   protected $age;

   function set_name($new_name)  {
      $name = $this -> new_name;
   }
   function get_name() {
      return $this -> name;
   }
```

Declaring a class (cont.)

- Classes can also have their own constants defined (using the "const" keyword), can have their own static properties and functions (using the keyword "static" before "var" or "function"), and can also can constructors and destructors (see below).
- Static properties and functions are accessed (see below) using a different format than usual for objects, and static functions cannot access the objects properties (i.e. the variable $this is not defined inside of a static function).

```
<?php
class HTMLtable {
   static function start() {
     echo "<table> \n";
   }
   static function end() {
     echo "</table> \n";
   }
}
HTMLtable::start();
?>
```

Accessing properties and methods

- Once you have an object, you access methods and properties (variables) of the object using the -> notation.

```
<?php
$me = new Person;
$me -> set_name('Russ');
$me -> print_name();
$name = $me -> get_name();
echo $me -> get_name();
$age = 36;
$me -> set_age($age);
?>
```

Constructors and destructors

- Constructors are methods that are (generally) used to initialize the object's properties with values as the object is created. Declare a constructor function in an object by writing a function with the name __construct().

```
<?php
class Person {
   protected $name;
   protected $age;
      function __construct($new_name, $new_age)  {
          $this -> name = $new_name;
          $this -> age = $new_age;
      }
  // . . . other functions here . . .
}


$p = new Person('Bob Jones', 45);
$q = new Person('Hamilton Lincoln', 67);
?>
```

Inheritance

- Use the "extends" keyword in the class definition to define a new object that inherits from another

```
<?php
  class Employee extends Person  {
     var $salary;

     function __construct($new_name, $new_age, $new_salary);  {
        $this -> salary = $new_salary;
        parent::__construct($new_name, $new_age);
     }
     function update_salary($new_salary)  {
             $this -> salary = $new_salary;
     }
$emp = new Employee('Dave Underwood', 25, 25000);
```

Inheritance (cont.)

- The constructor of the parent isn't called unless the child explicitly references it (as in this previous case). There is no automatic chain of calls to constructors in a sequence of objects defined through inheritance.
- You could "hard-code" the call to the parent constructor using the function call "Person::__construct($new_name, $new_age);" but it's better to define it in the manner given using the parent::method() notation. The same manner is used to call the method of a parent that has been overridden by its child.

Inheritance (cont.)

- You can use the "self" keyword to ensure that a method is called on the current class (if a method might be subclassed), in this style self::method();
- To check if an object is of a particular class, you can use the instanceof operator.

```
if ($p instanceof Employee) {
            //  do something here
            }
```

More on classes

- You can also define interfaces for objects (for which any object that uses that interface must provide implementations of certain methods), and you can define abstract classes or methods (that must be overridden by its children).
- The keyword "final" can be used to denote a method that cannot be overridden by its children

```
class Person {
    var $name;
    final function get_name() {
        return $this -> name;
    }
}
```

Interfaces

- Make your class follow a contract

```
<?php
// Declare the interface 'iTemplate'
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

// Implement the interface
//  This will work
class Template implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value, $template);
        }

        return $template;
    }
}

// This will not work
// Fatal error: Class BadTemplate contains 1 abstract methods
// and must therefore be declared abstract (iTemplate::getHtml)
class BadTemplate implements iTemplate
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }
}
```

More on classes (cont.)

- There are methods for "introspection" about classes, i.e. the ability of a program to examine an object's characteristics.
- For example, the function class_exists() can be used (surprise!) to determine whether a class exists or not.
- The function get_declared_classes() returns an array of declared classes.

```
$classes = get_declared_classes();
```

- You can also get an array of method names in any of the following (equivalent) manners:

```
$methods = get_class_methods(Person);
$methods = get_class_methods('Person');
$class = 'Person';
$methods = get_class_methods($class);
```

More introspection functions

- **There are a wide variety of introspection functions, several more are listed below**

```
get_class_vars($object);  /*  gets an associative array that maps
                              property names to values (including
                              inherited properties), but it *only*
                              gets properties that have default
                              values (those initialized with
                              simple constants)  */

is_object($object);  //  returns a boolean value
get_class($object);  /* returns the class to which an object belongs  */
method_exists($object, $method);  // returns a boolean value
get_object_vars($object);  /*  returns an associative array
                               mapping properties to values (for
                               those values that are set (i.e.
```

A word about object methods…

- When defining the names of your own object methods, avoid starting them with a double underscore.
  - There are some "built-in" or predefined PHP method names that start in this manner, most notably constructors and destructors using the __construct() and __destructor() names.
  - In the future (in new versions of PHP), it's possible that further methods might be defined that begin with a double underscore.
- Other reserved method names include __sleep() and __wakeup() which are used for object serialization and __get() and __set(), which can be defined so that if you try to access an object property that doesn't exist, these methods give an opportunity to either retrieve a value or set the (default?) value for that property. For example, if a class is used to represent data obtained from a database, you could write __get() and __set() methods that read and write data whenever requested.

PHP sessions

- By default, HTML and web servers don't keep track of information entered on a page when the client's browser opens another page. Thus, doing anything involving the same information across several pages can sometimes be difficult.
- Sessions help solve this problem by maintaining data during a user's visit, and can store data that can be accessed from page to page in your site.
- You can use session variables for storing information (this is one way that a "shopping cart" function can work for an online shop, for example).

PHP sessions

- Servers keep track of users' sessions by using a session identifier, which is generated by the server when a session starts and is then used by the browser when it requests a page from the server. This session ID can be sent through a cookie (the default behavior) or by passing the session ID in the URL string.
- Sessions only store information temporarily, so if you need to preserve information, say, between visits to the same site, you should likely consider a method such as using a cookie or a database to store such information.

PHP sessions (cont.)

- **To start a session, use the function session_start() at the beginning of your PHP script before you store or access any data. For the session to work properly, this function needs to execute before any other header calls or other output is sent to the browser**

```
<?php
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Session example</title>
</head>
<body>
<?php
include_once ('object.php');  //  Includes definition of the Person class
$_SESSION['hello'] = 'Hello world';
echo $_SESSION['hello'] . "<br/><br/>\n";
$_SESSION['one'] = 'one';
$_SESSION['two'] = 'two';
$me = new Person("Russ", 36, 2892700);
```

view the output page

Using session variables

- Once a session variable has been defined, you can access it from other pages.

```
<?php
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
<title>Session example 2</title>
</head>
<body>
<?php
echo "Welcome to a new page ". $_SESSION['name'] "!<br/>\n";
echo "Hope you enjoy your stay!  <br/>";
?>
<p>Back to regular HTML text...
</p>

</body>
</html>
```

view the output page

More on session variables

- You need to include a call to the session_start() function for each page on which you want to access the session variables.
- A session will end once you quit the browser (unless you've set appropriate cookies that will persist), or you can call the session_destroy() function. (Note, however, even after calling the session_destroy() function, session variables are still available to the rest of the currently executing PHP page.)
- The function session_unset() removes all session variables. If you want to remove one variable, use the unset($var) function call.
- The default timeout for session files is 24 minutes. It's possible to change this timeout.

Deleting all session variables

```
<?php
session_start();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
<title>Session example 3</title>
</head>
<body>
<?php
echo "Deleting all session variables using session_unset(); <br/>\n";
session_unset();
echo "Now the session variables are gone.  <br/>\n";
if (isset($_SESSION['name']))
   {  echo $_SESSION['name'] . "<br/>\n"; }
```

view the output page

Final Notes

It's Built In!

Available streams will vary – http, https, tcp, tcps, php are usually always present

```
<?php
$options = array(
   'http' => array(
      'method' => 'POST',
      'header'=>
         "Accept-language: en\r\n".
         "Content-type: application/x-www-form-urlencoded\r\n",
      'content' => http_build_query(array('foo'=>'bar'))
));

$context = stream_context_create($options);

$fp = fopen('http://www.example.com/', 'r', false, $context);

$response = stream_get_contents($fp);

$meta = stream_get_meta_data($fp);

fclose($fp);

print_r($response);
print_r($meta);
?>
```

```
<HTML>
<HEAD>
  <TITLE>Example Web Page</TITLE>
</HEAD>
<body>
<p>You have reached this web page by typing
&amp;quot;example.com&amp;quot;,
&amp;quot;example.net&amp;quot;,
  or &amp;quot;example.org&amp;quot; into your web browser.</p>
<p>These domain names are reserved for use in documentation
and are not available
  for registration. See <a href="http://www.rfc-editor.org/
rfc/rfc2606.txt">RFC
  2606</a>, Section 3.</p>
</BODY>
</HTML>
Array
(
    [wrapper_data] => Array
        (
            [0] => HTTP/1.1 200 OK
            [1] => Date: Sun, 07 Sep 2008 15:34:29 GMT
            [2] => Server: Apache/2.2.3 (CentOS)
            [3] => Last-Modified: Tue, 15 Nov 2005 13:24:10
GMT
            [4] => ETag: "280100-1b6-80bfd280"
            [5] => Accept-Ranges: bytes
            [6] => Content-Length: 438
            [7] => Connection: close
            [8] => Content-Type: text/html; charset=UTF-8
        )
    [wrapper_type] => http
    [stream_type] => tcp_socket
    [mode] => r+
    [unread_bytes] => 0
    [seekable] =>
    [uri] => http://www.example.com/
    [timed_out] =>
    [blocked] => 1
    [eof] => 1
)
```

Iterators

Foreach fun

```
<?php
class MyIterator implements Iterator
{
    private $var = array();

    public function __construct($array)
    {
        if (is_array($array)) {
            $this->var = $array;
        }
    }

    public function rewind() {
        echo "rewinding\n";
        reset($this->var);
    }

    public function current() {
        $var = current($this->var);
        echo "current: $var\n";
        return $var;
    }

    public function key() {
        $var = key($this->var);
        echo "key: $var\n";
        return $var;
    }

    public function next() {
        $var = next($this->var);
        echo "next: $var\n";
        return $var;
    }

    public function valid() {
        $var = $this->current() !== false;
        echo "valid: {$var}\n";
        return $var;
    }
}

$values = array(1,2,3);
$it = new MyIterator($values);

foreach ($it as $a => $b) {
    print "$a: $b\n";
}
```

```
rewinding
current: 1
valid: 1
current: 1
key: 0
0: 1
next: 2
current: 2
valid: 1
current: 2
key: 1
1: 2
next: 3
current: 3
valid: 1
current: 3
key: 2
2: 3
next:
current:
valid:
```

MySQLI

- http://php.net/mysqli
- i means improved (I didn't name it)
- Transactions, prepared statements, the stuff actually works
- Procedural API or OO API (take your pick)

Using Mysqli

Mysql Improved

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$city = "Amersfoort";

/* create a prepared statement */
if ($stmt = $mysqli-
>prepare("SELECT District FROM City WHERE Name=?")) {

    /* bind parameters for markers */
    $stmt->bind_param("s", $city);

    /* execute query */
    $stmt->execute();

    /* bind result variables */
    $stmt->bind_result($district);

    /* fetch value */
    $stmt->fetch();

    printf("%s is in district %s\n", $city, $district);

    /* close statement */
    $stmt->close();
}

/* close connection */
$mysqli->close();
```

```
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

$city = "Amersfoort";

/* create a prepared statement */
if ($stmt = mysqli_prepare($link, "SELECT District FROM City WHERE Nam
e=?")) {

    /* bind parameters for markers */
    mysqli_stmt_bind_param($stmt, "s", $city);

    /* execute query */
    mysqli_stmt_execute($stmt);

    /* bind result variables */
    mysqli_stmt_bind_result($stmt, $district);

    /* fetch value */
    mysqli_stmt_fetch($stmt);

    printf("%s is in district %s\n", $city, $district);

    /* close statement */
    mysqli_stmt_close($stmt);
}
```