CSE417: WEB ENGINEERING

Daffodil International University

You Will Be Able To

- ✔ Work with
  - ✔ forms, cookies, files, time and date.
- ✔ Create a basic checker for user-entered data.

Contents

- Including Files, time and date
- Forms and Validation
- Cookies
- Session
- Handling Files

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

?>
```

```
<?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

?>
```

*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
<body>
<?php
// Show all PHP information
phpinfo();
?>
<?php
// Show only the general information
phpinfo(INFO_GENERAL);
?>
</body>
</html>
```

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
INFO_ALL	Shows all of the above (default)

Server Variables

The $_SERVER array variable 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>
```

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

PHP Global Variables - Superglobals:
The PHP superglobal variables are:
$GLOBALS, $_SERVER, $_REQUEST, $_POST, $_GET, $_FILES, $_ENV, $_COOKIE, $_SESSION
What purpose do they serve?

Form Handling

Any form element is automatically available via one of the built-in PHP variables (provided the element has a "name" defined with it).

```
<html>
<-- form.html  -->
<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.

WHAT IS THE OUTPUT?

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  -->
<head>
<title>PHP Form example</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
}  //**  end of "print_from" function
```

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);
  }
}  //** end of "check_form" function
```

```
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";
}   //** end of "confirm_form" function
```

Main Program

```
/*Main Program*/

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

 <h3>Please enter your information</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>
```

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>
```

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>
```

`$_COOKIE`
contains all COOKIE data.

`isset()`
finds out if a cookie is set

use the cookie name as a variable

Session

- When you work with an application, you open it, do some changes, and then you close it. This is much like a Session.
  - HTTP address doesn't maintain state.
  - Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc).
  - By default, session variables last until the user closes the browser.
  - Session variables hold information about one single user
  - available to all pages in one application. ie, logged in

Example

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

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**

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

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

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

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

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

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

```php
<?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);
?>
```

Getting Time and Date

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

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

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

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

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

```php
<?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";
?>
```

time() returns current Unix timestamp

*Here is more on date/time formats: http://php.net/date

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.)

```
<?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

?>
```

Defining (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;
   }


}
```

- Use the “$this” variable when accessing properties and functions of the current object. Inside a method this variable contains a reference to the object on which the method was called.

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().
- Destructors (defined with a function name of __destructor()) are called when an object is destroyed, such as when the last reference to an object is removed or the end of the script is reached (the usefulness of destructors in PHP is limited, since, for example dynamic memory allocation isn't possible in the same way that it is in C/C++).

```
<?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, $new_age);  //  call the
   constructor
                                                          //    of parent object
     }
     function update_salary($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 typically 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.

- 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;
   }
}
```

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.
                               not null)  */
```

Exercise

- Design a form and validate its data.
- Design a user registration system
- **READINGS/Practice**
  - M Schafer: Ch. 29-32
  - https://www.w3schools.com/php/php_form_validation.asp
  - Designing a Sign-up/Log-in page

Acknowledgement

- This module is designed and created with the help from following sources-
  - https://cgi.csc.liv.ac.uk/~ullrich/COMP519/
  - http://www.csc.liv.ac.uk/~martin/teaching/comp519/