Advanced Web Design

Lecture 7 – Introduction to PHP

Origins and Uses of PHP

- PHP was developed by Rasmus Lerdorf of the Apache Group to provide Lerdorf a way to track visitors to his personal web site.
- Lerdorf originally named it Personal Home Page but it currently is known as PHP Hypertext Preprocessor
- Within two years, PHP outgrew Lerdorf's ability to manage its development. It remains open source.
- PHP is a server-side scripting language with good tools for form handling and database access.

Overview of PHP

- PHP is a server-side XHTML-embedded scripting language that is an alternative to MS Active Server Pages and Java Server Pages.
- PHP is analogous to JavaScript in that it is embedded in an XHTML document and its interpreter is called when a browser encounters a PHP script. It is indicated by the filename extensions `.php`, `.php3`, and `.phtml`.

PHP's Modes of Operation

- PHP has two modes of operations:
  - **copy mode** – when it encounters XHTML, it simply copies it into the output file.
  - **interpret mode** – when it encounters a PHP script, it interprets it and places its output in the output file.
- PHP is usually purely interpreted, although there are newer implementations that use some precompilation.

PHP and Data Types

- PHP uses dynamic typing; variables do not have an intrinsic type. It is set every time they take on a new value with the value's data type.
- PHP's arrays have characteristics of both indexed arrays and associative arrays.
- PHP also has a large library of functions, making it a flexible and powerful language for server-side web programming.

General Syntactic Characteristics

- PHP scripts can be contained within the (X)HTML document or can be in a separate file and referenced by an (X)HMTL document.
  - PHP code is embedded in an (X)HTML document between **`<?php`** and **`?>`** tag delimiters.
  - PHP code appearing in a separate file can be referenced by writing

```
<?php include("table2.inc"); ?>
```

PHP, Variables and Reserved Words

- ALL PHP variables begin with $, followed by a underscore or letter and then more underscores, letters or digits. PHP variable names are case-sensitive.
- PHP reserved words are NOT case-sensitive.
- PHP comments can begin with `#` or `//` and run until the end of the line or be enclosed in `/*` and `*/`
- PHP statements are terminated with a semi-colon.
- Compound statements are enclosed in braces but generally cannot be used to define locally scoped variables other than for another entire function.

PHP's Reserved Words

`and`	`else`	`global`	`require`	`virtual`
`break`	`elseif`	`if`	`return`	`xor`
`case`	`extends`	`include`	`static`	`while`
`class`	`false`	`list`	`switch`	
`continue`	`for`	`new`	`this`	
`default`	`foreach`	`not`	`true`	
`do`	`function`	`or`	`var`	

Primitives, Operations, and Expressions

- PHP has four scalar data types: Boolean, integer, double and string.
- PHP has two compound data types: array and object.
- PHP has two special types: resource and NULL.

Variables

- Variables are not declared in PHP; they have a value and type of NULL until they are assigned a value, in which case, they take the type of the value.
- When an unbounded variable (without an assigned value) is used, its value of NULL is coerced to a 0 or an empty string, depending on its context.
- `IsSet($myVariable)` can be used to see if `$myVariable` is unbounded (`false`) or assigned a value (`true`).

Numeric Types

- PHP has one integer type, which corresponds to C's **`long`** integer. It is usually 32 bits.
- PHP's **`double`** type corresponds to C's double type.
  - It can have a decimal point, exponent or both.
  - These are valid doubles in PHP:

    .345 345. 3.45E2 .34e-2

String Type

- PHP characters are single bytes (UNICODE is not supported). A single character is a string of length 1.
- String literals can be enclosed in single quotes, in which case everything inside them is taken literally, including escape sequences such as **`\n`**.
- To have variable values and escape sequences used in a string, enclose them in double quotes.
- Example, if $sum = 10.2
  - **`'The sum is $sum'`** is outputted exactly as written.
  - **`"The sum is $sum"`** is ouputted as 'The sum is 10.2'

Boolean Type

- Boolean values are either **`TRUE`** or **`FALSE`**, both of which are case insensitive.
- Integer values of 0 are interpreted as false; others as true.
- Strings that are empty or contain "0" are false; others are true.
- Doubles that are exactly 0.0 are false; others are true.

Arithmetic Operators And Expressions

- PHP uses the usual operators (`+`, `-`, `*`, `/`, `%`, `++`, `--`)
- If both operands are integer, `+`, `-` and `*` will produce integer values. `/` will only produce an integer value if the quotient is integer; otherwise it is coerced to double.
- If either operand is double, the result is double.
- `%` expects integer operands and will coerce to integer if necessary.

Predefined Functions in PHP

Function	Parameter Type	Returns
floor	Double	Largest integer ≤ parameter
ceil	Double	Smallest integer ≥ parameter
round	Double	Nearest integer
srand	Integer	Initializes a random number generator with the parameter
rand	Two numbers	A pseudorandom number firstParam < rand < secondParam
abs	Number	Absolute value of the parameter
min	One or more numbers	Smallest
max	One or more numbers	Largest

String Operators

- The only string operator is . (concatenation)
- Strings can be treated like arrays of characters, e.g., `$str = "apple"` means that `$str{3}` is `"l"`
- There are many string functions available:
- Example

```
$str = "Apples are good";
$sub = substr($str, 7, 1);
```

`$sub` has the value `'a'`

Commonly Used PHP String Functions

Function	Parameter Type	Returns
`strlen`	A string	The # of characters in the string
`strcmp`	2 strings	=0 if they're identical, <0 if string 1 precedes string 2; >0 if string 1 follows string 2
`strpos`	2 strings	The character position in the string 1 of the first character in string 2 if string2 is contained within string1; false if not
`strstr`	A string and an integer	A substring of the string starting from the integer parameter. If a third parameter (an integer is specified), it specifies the substring's length.

Commonly Used PHP String Functions (continued)

Function	Parameter Type	Returns
`chop`	A string	The parameter with all trailing white space removed
`trim`	A string	The parameter with all white space removed from both ends.
`ltrim`	A string	The parameter with white space removed from the beginning
`strtolower`	A string	The parameter with all uppercase letters converted to lower case
`strtoupper`	A string	The parameter with all lowercase letters converted to upper case

Implicit Scalar Type Conversions

- Type conversions in PHP can be implicit (coercions), where the context of the expressions determines what data type is expected or required.
  - There are also numeric-string conversions, where a string with e or a period is converted to double and those without it are converted to integer. If there is not sign or digit, it fails and uses 0.
  - Converting double to integer results in truncation.

Explicit Scalar Type Conversions

- Explicit type conversion can be done in 3 ways:
  1. Casting involves putting the type name in parentheses:
     `$sum = 4.777;`
     `(int) $sum` produces 4
  2. An explicit conversion can be done using the functions **`intval`**, **`doubleval`** or **`strval`**
     `intval($sum)` produces 4
  3. The **`settype`** function can convert the first parameter to the type indicated by the second parameter
     `settype($sum, "integer");` produces 4

`gettype`

- A program can determine the type of a variable in two ways:
  - using the gettype function which returns the type as a string (which can also be **`"unknown"`**)
  - using one of the functions **`is_int`**, **`is_integer`**, **`is_long`** (which test for integers), **`is_double`**, **`is_real`** and **`is_float`** (which test for doubles), **`is_bool`** and **`is_string`**.

Assignment Operators

- PHP includes the usual C-style assignment operators such as **`+=`** and **`-=`**.

Output

- Any output from a PHP script becomes part of the document that the PHP processor is building. Output must be in the form of an XHTML document.
- The **`print`** function is the easiest way to produce unformatted output:

```
print "Apples are red <br />";
print "Kumquats aren't <br />";
```

- The print function expects string but it will take other parameters:

```
print(47);
```

- Double-quoted strings have their variable names replaced by their values:

```
print "The result is $result <br />";
```

`printf`

- PHP borrows the printf function from C, whose form is

```
printf(controlString, param1,
       param2, …);
```

- The control string is a template that includes specifiers as to how the other parameters will be printed:

`%10s`	a character field of 10 characters
`%6d`	an integer field of 6 character
`%5.2f`	a float or double field of 5 characters and 2 decimal place

`printf`: An Example

```
$day = "Tuesday";
$high = 79;
printf("The high on %7s was %3d",
       $day, $high);
```

`date`

- The date function can return the date is text format, based on its parameters:
  - `l` requests the day of the week
  - `F` requests the month
  - `j` requests the day of the month
  - `S` gives us the appropriate suffix after the day of the month
  - `Y` gives the year in 4-digit format (y gives the year in 2-digit format)

today.php

```
<!?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/XHTML11/DTD/xhtml11.dtd">

<!-- today.php - A trivial example to illustrate a
  PHP document -->

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> today.php </title>
  </head>
  <body>
```

```
<p>This is a test</p>
    <p>
      <?php
         print
         "<b>Welcome to my home page <br /> <br />";
         print "Today is: </b> ";
         print date("l, F jS Y");
         print "<br />";
      ?>
    </p>
  </body>
</html>
```

Control Statements

- The control structures in PHP are very similar to C/C++/Java.
- The control expression (or condition) can be any type and is coerced to Boolean.
- The control statements include:
  - **`if`** and **`if-else`**
  - **`while`**
  - **`for`**
  - **`switch`**
  - **`do..while`**
  - **`foreach`**

Relational Operators

- PHP has 8 relational operators:
  - The usual 6 (**`==`**, **`!=`**, **`>`**, **`>=`**, **`<`**, **`<=`**)
  - **`===`** (which is true if both operands have the same value and type)
  - **`!==`** (which is the opposite of **`===`**)

Comparisons and Types

- If the operands do not have the same type, one is coerced into the type of the other.
  - If one operand is a number and the other a string, the string is coerced and then compared to the other. If the string cannot be coerced, the number is coerced and then the comparison is done.
  - If both operands are strings and can be converted to numbers, they are converted and then compared. This can be avoided by using **`strcmp`**.

Boolean Operators

- There are 6 Boolean operators (in order of precedence):
  - **`!`**
  - **`&&`**
  - **`||`**
  - **`and`**
  - **`or`**
  - **`xor`**

`if` Statements

- The `if` statement in PHP is like that of C, although there is also an `elseif`. An `if` statement can include any number of `elseif` clauses. The condition can be of any type but it is coerced to Boolean.
- Example

```
if ($day == "Saturday" || $day == "Sunday")
   $today = "weekend';
else {
   $today = "weekday";
   $work = true;
}
```

`if` Statements – An Example

```
if ($num > 0)
     $pos_count++;
elseif ($num < 0)
     $neg_count++;
else {
     $zero_count++;
     print "Another zero! <br />";
}
```

`switch` Statements

- PHP's switch statement is the same as in JavaScript.
- The control expression and case expressions can be integer, double or string, with the case expressions coerced into the type of the control expression.
- Break is necessary to avoid falling through to the next case.

`switch` Statements – An Example

```
switch ($bordersize)   {
   case "0": print "<table>"; break;
   case "1": print "<table border = "1">";
             break;
   case "4": print "<table border = "4">";
             break;
   case "8": print "<table border = "8">";
             break;
   default: print "Error – invalid value:",
     "$bordersize <br />";
}
```

Loop Statements

- PHP has **`while`**, **`for`** and **`do-while`** loops that works exactly like those in JavaScript, as well as a **`foreach`** statement.
- Example of **`while`**

```
$fact = 1;
$count = 1
while ($count < $n)    {
   $count++;
   $fact *= $count;
}
```

`do-while` and **`for`** in PHP

```
$count = 1;
$sum = 0;
do   {
  $sum += $count;
  $count++;
} while ($count <= 100);
```

```
for ($count = 1, $fact = 1; $count < $n){
  $count++;
  $fact *= $count;
}
```

Example: **powers.php**

```
<!DOCTYPE html PUBLIC "-//w3c//DTD/XHTML 1.1 //EN"
 "http://www.w3/org/TR/xhtml11/DTD/xhtml11.dtd">

<!-- powers.php
    An example to illustrate loops and arithmetic
   -->
<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> powers.php </title>
  </head>
  <body>
    <table border = "border">
      <caption> Powers table </caption>
      <tr>
        <th> Number </th>
        <th> Square Root </th>
```

```
        <th> Square </th>
        <th> Cube </th>
        <th> Quad </th>
      </tr>
      <?php
        for ($number = 1; $number <= 10; $number++) {
          $root = sqrt($number);
          $square = pow($number, 2);
          $cube = pow($number, 3);
          $quad = pow($number, 4);
          print
            ("<tr align = 'center'> <td> $number </td>");
          print
            ("<td> $root </td> <td> $square </td>");
          print
            ("<td> $cube </td> <td> $quad </td> </tr>");
        }
      ?>
    </table>
  </body>
```

Arrays

- Arrays in PHP are a cross between standard indexed arrays appearing in both conventional programming languages and hashes found in Perl, Python and Ruby.
- Each element in the array consists of a key and value.
  - If the array's logical structure is like indexed arrays, the keys can happen to be simply non-negative integers in ascending order.
  - If the array's logical structure is like hashes, the keys are string and their order is based on a system-designed hashing function.

Array Creation By Assignment

- Arrays in PHP can be created by simply assigning a value to an array element
  - Example

```
$mylist[0] = 17; # if $mylist was a scalar
                 # before, it's not an array
$yourlist[] = 5; # if the array is previously
                 # undefined this is the 0th
                 # element
$yourlist[1] = "Today is my birthday!";
$yourlist[] = 42 # stored in $mylist[2];
```

Array Creation By **`array`** Construct

- Arrays in PHP can be created by using the **`array()`** construct

```
$list = array(17, 24, 45, 91); # indexed array
                               # beginning at index 0
$list = array(1 =>17, 2 => 24, 3 => 45, 4 =>91);
                # indexed beginning at index 1
$list = array(); # empty array

$list = array("Joe" => 42, "Mary" => 42,
                "Bif" => 17);  # hashed array

$list = array("make" => "Cessna", "model" => C210,
        "year" => 1960, 3 => "sold"); #mixed array
```

Accessing Array Elements

- Individual array elements can be accessed by subscripting which is the key.
  - Brackets are used for both a numeric and a string key.

```
$ages['Mary'] = 29;
```

- Multiple elements of an array can be assigned to different variables using the list construct:

```
$trees = array("oak", "pine", "binary");
list($hardwood, $softwood,
  $data_structure) = trees;
```

Functions for Dealing with Arrays

- An entire array or a single element can be deleted using unset

```
$list = array(2, 4, 6, 8);
unset ($list[2]); # the list now has 3
                  # members
```

- The keys and the values can be separately extracted from the array using the functions **`array_keys`** and **`array_values`**

```
$highs = array ("Mon" => 74, "Tue" => 70,
"Wed" => 67, "Thu" => 62, "Fri" => 65);
$days = array_keys($highs);
$temps = array_values($highs);
```

Dealing With Arrays: **`array_key_exists`**

- The existence of an element in an array with a specific key can be determined using array_key_exists, which returns a Boolean:

```
$highs = array ("Mon" => 74, "Tue" => 70,
"Wed" => 67, "Thu" => 62, "Fri" => 65);
if (array_key_exists("Tue", $highs)) {
  $tues_high = $high("Tue");
  print
      "The high on Tuesday was $tues_high <br />";
}
```

`is_array`, `in_array`, `sizeof`

- **`is_array`** returns true if its parameter is an array.
- **`in_array`** gets 2 parameters: an expression and an array. If the expression is a value in the array, it returns true.
- **`sizeof`** returns the number of elements in an array:

```
$list = array("Bob", "Fred", "Alan",
  "Bozo");
$len = sizeof($list); #  $len = 4
```

`implode` and `explode`

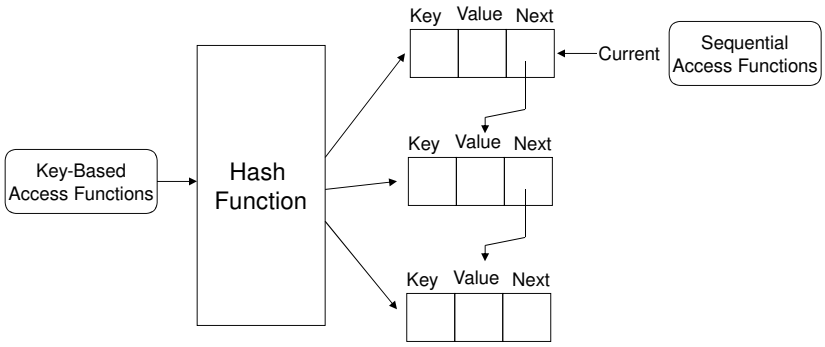
- Strings can be converted to arrays and vice versa using **`explode`** and **`implode`**.
- **`explode`** allows a string to be separated into different array elements.

```
$str = "April in Paris, Texas is nice";
$words = explode(" ", $str);
# words contains "April", "in", "Paris", "Texas",
"is", "nice"
```

- implode allows the elements of an array to be joined into a string.

```
$words = ("Are", "you", "lonesome", "tonight);
$str = implode(" ", $words);
# The string is "Are you lonesome tonight"
```

Logical Internal Structure of Arrays



Sequential Access to Array Elements

- Every array has an internal pointer that references one element of the array, which is initially pointing to the first element. It can accessed using the current function.

```
$cities = array("Hoboken", "Chicago", "Moab",
  "Atlantis");
$city = current($cities);
print("The first city is $city <br />";
```

produces **`The first city is Hoboken`**

- The next function moves the next pointer to the next element in the array; if it is already pointing to the last element, it returns false.

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

Arrays and Loop Control

- **`each`** returns a 2-element array, consisting of the key and "current" element's value.
  - It only returns false if the current pointer has gone beyond the end of the array.
  - The keys of these values are "key" and "value"

```
$salaries = array("Mike" => 42500,
          "Jerry" => 51250, "Fred" => 37900);
while ($employee = each ($salaries)) {
  $name = $employee["key"];
  $salary = $employee["value"];
  print("The salary of $name is $salary <br />");
}
```

Arrays and Loop Control (continued)

- The function **`prev`** returns the value of the previous element in the array.
- The function **`key`** returns the key of the current element.
- array_push and array_pop allow the programmer to implement a stack.

```
$num_items = array_push($list, $item1, $item2);
  places them at the end
$item = array_pop($list); # false if empty
```

foreach

- **foreach** allows each element in the array to be processed.
- There are two forms:

```
foreach (array as scalar_var) loop body
foreach (array as key => value) loop body
```

- Example:

```
foreach ($list as $temp)
   print ("$temp <br />");
```

- Example:

```
$lows = array("Mon" => 23, "Tue" => 18, "Wed" =>
  27);
foreach ($lows as $day => $temp)
print
("The low temperature on $day was $temp <br />)";
```

Sorting Arrays

- **sort** sorts an array, replacing the keys with numeric keys.
- **asort** sort arrays that correspond to Perl hashes, preserving the key-value relationship.
- **ksort** sorts the array by key, not value , preserving the key-value relationship.
- **rsort**, **arsort** and **krsort** work like sort asort and ksort repsectively, but sorting in reverse order.

sorting.php

```
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11/dtd">

<!-- sorting.php - An example to illustrate several
                   of the sorting functions -->

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head> <title> Sorting </title>
  </head>
  <body>
    <?php
      $original = array("Fred" => 31, "Al" =>27,
                  "Gandalf" => "wizard",
                  "Betty" =>42, "Frodo" => "hobbit");
    ?>
    <h4> Original Array </h4>
```

```
    <?php
      foreach ($original as $key => $value)
        print ("[$key] => $value <br />");

      $new = $original;
      sort($new);
    ?>

    <h4> Array sorted with sort </h4>

    <?php
      foreach ($new as $key => $value)
        print ("[$key] => $value <br />");

      $new = $original;
      asort($new);
    ?>
```

```
    <h4> Array sorted with asort </h4>

    <?php
      foreach ($new as $key => $value)
        print ("[$key] => $value <br />");

      $new = $original;
      ksort($new);

    ?>

    <h4> Array sorted with ksort </h4>

    <?php
      foreach ($new as $key => $value)
        print ("[$key] => $value <br />");
    ?>
  </body>
</html>
```

Functions

- The general form of a function is:

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

- The parameters are optional.
- A function can be placed anywhere in the document.
- Function overloading is not allowed.
- Functions can be nested but it is not a good idea.
- A return statement is only necessary when the function is returning a value.

General Characteristics of a Function

- The main program passes actual parameters; the function receives formal parameters.
- The number of actual and formal parameters do not have to match.
  - Excess formal parameters are unbounded.
  - Excess actual parameters are ignored.
- Parameters are passed by value. If the function needs to return more than one variable, a reference to the variable can be passed.

Example: **max_abs**

```
function max_abs($first, $second)  {
  $first = abs($first);
  $second = abs($second);
  if ($first >= 4second)
     return $first;
  else
     return $second;
}
```

Example: **set_max**

```
function set_max(&$max, $first, $second){
  if ($first >= 4second)
     $max = $first;
  else
     $max = $second;
}
```

Scope of Variables

- Variables defined in a function are local and are visible only within the function in which they're used.
- Local variables and variables outside the function with the same name do not interfere with each other.
- If it is necessary to use a variable external to the function within the function, it can be declared as global within the function.

Example of Local Variables

```
function summer($list)  {
  $sum = 0;
foreach ($list as $value)
  $sum += $value;
}

$sum = 10;
$nums = array(2, 4, 6, 8);
$ans = summer($nums);
print
  "The sum of the values in \$num is :$ans <br /> ";
print "The value of \$sum is still: $sum <br /> ";
```

Example of Global Variables

```
$big_sum = 0;
… …
function summer($list)  {
  global $big_sum;
$sum = 0;
foreach($lsit as $value)
  $sum += $value;
$big_sum += $sum;
return $sum;
}
…
$ans1 = summer ($list1);
$ans2 = summer ($list2);
print
"The sum of all array elements is: $big_sum <br />";
```

The Lifetime of Variables

- To preserve information from earlier function calls, PHP provides static variables whose lifetime begins with the script's execution and ends with the script's termination.
- Local variables can be specified as static if they re declared using the reserved **static**.

```
function do_it($param)      {
    static $count = 0;
    $count++;
    print
     "do_it has been called $count times <br />"
}
```

Pattern Matching

- PHP supports Perl-style pattern matching.
- This include preg_match:

```
if (if preg_match("/^PHP/", $str)
  print("\$str begins with PHP <br />";
else
  print
   ("\$str does not being with PHP <br />"
```

word_table.php

```
<!DOCTYPE html PUBLIC "-//w3c//DTD XHTML 1.1 //EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<!--   word_table.php
       Uses a function to split a given string of
       text into its constituent words.  It also
       determines the frequency of occurrence of
       each word.  The words are separated by
       whitespace or punctuation, possibly followed
       by whitespace.
       The punctuation can be a period, a comma, a
       semicolon, a colon, an exclamation point, or
       a question mark.
    -->
<html xmlns="http://www.w3.org/1999/xhtml">
  <head> <title> word_table.php </title>
  </head>
```

```
  <body>
    <?php
      // Function splitter
      // Parameter: a string of text containing
      // words and punctuation
      // Returns: an array in which the unique words
      // of the string are the keys and their
      // frequencies are the values.
     function splitter($str)  {
        // Create the empty word frequency array
        $freq = array();

       // split the paramneter string into words
       $words = preg_split
            ("/[\.,;:!\?\s]\s*/", $str);

       // Loop to count the words (either increment
       // or initialize to 1)
```

```
        foreach ($words as $word)  {
          $keys = array_keys($freq);
          if (in_array($word, $keys))
            $freq[$word]++;
          else
            $freq[$word] = 1;
        }
      return $freq;
      }

      // Main test driver
      $str = "apples are good for you, or don't you
              like apples? or maybe you like oranges
              better than apples";

      print $str;
      //Call splitter
      $tbl = splitter($str);
```

```
      // Display the word and their frequencies
      print "<br /> Word Frequency <br /> <br />";

      $sorted_keys = array_keys($tbl);
      sort($sorted_keys);
      foreach ($sorted_keys as $word)
        print "$word $tbl[$word] <br />";
    ?>
  </body>
</html>
```

Form Handling

- After filling out a form and clicking the Submit button, the contents of the form are encoded and transmitted to the server.
- When PHP is used to process the data, it implicitly decodes it.
- Although the PHP script that handles the form data can be in the same HTML document, it may be clearer to use a separate document.

Form Data and PHP

- PHP can be configured so that form data values are directly available as implicit whose names match the names of the corresponding form elements. But this is a potential security risk.
- The recommended method is to use the implicit **`$_POST`** and **`$_GET`** for form values.

$_POST and $_GET

- **$_POST** and **$_GET** are arrays that have keys matching the form element names and values that were input by the client.
- E.g., if the form has a text box named phone and the form method is POST, the value of that element is available to the PHP script as

  $_POST["phone"]

popcorn3.html

```
<!DOCTYPE html>
<!-- popcorn3.html - This describes the popcorn
sales form -->
<html lang = "en">
  <head>
    <title> Popcorn Sales - for PHP handling
</title>
    <meta charset = "utf-8" />
    <style type = "text/css">
      td, th, table {border: thin solid black;}
    </style>
  </head>
```

```
<body>
  <form action = "http://localhost/popcorn3.php"
        method = "post">
    <h2> Welcome to Millenium Gymnastics Booster
         Club Popcorn Sales </h2>

    <table>
 <!-- Text widgets for the customer's name and
      address -->
      <tr>
        <td> Buyer's Name: </td>
        <td> <input type = "text" name = "name"
                    size = "30" /> </td>
      </tr>
```

```
      <tr>
        <td> Street Address: </td>
        <td> <input type = "text" name = "street"
                    size = "30" /> </td>
      </tr>
      <tr>
        <td> City, State, Zip </td>
        <td> <input type = "text" name = "city"
                    size = "30" /> </td>
      </tr>
    </table>
    <p />
```

```
  <table>
<!-- First, the column headings -->
    <tr>
      <th> Product </th>
      <th> Price </th>
      <th> Quantity </th>
    </tr>

<!-- Now, the table data entries -->
    <tr>
      <td> Unpopped Popcorn (1 lb.) </td>
      <td> $3.00 </td>
      <td>
        <input type = "text" name = "unpop"
               size = "3" /> </td>
    </tr>
```

```
    <tr>
      <td> Caramel Popcorn (2 lb. cannister)
      </td>
      <td> $3.50 </td>
      <td>
        <input type = "text" name = "caramel"
               size = "3" /> </td>
    </tr>
    <tr>
      <td> Caramel Nut Popcorn (2 lb. cannister)
      </td>
      <td> $4.50 </td>
      <td>
        <input type = "text" name = "caramelnut"
               size = "3" /> </td>
    </tr>
```

```
      <tr>
        <td> Toffey Nut Popcorn (2 lb. cannister)
        </td>
        <td> $5.00 </td>
        <td>
          <input type = "text" name = "toffeynut"
                 size = "3" /> </td>
      </tr>
    </table>
    <p />
<!-- The radio buttons for the payment method -->
    <h3> Payment Method </h3>
    <p>
      <input type = "radio" name = "payment"
             value = "visa" checked = "checked" />
      Visa <br />
```

```
      <input type = "radio" name = "payment"
             value = "mc" />
      Master Card <br />
      <input type = "radio" name = "payment"
             value = "discover" />
      Discover <br />
      <input type = "radio" name = "payment"
             value = "check" />
      Check <br /> <br />

  <!-- the submit and reset buttons -->
      <input type = "submit"
             value = "Submit Order" />
      <input type = "reset"
             value = "Clear Order Form" />
```

```
      </p>
    </form>
  </body>
</html>
```

Cookies

- A session is the time span during which a browser interacts with a particular server, beginning when a browser connects to the server and ends when the browser is terminated or the server terminates the session because of inactivity.
- HTTP does not have any method for storing information about a given session that a subsequent session might use.
  - Shopping need to be able identify their specific clients
  - ***Personalization*** – creating profiles of visitors to web sites.

Introduction to Cookies

- A cookie is a small object of information that includes a name and a textual value. It is created by software system on the server.
- Cookies are assigned a lifetime; at the end of its lifetime, they are deleted from the browser's host machine.
- Cookies raise concerns about the client's privacy; browsers typically allow the user to delete cookies.

PHP Support for Cookies

- A cookie is set using

```
setcookie(cookieName[,newValue]
          [,expirationTime]);
```

- If a new value isn't specified, **`setcookie`** undefines the cookie.
- The default value for expiration time is 0; if specified, it's given as the seconds since January 1, 1970.
- Example

```
setcookie("voted", "true", time()+86400);
```

Session Tracking

- Sometimes information about a session is needed only during the session.
- Rather than using cookies, session tracking uses a single session array to store information about the client's previous requests during the session.
- In PHP, a session ID is an internal value that identifies the session.

`session_start()`

- The first call to **`session_start()`** makes PHP aware that the script is interested in session tracking.
- During subsequent call, the function retrieves the $_SESSION array, which stores any session variables and their values that wer registered in previously executed scripts during that session.

Session Tracking – A Example

```
session_start();
if (!IsSet($_SESSION["page_number"]))
  $_SESSION["page_number"] = 1;
$page_num = $_SESSION["page_number"];
print("You have now visited $page_num pages<br /> ");
$_SESSION["page_num"]++;
```