Chapter 5

Programming in PHP and MySQL

Origins and Usage of PHP:

- PHP was developed by Rasmus Lerdorf, a member of the Apache Group, in 1994.
- Its initial purpose was to provide a tool to help track visitors personal Web site, called Personal Home Page Tools
- Originally, PHP was an acronym for Personal Home Page.
- Later, its user community began using the recursive name PHP: Hypertext Preprocessor
- Within two years of its release, PHP was being used at a large number of Web sites.
- Now, it is powerful enough to be at the core of the biggest blogging system on the web (WordPress) and deep enough to run the largest social network (Facebook).
- Today, PHP is developed, distributed, and supported as an **open-source** product.

Why PHP is called server-side scripting language?

It is because of the following reasons:

- To run PHP code we need to install PHP Parser (CGI or server module) on Web server.
- Every PHP code is executed on the server, generating HTML which is then sent back to the requesting client.

Main uses of PHP:

- PHP is popularly used server-side scripting language, that we can use to do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies. The three main areas where PHP scripts are used:
  1. **Server-side Scripting (mostly used for web programming):**
     As a server-side scripting language, PHP is widely used to develop dynamic and interactive web pages. PHP is popularly used for:
     - Form handling,
     - File processing,
     - Database access.
     - Maintaining session and cookies,
     - Data encryption and so on.
  2. **Command-line Scripting (used in web programming in special cases):**
     - Without using Web server and client, the PHP parser can only be used for developing scripts executing regularly using **cron** (on *nix or Linux) or Task Scheduler (on Windows).
     - These scripts can also be used for simple text processing tasks.
  3. **Writing Desktop Applications (Not the very best language):**
     - Using advanced PHP features, mainly the extension PHP-GTK, PHP can be used to create a desktop application with a graphical user interface.

- PHP supports
  - the common electronic mail protocols: POP3 and IMAP.
  - the distributed object architectures COM and CORBA.

Overview of PHP:

- PHP is a server-side HTML- or XHTML-embedded scripting language and a powerful tool for making dynamic and interactive Web pages.
- It is a widely-used, free, and efficient alternative to: CGI, APS (ms Active Server Pages), JSP (sun's JavaServer Pages), ColdFusion (Allaire's), etc.
- `PHP` has similarities with `JavaScript`
- When a browser finds `JavaScript` in an HTML document The browser calls the `JavaScript` interpreter to execute the script
  - When a browser request a document that includes a `PHP` script. The *server* providing the document calls the `PHP` processor

- A document is recognized as including `PHP` script if its extension is: `.php`, `.php3`, `.phtml`
- The `PHP` preprocessor modes of operation:
  Copy mode:
  - HTML code in the input file ==> copies it to the output file
  - Which may include client-side scripts

  Interpreter Mode:
  - `PHP` script in the input file ==> interprets it and sends any output to the output file

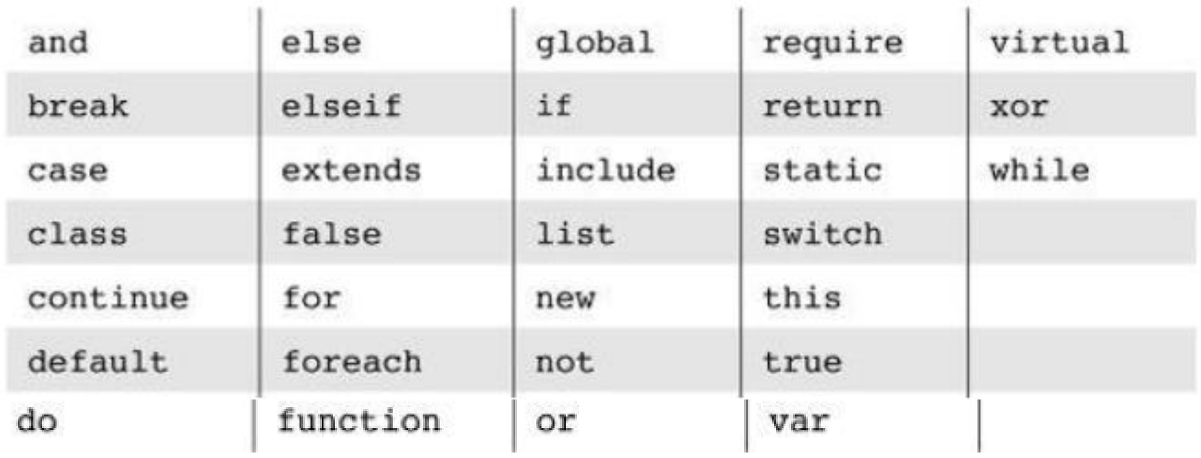
- The output form the `PHP` script must be HTML which may have embedded client-side scripts
- Syntax and semantics of `PHP` closely related to `JavaScript` and `Perl`
- `PHP` uses dynamic typing. Variables take on the type of the value it is being assigned
- `PHP` arrays are a merge of the arrays of `C/Java` and the hashes of `Perl.`
- There is a large collection of functions for creating and manipulating `PHP`'s arrays
- PHP supports both procedural and object-oriented programming.
- PHP has an extensive library of functions, making it a flexible and powerful tool for server-side software development.
- Many of the predefined functions are used to provide interfaces to other software systems, such as mail and database systems.
- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases

General Syntactic Characteristics

- PHP scripts either are embedded in HTML or XHTML documents or are in files that are referenced by such documents.
- PHP code is embedded in documents by enclosing it between the `<?php` and `?>` tags.

- If a PHP script is stored in a different file, it can be brought into a document with the `include` construct, which takes the filename as its parameter— for example, `include("table2.inc");`
- Content of file `table2.inc` are copied into the document where the call appears. `PHP` preprocessor changes from interpreter to copy mode when an `include` is encountered.
- All variable names in PHP begin with a dollar sign (`$`).
- Followed by a letter or underscore, followed by any number of letters, digits, or underscores
- Variable names are case sensitive
- Neither reserved names nor functions are case sensitive

Reserved Words of PHP



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	

- PHP allows comments to be specified in different ways:
  - Single-line comments can be specified either with `#` or with `//`, as in JavaScript.
  - Multiple-line comments are delimited with `/*` and `*/`, as in many other programming languages.
- `PHP` statements terminated with a semi-colon (`;`).
- Braces used to form compound statements for control structures.

<u>Primitives, Operations and Expressions</u>

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

1. **<u>Variables:</u>**
   - Because PHP is dynamically typed, it has no type declarations. Thus, there is no way or need to ever declare the type of a variable.

- The type of a variable is set every time the variable is assigned a value.
- An unassigned variable, sometimes called an unbound variable, has the value `NULL`, which is the only value of the NULL type.
- If an unbound variable is used in an expression, `NULL` is coerced to a value that is dictated by the context of the use:
  - If the context specifies a number, `NULL` is coerced to `0`;
  - If the context specifies a string, `NULL` is coerced to the empty string.
- The function `IsSet` takes a variable's name as its parameter and returns a `Boolean`:
  - `TRUE` if the variable has a non-NULL value
  - `FALSE` otherwise
- The function `unSet` sets a variable back to the unassigned state
- `error_reporting(15);`
- Changes the error reporting level of the `PHP` interpreter
  so the user can be informed when an unbound variable is referenced.
  The default level is `7`.

2. **<u>Integer Type:</u>**
   - `PHP` has a single integer type `integer`
   - which means its size is that of the word size of the machine on which the program is run.

3. **<u>Double Type:</u>**
   - Double literals can include a decimal point, an exponent, or both.
   - The exponent has the usual form of an `E` or an `e`, followed by a possibly signed integer literal.
   - There need not be any digits before or after the decimal point, so both `.345` and `345.` Are legal double literals.

4. **<u>String Type:</u>**
   - Characters in PHP are single bytes; UNICODE is not supported.
   - There is no character type.
   - A single character data value is represented as a string of length 1.
   - String literals are defined with either single-quote (`'`) or double-quote (`"`) delimiters.
   - In single-quoted string literals, escape sequences, such as `\n`, are not recognized as anything special and the values of embedded variables are not substituted for their names. (Such substitution is called interpolation.)
   - In double-quoted string literals, escape sequences are recognized and embedded variables are replaced by their current values. For example, the value of
     `'The sum is: $sum'`
     is exactly as it is typed. However, if the current value of `$sum is 10.2`, then the value of
     `"The sum is: $sum"` is The sum is: 10.2

- If a double-quoted string literal includes a variable name, but you do not want it interpolated, precede the first character of the name (the dollar sign) with a backslash (\).
- If the name of a variable that is not set to a value is embedded in a double-quoted string literal, the name is replaced by the empty string.
- Double-quoted strings can include embedded newline characters that are created with the Enter key.
- Such characters are exactly like those that result from typing `\n` in the string.
- The length of a string is limited only by the memory available on the computer.

5. **Boolean Type:**
    - An integer expression used in Boolean context evaluates to

      FALSE if the expression value is `0`,

      TRUE if the expression value is NOT `0`

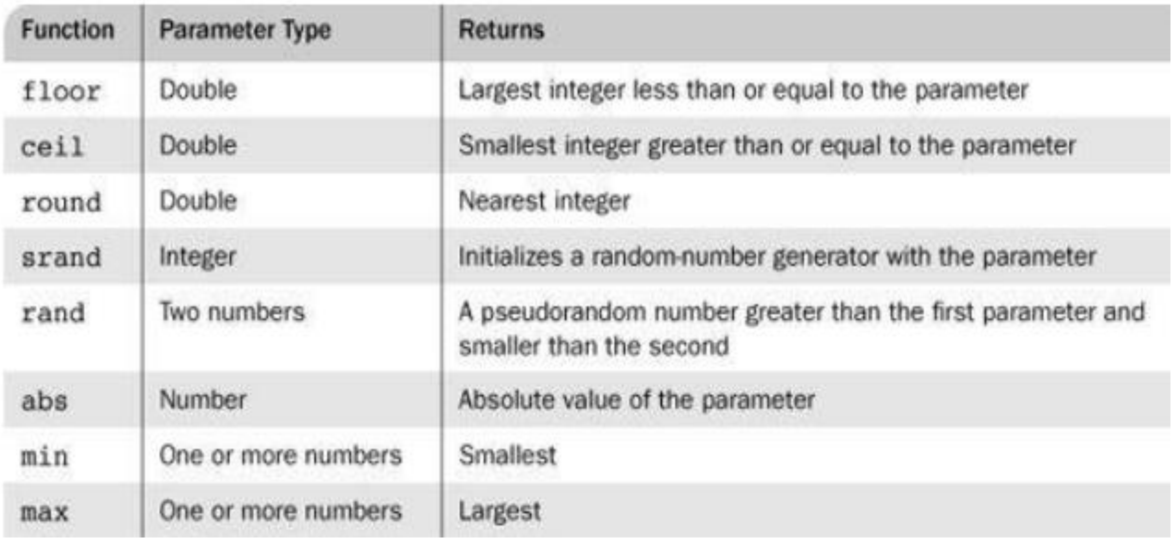
    - A string expression used in a Boolean context evaluates to

      i. FALSE if the expression value is `"0"`, or the empty string
      ii. TRUE otherwise

    - The string "0.0" evaluates to TRUE.

6. **Arithmetic Operators/Expressions:**
    - Arithmetic Operators: `+, -, *, /, %, ++,` and `−`
    - `+, -, *` produce a double(integer) for double (integer) operands
    - If the result of integer division is not an integer, a double is returned
    - The operands of the modulus operator `%` are coerced to integers if they are not.
    - Large number of predefined functions to operate on numeric values

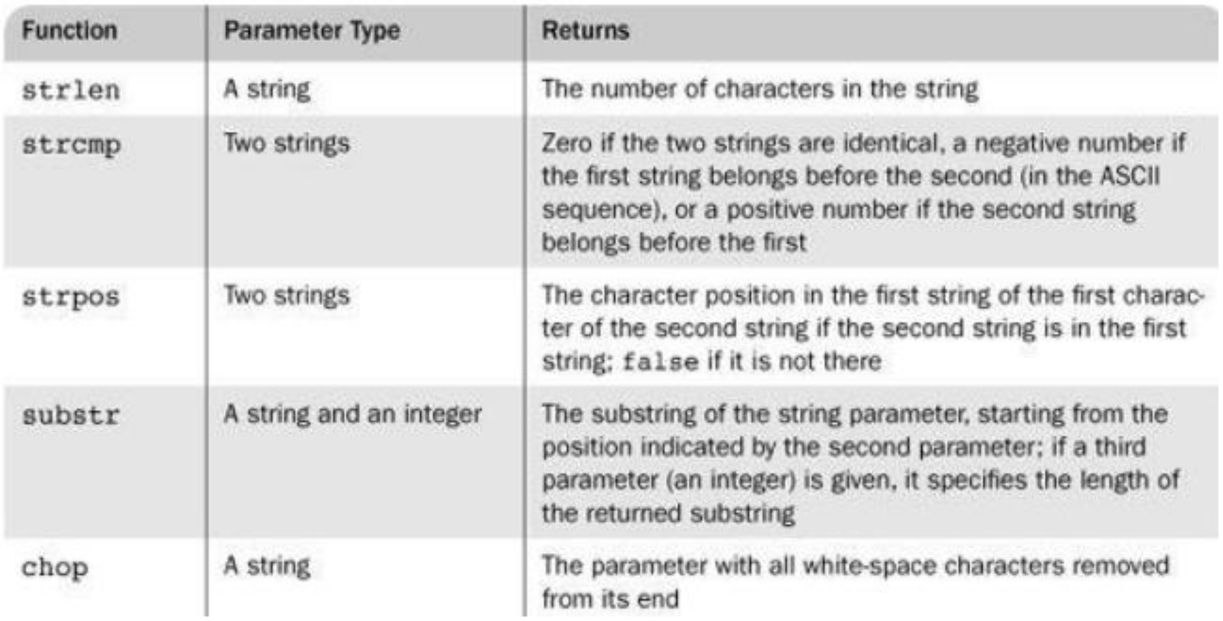
Some useful predefined functions



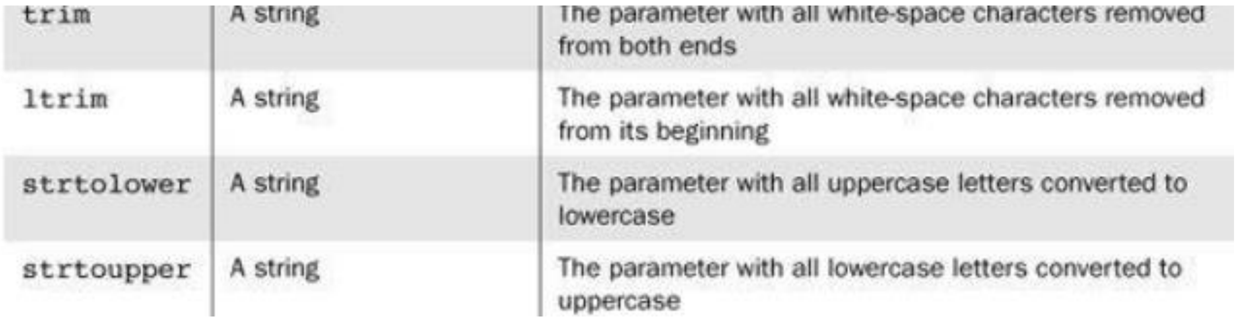
Function	Parameter Type	Returns
`floor`	Double	Largest integer less than or equal to the parameter
`ceil`	Double	Smallest integer greater than or equal to the parameter
`round`	Double	Nearest integer
`srand`	Integer	Initializes a random-number generator with the parameter
`rand`	Two numbers	A pseudorandom number greater than the first parameter and smaller than the second
`abs`	Number	Absolute value of the parameter
`min`	One or more numbers	Smallest
`max`	One or more numbers	Largest

7. **String Operations:**
   - The only string operator is the catenation operator, specified with a period (`.`).
   - String variables can be treated somewhat like arrays for access to individual characters.
   - The position of a character in a string, relative to zero, can be specified in braces immediately after the variable's name. For example, if `$str` has the value `"apple"`, `$str{2}` is `"p"`

Some commonly used string functions



Function	Parameter Type	Returns
`strlen`	A string	The number of characters in the string
`strcmp`	Two strings	Zero if the two strings are identical, a negative number if the first string belongs before the second (in the ASCII sequence), or a positive number if the second string belongs before the first
`strpos`	Two strings	The character position in the first string of the first character of the second string if the second string is in the first string; `false` if it is not there
`substr`	A string and an integer	The substring of the string parameter, starting from the position indicated by the second parameter; if a third parameter (an integer) is given, it specifies the length of the returned substring
`chop`	A string	The parameter with all white-space characters removed from its end



`trim`	A string	The parameter with all white-space characters removed from both ends
`ltrim`	A string	The parameter with all white-space characters removed from its beginning
`strtolower`	A string	The parameter with all uppercase letters converted to lowercase
`strtoupper`	A string	The parameter with all lowercase letters converted to uppercase

- Because `false` is interpreted as zero in a numeric context, this can be a problem. To avoid it, use the === operator to compare the returned value with zero to determine whether the match was at the beginning of the first string parameter (or whether there was no match).

  Consider the following example of the use of a string function:
  ```
  $str = "Apples are good";
  $sub = substr($str, 7, 1);
  The value of $sub is now 'a'
  ```

8. <u>Scalar Type Coversions:</u>

- PHP includes: both implicit and explicit type conversions.

1. <u>Implicit Conversion:</u>
   - Implicit type conversions are *coercions*
   - Numeric values in a string context are coerced to a string
   - String values appearing in a numeric context are coerced to numeric values
   - If the string contains only digits, it is converted to an integer
   - If the string contains only digits and a period or an `e/E`, it is converted to an double
   - If the string does not begin period, a sign or a digit, it is converted to zero
   - When a double is converted to an integer the fractional part is dropped

2. <u>Explicit Conversion</u>:
   - An expression can be cast to a different type
     ```
     $sum = 4.777;
     $res = (int)$sum;  # value of $res is now 4
     ```
   - Use type conversion functions: `intval, doubleval,` or `strval`
     ```
         $res = intval(4sum);  # value of $res is now
     ```
   - The type of a variable can be determined
     - The `gettype` function takes on the variable's name as a parameter

       Returns a string with the name of the type of the variable

       It may return "`unknown`"

- The testing type functions: `is_integer, is_double, is_string, is_bool`, etc.

9. **Assignment Operators**
   PHP has the same set of assignment operators as its predecessor language, C, including the compound assignment operators such as `+= and /=.`

Output and Control Statements:

Output:

Any output of a `PHP` script becomes part of the document the `PHP` preprocessor is writing.

All output must be in the form of HTML which may include embedded client-side scripts

There are three forms of producing output in a `PHP` script

`echo`, `print`, and `printf`

1. `echo`
   - Can be called without parenthesis around several parameters

   - `echo "A rose by any other name <br />", "Is a rose <br />";`

   - If parenthesis are used, only a single parameter is allowed `echo` does not return a value
2. `print`
   - Can never be called with more than one parameter
   - It returns a value: `1` if succeeded; `0` if failed

The following example displays the day of the week using `date` function:

- `l` requests the day of the week
- `F` requests the month
- `j` requests the day of the month
- `S` gets the correct ordinal suffix

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 strict//EN"
"http://www.w3.org/TR/xhtml1/dtd/xhtml1-strict.dtd">
<!-- today.php
  A trivial example to illustrate a php document
  -->
```

```
<html>
<head>
<title>
today.php</title>
</head>
<body>
<p>
<?php
 print "<b>Welcome to my home document <br /><br />";
 print "Today is: </b>";
 print date("l, F jS");
 print "<br /><br />";
?>
</p>
</body>
</html>
```

3. `printf`
   - It is used when control over the format of displayed data is required. The general form of a call to `printf` is as follows:

     printf(*literal_string*, *param1*, *param2*, ...)

     - The literal string can include labeling information about the parameters whose values are to be displayed. It also contains format codes for those values.
     - The form of the format codes is a percent sign (`%`) followed by a field width and a type specifier. The most common type specifiers are `s` for strings, `d` for integers, and `f` for floats and doubles.
     - The field width is either an integer literal (for integers) or two integer literals separated by a decimal point (for floats and doubles).
     - The integer literal to the right of the decimal point specifies the number of digits to be displayed to the right of the decimal point. The following examples illustrate how to specify formatting information:
       `%%5.2f`—a float or double field of eight spaces, with `10s`—a character string field of 10 characters
       `%6d`—an integer field of six digits
       two digits to the right of the decimal point, the decimal point, and five digits to the left.

Control Statements:

Control statements in `PHP` are very similar to those of `C`

Control expressions can be of any type

If necessary the interpreter evaluates the expression and coerces them Boolean

- **Relational Operators**

  - `PHP` uses the same relational operators as JavaScript
    `>, <, >=, <=, !=, ==`
    If the types of the operands are not the same, *one is coerced to the type of the other*
  - `PHP` also checks for same type and value with ===
    The opposite of === is !==
  - `PHP` prefers to do numeric comparisons.
  - Any operand that is a string but can be coerced to a number will be converted to number and numeric comparison will be done.
  - .If the string cannot be converted to a number, the numeric operand will be converted to a string and a string comparison will be done.
  - If the string cannot be converted to a number, the numeric operand will be converted to a string and a string comparison will be done. If both operands are strings that can be converted to numbers,
  - To avoid it and similar problems associated with string-to-number coercions, if either or both operands are strings that could be converted to numbers, the `strcmp` function should be used rather than one of the comparison operators.

- **Boolean Operators**

  - Six Boolean operators: `and, or, xor, !, &&, ||`
  - The precedence of `and, or` is higher than `&&, ||`
- **Selection Statements**

  - The `if` is like `C`
  - The control expression can be of any type,
    Its value is coerced to Boolean
  - The `switch` statement has the for of `JavaScript`
- **Loop Statements**

  - The `while, for, do-while` statements are exactly like `JavaScript`
  - `PHP` has a `foreach` statement similar to `Perl`
  - `break, continue` statements are similar to `JavaScript` .

- **Alternate Compounds Delimeters**

  - Compound statements being controlled by `if, switch, for, while` can be controlled by braces or alternative by colon (`:`)

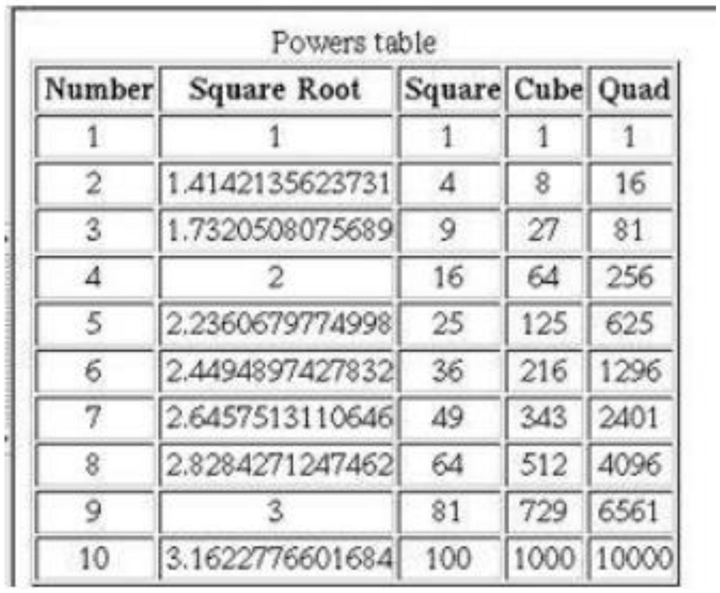
- If a colon is used to delimit the statements, they are closed with `endif`, `endswitch`, `endfor`, `endwhile`

```
while ($a <100) :
   $a = $a * $b + 7;
   $b ++;
endwhile;

if ($a < $b):
   $smaller =$a;
   print("\$a is smaller
");
else:
   $smaller =$b;
   print("\$b is smaller
");
endif;
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 strict//EN"
"http://www.w3.org/TR/xhtml1/dtd/xhtml1-strict.dtd">

<!--  powers.php
      An example to illustrate loops and arithmetic
      -->
<html>
<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>
</html>
```

OUTPUT:



Powers table

Number	Square Root	Square	Cube	Quad
1	1	1	1	1
2	1.4142135623731	4	8	16
3	1.7320508075689	9	27	81
4	2	16	64	256
5	2.2360679774998	25	125	625
6	2.4494897427832	36	216	1296
7	2.6457513110646	49	343	2401
8	2.8284271247462	64	512	4096
9	3	81	729	6561
10	3.1622776601684	100	1000	10000

Arrays:

- An array is a special variable, which can hold more than one value at a time.
- Like a Perl hash, arrays in PHP have two parts: a **key** and a **value**
- PHP arrays can have some elements with integer keys and some with string keys.

Type of Array in PHP:

In PHP, there are three types of arrays:

i. **Indexed arrays** - Arrays with a numeric index
ii. **Associative arrays** - Arrays with named keys
iii. **Multidimensional arrays** - Arrays containing one or more arrays

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:
```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:
```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:
```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:
```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

To loop through and print all the values of an associative array, you could use a foreach loop as below:
```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
   echo "Key=" . $x . ", Value=" . $x_value;
   echo "<br>";
}
?>
```

PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP understands multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

The dimension of an array indicates the number of indices you need to select an element.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays.

For example:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array
 (
 array("Volvo",22,18),
 array("BMW",15,13),
 array("Saab",5,2),
 array("Land Rover",17,15)
 );
```

We can also put a `for` loop inside another `for` loop to get the elements of the $cars array as below:

```
<?php
for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>
```

- **Array Creation**
  - Assigning a value to an array that does not yet exist, creates the array

```
$list[0] =17;

$list[2]="This is a nice day";

$list[]=33;  #this element key will be 3
```

Using the `array` construct

```
$list2 = array(12, 34, 45, 56);
   /* traditional array with 4 elements
      with the keys 0, 1, 2, 3 */

$list3 = array( 1 => 12, 3 => 34, 4 => 45, 7 => 56);
   /* an array with four elements
      with the keys 1, 3, 4, 7 */

$list4 = array();  # an empty array

$list5 = array("Tom" => 13, "Dick" => 18, "Jerry" =>
24);
   /* an array with three elements
      with the keys "Tom", "Dick", "Jerry" */
$list6 = array( "make" => "volvo", "model" => "245",
         "year" => 1974, 3 => "Sold" );
   /* PHP arrays can be mixtures of
      traditional arrays and hashes */
```

- **Accessing Array Elements**
  - Array elements are accessed by subscripting

```
$val = $list3[3];  # $val has a value of 34
$val = $list5['Tom']; # $val is 13
$val = $list6[3]; # $val is "Sold"
```

  - The `list` construct works just like in `Perl`
  - The `array_keys` function takes an array and returns the array keys

```
$keys6 = array_keys($list6);
/* $keys6 holds the keys of $list6 */
$valu6 = array_values($list6);
   /* $valu6 holds the values of $list6 */
```

- **Functions for Dealing with Arrays**
  - A whole array can be deleted with `unset`, as with a scalar variable. Individual elements of an array also can be removed with `unset`, as in the following code:
    $list = array(2, 4, 6, 8);
    unset($list[2]);

  - After executing these statements, `$list` has three remaining elements with keys `0`, `1`, and `3` and elements `2`, `4`, and `8`.
  - The collection of keys and the collection of values of an array can be extracted with built-in functions.
  - The `array_keys` function takes an array as its parameter and returns an array of the keys of the given array.
  - The returned array uses `0`, `1`, and so forth as its keys.
  - The `array_values` function does for values what `array_keys` does for keys. For example, sets the value of `$days` to (`“Mon‟, “Tue‟, “Wed‟, “Thu‟, “Fri‟`) and the value of `$temps` to (`74, 70, 67, 62, 65`). In both cases, the keys are (`0, 1, 2, 3, 4`).
  - The existence of an element of a specific key can be determined with the `array_key_exists` function, which returns a Boolean value. The following code is illustrative:

```
$highs = array("Mon" => 74, "Tue" => 70, "Wed" => 67,
               "Thu" => 62, "Fri" => 65);
if (array_key_exists("Tue", $highs)) {
  $tues_high = $highs["Tue"];
  print "The high on Tuesday was $tues_high <br />";
} else
  print
   "There is data for Tuesday in the \$highs array <br />";
```

- The `is_array` function is similar to the `is_int` function: It takes a variable as its parameter and returns `TRUE` if the variable is an array, `FALSE` otherwise.
- The `in_array` function takes two parameters—an expression and an array—and returns `TRUE` if the value of the expression is a value in the array; otherwise, it returns `FALSE`.
- The number of elements in an array can be determined with the `sizeof` function. For example, after the code

  $list = array(“Bob”, “Fred”, “Alan”, “Bozo”);
  $len = sizeof($list);

  is executed, `$len` will be `4`
- It is often convenient to be able to convert between strings and arrays.
- These conversions can be done with the `implode` and `explode` functions.
- The `explode` function explodes a string into substrings and returns them in an array.
- The delimiters of the substrings are defined by the first parameter of `explode`, which is a string; the second parameter is the string to be converted. For example, consider the following:

  $str = “April in Paris, Texas is nice”;
  $words = explode(“ “, $str);

  Now `$words` contains (`“April″`, `“in″`, `“Paris,″`, `“Texas″`, `“is″`, `“nice″`);

- The `implode` function does the inverse of `explode`.

Logical Internal Structure of Array:

- Internally, the elements of an array are stored in a linked list of cells, where each cell includes both the key and the value of the element.
- The cells themselves are stored in memory through a key-hashing function so that they are randomly distributed in a reserved block of storage.
- Accesses to elements through string keys are implemented through the hashing function.
- However, the elements all have links that connect them in the order in which they were created, allowing them to be accessed in that order if the keys are strings and in the order of their keys if the keys are numbers.

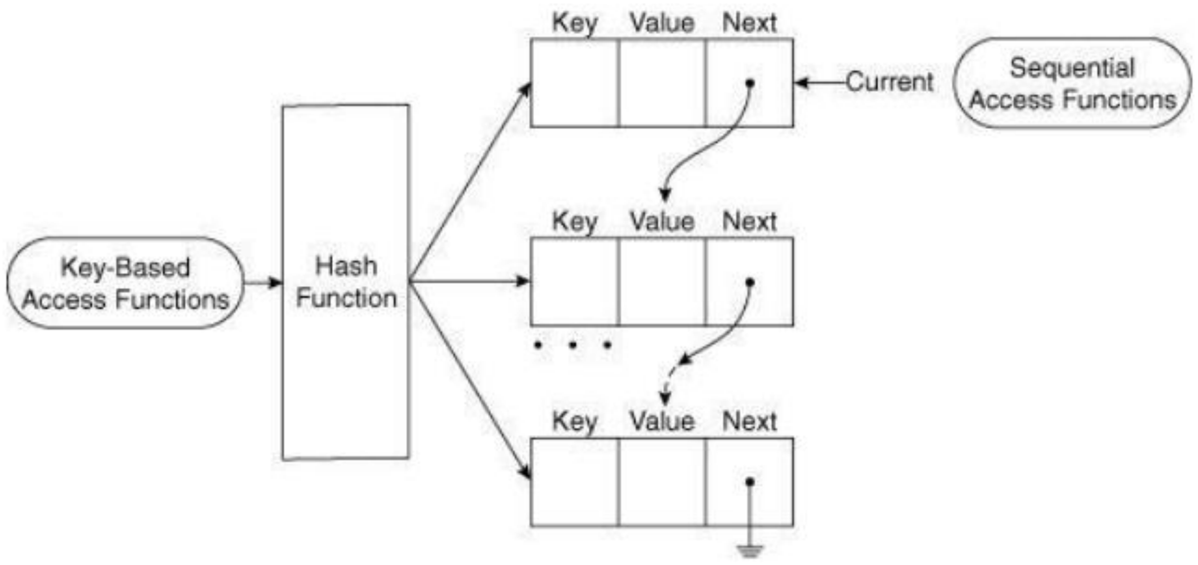


Fig. Logical Internal Structure of Array

- **Sequential Access to Array Elements**

  - PHP includes several different ways to access array elements in sequential order.
  - Every array has an internal pointer that references one element of the array.
  - We call this the "current" pointer.
  - This pointer is initialized to reference the first element of the array at the time the array is created.
  - The element being referenced by the pointer can be obtained with the `current` function. For example, the code:

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

Output:

The first city is Hoboken

- The "current" pointer can be moved with the `next` function, which both moves the pointer to the next array element and returns the value of that element.
- If the "current" pointer is already pointing at the last element of the array, `next` returns `FALSE`.
- For example, if the "current" pointer is referencing the first element of the `$cities` array, the following code produces a list of all of the elements of that array:

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

- Due to various problems in **next** function, **each** function is used.
- As an example of the use of `each`, consider the following code:

```
$salaries = array("Mike" => 42500, "Jerry" => 51250,
                  "Fred" => 37920);
while ($employee = each($salaries)) {
  $name = $employee["key"];
  $salary = $employee["value"];
  print("The salary of $name is $salary <br />");
}
```

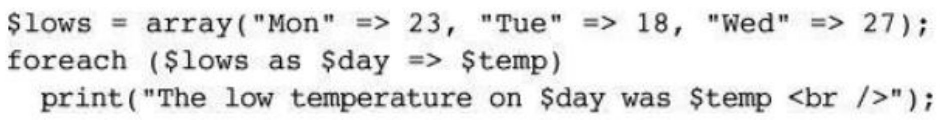
Output:

```
The salary of Mike is 42500
The salary of Jerry is 51250
The salary of Fred is 37920
```

- The "current" pointer can be moved backward (i. e., to the element before the "current" element) with the `prev` function.
- Like the `next` function, the `prev` function returns the value of the element referenced by the "current" pointer after the pointer has been moved.
- The "current" pointer can be set to the first element with the `reset` function, which also returns the value of the first element.
- It can be set to the last element of the array with the `end` function, which also returns the value of the last element.
- The `key` function, when given the name of an array, returns the key of the "current" element of the array.

- The `array_push` and `array_pop` functions provide a simple way to implement a stack in an array.
- The `array_push` function takes an array as its first parameter. After this first parameter, there can be any number of additional parameters. The values of all subsequent parameters are placed at the end of the array.
  The `array_push` function returns the new number of elements in the array.
- The `array_pop` function takes a single parameter: the name of an array. It removes the last element from the array and returns it. The value `NULL` is returned if the array is empty.
- The `foreach` statement is designed to build loops that process all of the elements of an array. This statement has two forms:
  foreach (*array* as *scalar_variable*) *loop body*
  foreach (*array* as *key* => *value*) *loop body*
  produces the values of all of the elements of `$list`.

  The second form of `foreach` provides both the key and the value of each element of the array:



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

<u>Sorting Arrays:</u>

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key

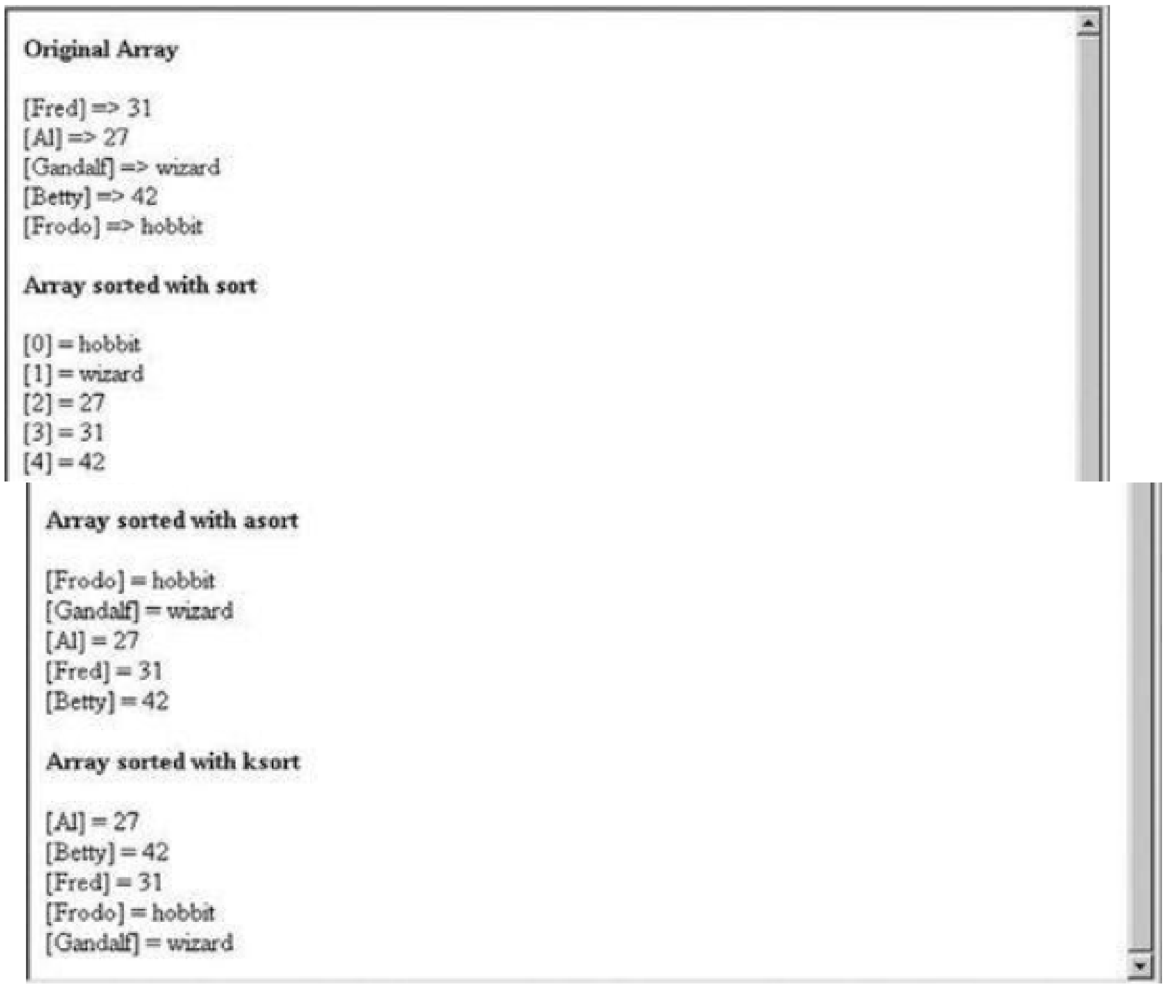
- The following example illustrates `sort`, `asort`, and `ksort`:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 strict//EN"
"http://www.w3.org/TR/xhtml1/dtd/xhtml1-strict.dtd">

<!--  sorting.php
      An example to illustrate sorting functions
      -->
<html>
<head>
   <title>
      Sorting Functions
```

```
    </title>
</head>

<body>
<?php
$original = array("Fred"=> 31, "Al" => 27, "Gandalf" =>
"wizzard","Betty" => 42, "Frodo" => "hobbit" );
    ?>
    <h4>Original Array</h4>
    <?php
    foreach ($original as $key => $value )
       print("[$key] => $value <br />");

    $new = $original;
    sort($new);
    ?>
    <h4>Array Sorted by <tt>sort</tt></h4>
    <?php
    foreach ($new as $key => $value )
       print("[$key] => $value <br />");

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

    <h4>Array Sorted by <tt>sort,  SORT_NUMERIC</tt></h4>
    <?php
    foreach ($new as $key => $value )
       print("[$key] => $value <br />");

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

    <h4>Array Sorted by <tt>rsort</tt></h4>
    <?php
    foreach ($new as $key => $value )
       print("[$key] => $value <br />");

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

    <h4>Array Sorted by <tt>asort</tt></h4>
    <?php
    foreach ($new as $key => $value )
       print("[$key] => $value <br />");
```

```
$new = $original;
arsort($new);
?>
<h4>Array Sorted by <tt>arsort</tt></h4>
<?php
foreach ($new as $key => $value )
   print("[$key] => $value <br />");
?>
</body>
</html>
```

Output:



```
Original Array

[Fred] => 31
[Al] => 27
[Gandalf] => wizard
[Betty] => 42
[Frodo] => hobbit

Array sorted with sort

[0] = hobbit
[1] = wizard
[2] = 27
[3] = 31
[4] = 42

Array sorted with asort

[Frodo] = hobbit
[Gandalf] = wizard
[Al] = 27
[Fred] = 31
[Betty] = 42

Array sorted with ksort

[Al] = 27
[Betty] = 42
[Fred] = 31
[Frodo] = hobbit
[Gandalf] = wizard
```

Fig. Output of `sorting.php`

Functions

General Characteristics of Functions:

- The general form of a PHP function definition is as follows:
  function *name*([*parameters*]) {
  ...
  }
- Function definitions can be nested, as they can in JavaScript.
- Function names are not case sensitive.
- A document cannot have a function named `sum` and another named `Sum`. The PHP interpreter will see
  them as the same function and issue an error message stating that the document has two definitions for the same function.
- The `return` statement is used in a function to specify the value to be returned to the caller. Function execution ends when a `return` statement is encountered or the last statement in the function has been executed. In either case, control returns to the caller. If no `return` statement was executed, no value is returned.
- If one or more related functions are used by more than one document, it is convenient to store their definitions in a separate file and copy that file into those documents when they are requested by a client (browser). This is done with the `include` function.

Parameters:

An actual parameter can be any expression. A formal parameter must be a variable name.

- The number of actual parameters in a call to a function need not match the number of formal parameters defined in that function.
- If there are too few actual parameters in a call, the corresponding formal parameters will be unbound variables.
- If there are too many actual parameters, the excess actual parameters will be ignored. The absence of a requirement for matching numbers of parameters allows the language to support functions with a variable number of parameters.
- The default parameter-passing mechanism of PHP is pass by value.
- This means that, in effect, the values of actual parameters are copied into the memory locations associated with the corresponding formal parameters in the called function. The values of the formal parameters are never copied back to the caller, so passing by value implements one-way communication to the function. This is the most commonly needed mechanism for parameter passing.
- For example:

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

- This function returns the larger of the absolute values of the two given numbers. Although it potentially changes both of its formal parameters, the actual parameters in the caller are unchanged (because they were passed by value).
- Pass-by-reference parameters can be specified in PHP in two ways. One way is to add an ampersand (&) to the beginning of the name of the formal parameter that you want to be passed by reference.

For example:

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

<u>The Scope of Variables:</u>

- The default scope of a variable defined in a function is local.
- If a variable defined in a function has the same name as a interference between the two. A local variable is visible only in the function in which it is used. For example, the code:

```
function summer($list) {
  $sum = 0;
  foreach ($list as $value)
    $sum += $value;
  return $sum;
}
$sum = 10;
```

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

produces the following output:

```
The sum of the values in $nums is: 20
The value of $sum is still: 10
```

- This output shows that the value of `$sum` in the calling code is not affected by the use of the local variable `$sum` in the function.
- The purpose of the design of local variables is simple: A function should behave the same way, regardless of the context of its use.
- In some cases, it is convenient for the code in a function to be able to access a variable that is defined outside the function.
- For this situation, PHP has the `global` declaration. When a variable is listed in a `global` declaration in a function, that variable is expected to be defined outside the function.
- So, such a variable has the same meaning inside the function as outside. For example, consider the following code:

```
$big_sum = 0;
...
/* Function summer
   Parameter: An array of integers
   Returns: The sum of the elements of the parameter
            array
   Side effect: Add the computed sum to the global,
                $big_sum
*/
function summer ($list) {
  global $big_sum;   //** Get access to $big_sum
  $sum = 0;
  foreach ($list as $value)
    $sum += $value;
  $big_sum += $sum;
  return $sum;
} //** end of summer

...
$ans1 = summer($list1);
$ans2 = summer($list2);
...
print "The sum of all array elements is: $big_sum <br />";
```

- If the `global` declaration were not included in the function, the script would have two variables named `$big_sum`: the global one and the one that is local to the function. Without the declaration, this script cannot do what it is meant to do.
- In PHP, a local variable in a function can be specified to be static by declaring it with the reserved word `static`. Such a declaration can include an initial value, which is only assigned the first time the declaration is reached. For example, the function

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

  displays the number of times it has been called, even if it is called from several different places. The fact that its local variable `$count` is static allows this to be done.

Pattern Matching:

- PHP includes two different kinds of string pattern matching using regular expressions:
  - one that is based on POSIX regular expressions, and
  - one that is based on Perl regular expressions, like those of JavaScript.
- The POSIX regular expressions are compiled into PHP, but the Perl-Compatible Regular Expression (PCRE) library must be compiled before Perl regular expressions can be used.
- The PHP function `preg_match` is popularly used for pattern matching.
- For example:

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

- The `preg_split` function operates on strings but returns an array and uses patterns.
- For example:

```
$fruit_string = "apple : orange : banana";
$fruits = preg_split("/ : /", $fruit_string);
```

  The array `$fruits` now has `("apple", "orange", "banana").`

The following example illustrates the use of `preg_split` on text to parse out the words and produce a frequency-of-occurrence table:

```
<!-- 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
       white space or punctuation, possibly followed by white space.
       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 parameter string into words
  $words = preg_split("/[ \.,;:!\?]\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;
} #** End of splitter
```

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

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

// Display the words 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>
```

OUTPUT:

```
Word Frequency

apples 3
are 1
better 1
don't 1
for 1
good 1
like 2
maybe 1
or 2
oranges 1
than 1
you 3
```

Form Handling in PHP:

- One common way for a browser user to interact with a Web server is through forms.
- A form is presented to the user, who is invited to fill in the text boxes and click the buttons of the form.
- The user submits the form to the server by clicking the form's Submit button.
- The contents of the form are encoded and transmitted to the server, which must use a program to decode the contents, perform whatever computation is necessary on the data, and produce output.
- When PHP is used to process form data, it implicitly decodes the data.

- When PHP is used for form handling, the PHP script is embedded in an XHTML document, as it is with other uses of PHP.
- Although it is possible to have a PHP script handle form data in the same XHTML document that defines the form, it is perhaps clearer to use two separate documents.
- For this latter case, the document that defines the form specifies the document that handles the form data in the `action` attribute of its `<form>` tag.
- PHP can be configured so that form data values are directly available as implicit variables whose names match the names of the corresponding form elements.
- The recommended approach is to use the implicit arrays `$_POST` and `$_GET` for form values.
- These arrays have keys that match the form element names and values that were input by the client. For example, if a form has a text box named `phone` and the form method is POST, the value of that element is available in the PHP script:

  ```
  $_POST[“phone”]
  ```

For example:

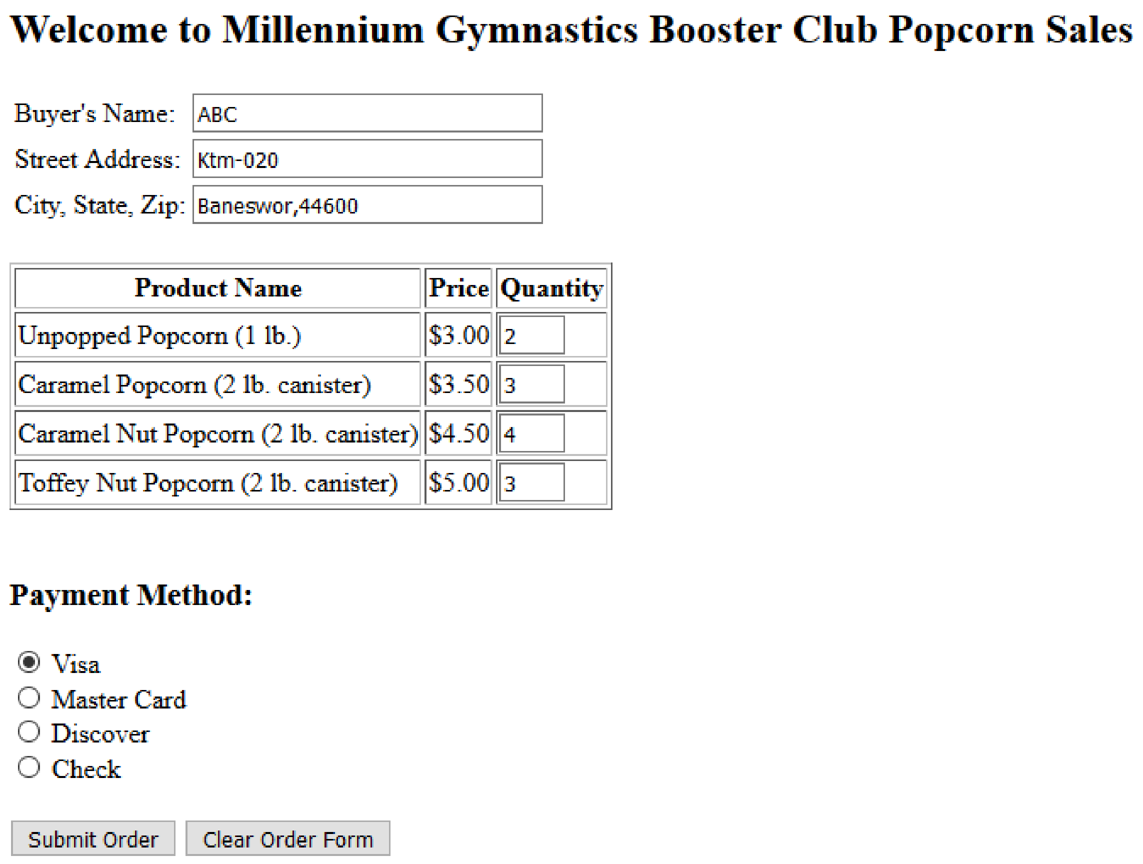
The Popcorn Sales Example:

```
<!-- popcorn.html - This describes popcorn sales form page -->
<html>
<head>
<title> Popcorn Sales CGI program </title>
</head>
<body>

<!-- The next line gives the address of the CGI program -->

<form action = "popcorn3.php" method = "POST">
      <h2> Welcome to Millennium Gymnastics Booster Club Popcorn
Sales </h2>

      <table>

      <!-- Text widgets for name and address -->

          <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>
```

```
 <br/>

 <table border = "border">

 <!-- First, the column headings -->

    <tr>
        <th> Product Name </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
="2"> </td>
    </tr>
    <tr>
        <td> Caramel Popcorn (2 lb. canister) </td>
        <td> $3.50 </td>
        <td> <input type = "text"  name = "caramel" size =
"2"> </td>
    </tr>
    <tr>
        <td> Caramel Nut Popcorn (2 lb. canister) </td>
        <td> $4.50 </td>
        <td> <input type = "text"  name = "caramelnut" size
= "2"></td>
    </tr>
    </tr>
        <td> Toffey Nut Popcorn (2 lb. canister) </td>
        <td> $5.00 </td>
        <td> <input type = "text"  name = "toffeynut" size
= "2"> </td>
    </tr>

 </table>
 <br/>

<!-- The radio buttons for the payment method -->

<h3> Payment Method: </h3>
<input type = "radio” name = "payment” value = "visa"
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">

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

Output of popcorn.html(with form fill-up):



Welcome to Millennium Gymnastics Booster Club Popcorn Sales

Buyer's Name:	ABC
Street Address:	Ktm-020
City, State, Zip:	Baneswor,44600

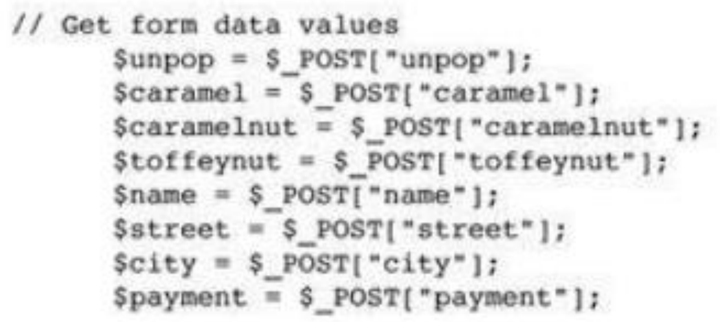
Product Name	Price	Quantity
Unpopped Popcorn (1 lb.)	$3.00	2
Caramel Popcorn (2 lb. canister)	$3.50	3
Caramel Nut Popcorn (2 lb. canister)	$4.50	4
Toffey Nut Popcorn (2 lb. canister)	$5.00	3

Payment Method:

[✓ Visa]
Master Card
Discover
Check

The script that handles the popcorn form is given below.

```
<(!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 strict//EN"
"http://www.w3.org/TR/xhtml1/dtd/xhtml1-strict.dtd">

<!--  popcorn3.php
      Processes the data collected from popcorn3.html -->

<html>
<head>
<title>
      Procees the Propcorn order form
</title>
</head>
<body>
<?php
```



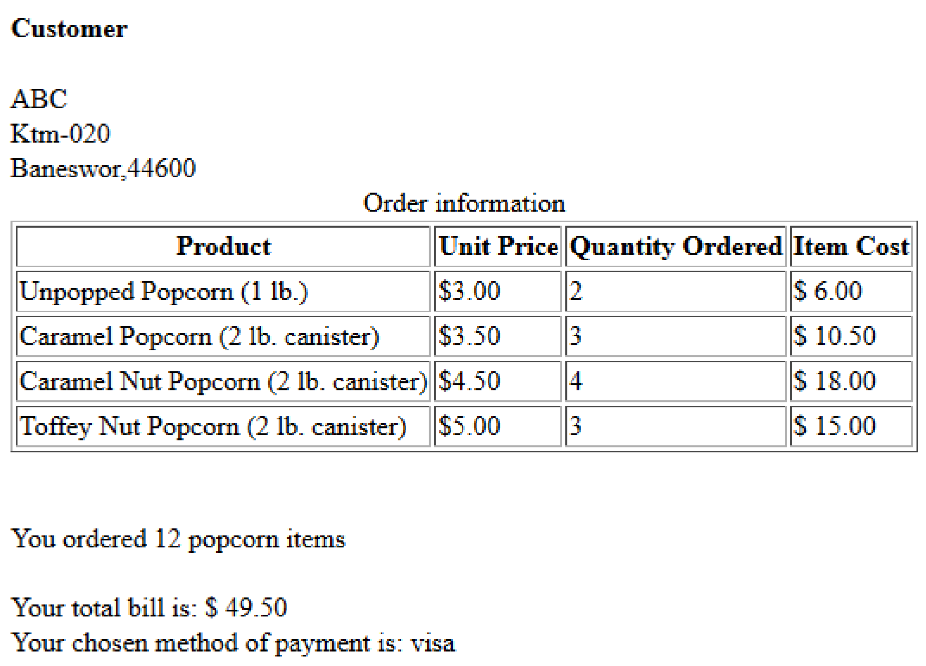
```
 // Get form data values
      $unpop = $_POST["unpop"];
      $caramel = $_POST["caramel"];
      $caramelnut = $_POST["caramelnut"];
      $toffeynut = $_POST["toffeynut"];
      $name = $_POST["name"];
      $street = $_POST["street"];
      $city = $_POST["city"];
      $payment = $_POST["payment"];
```

```
// If any of the quatities are blank, set them to zero

if ($unpop == "") $unpop = 0;
if ($caramel == "") $caramel = 0;
if ($caramelnut == "" ) $caramelnut = 0;
if ($toffeynut == "") $toffeynut =0;

//Compute the item costs and total cost

$unpop_cost = $unpop * 3.0;
$caramel_cost =  $caramel *3.5;
$caramelnut_cost = 4.5 * $caramelnut ;
$toffeynut_cost = 5.0 * $toffeynut;

$total_price = $unpop_cost + $caramel_cost +
      $caramelnut_cost + $toffeynut_cost ;

$total_items = $unpop + $caramel + $caramelnut + $toffeynut;

//Return results to the browser in a table
```

```
?>

<h4>Customer</h4>
<?php
print ("$name <br /> $street <br /> $city <br />");
?>

<table border = border">
      <caption>Order information</caption>
      <tr>
            <th>Product</th>
            <th>Unit Price </th>
            <th>Quantity Ordered</th>
            <th>Item Cost</th>
      </tr>
      <tr aling = "center">
            <td> Unpopped Popcorn (1 lb.) </td>
            <td> $3.00 </td>
            <td> <?php print ("$unpop");?> </td>
            <td> <?php printf ("$ %4.2f", $unpop_cost); ?> </td>
      </tr>
      <tr aling = "center">
            <td> Caramel Popcorn (2 lb. canister) </td>
            <td> $3.50 </td>
            <td> <?php print ("$caramel");?> </td>
               <td> <?php printf ("$ %4.2f", $caramel_cost); ?>
</td>
        </tr>
      <tr aling = "center">
            <td> Caramel Nut Popcorn (2 lb. canister) </td>
            <td> $4.50 </td>
            <td> <?php print ("$caramelnut");?> </td>
               <td> <?php printf ("$ %4.2f", $caramelnut_cost);
?> </td>
        </tr>


      <tr aling = "center">
            <td> Toffey Nut Popcorn (2 lb. canister) </td>
            <td> $5.00 </td>
            <td> <?php print ("$toffeynut");?> </td>
               <td> <?php printf ("$ %4.2f", $toffeynut_cost);
?> </td>
        </tr>
</table>
<br /><br />
```

```
<?php
print ("You ordered $total_items popcorn items <br/> <br/> \n");
printf ("Your total bill is: \$ %5.2f <br />", $total_price);
print ("Your chosen method of payment is: $payment <br />");
?>
</body>
</html>
```

Output of popcorn3.php



Customer

ABC
Ktm-020
Baneswor,44600

Order information

Product	Unit Price	Quantity Ordered	Item Cost
Unpopped Popcorn (1 lb.)	$3.00	2	$ 6.00
Caramel Popcorn (2 lb. canister)	$3.50	3	$ 10.50
Caramel Nut Popcorn (2 lb. canister)	$4.50	4	$ 18.00
Toffey Nut Popcorn (2 lb. canister)	$5.00	3	$ 15.00

You ordered 12 popcorn items

Your total bill is: $ 49.50
Your chosen method of payment is: visa

Form Validation with PHP:

PHP - Validate Name

The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);
if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
  $nameErr = "Only letters and white space allowed";
}
```

PHP - Validate E-mail

The easiest and safest way to check whether an email address is well-formed is to use PHP's filter_var() function.

In the code below, if the e-mail address is not well-formed, then store an error message:

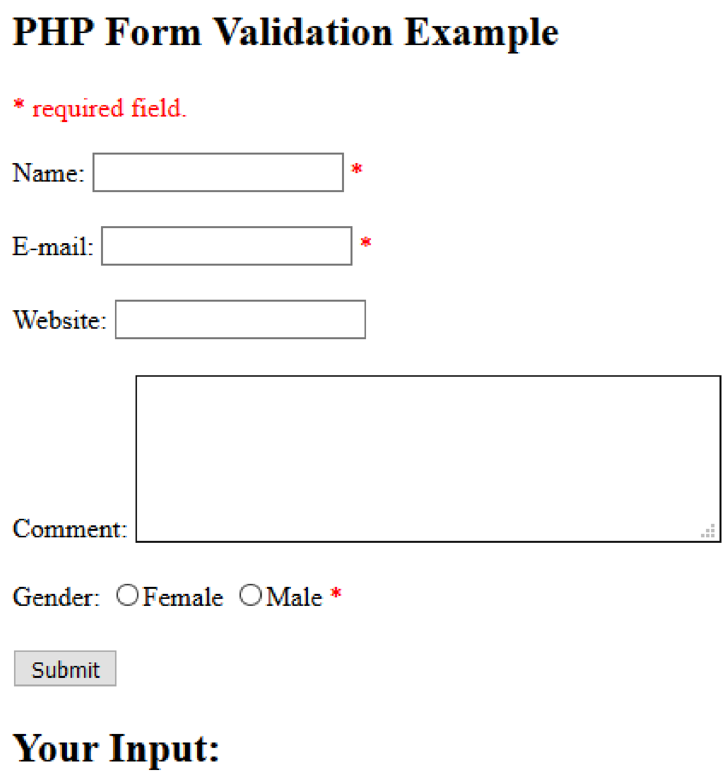
```
$email = test_input($_POST["email"]);
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
  $emailErr = "Invalid email format";
}
```

PHP - Validate URL

The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);
if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
  $websiteErr = "Invalid URL";
}
```

PHP Complete Form with Validation:



PHP Form Validation Example

* required field.

Name: *

E-mail: *

Website:

Comment:

Gender: ○Female ○Male *

Submit

Your Input:

PHP code:

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression also allows dashes in
the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~_|!:,.;]*[-a-z0-
9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>

<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field.</span></p>
<form method="post" action="<?php echo
htmlspecialchars($_SERVER["PHP_SELF"]);?>">
  Name: <input type="text" name="name" value="<?php echo $name;?>">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email" value="<?php echo $email;?>">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website" value="<?php echo $website;?>">
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"><?php echo
$comment;?></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" <?php if (isset($gender) && $gender=="female")
echo "checked";?> value="female">Female
  <input type="radio" name="gender" <?php if (isset($gender) && $gender=="male")
echo "checked";?> value="male">Male
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

File Handling:

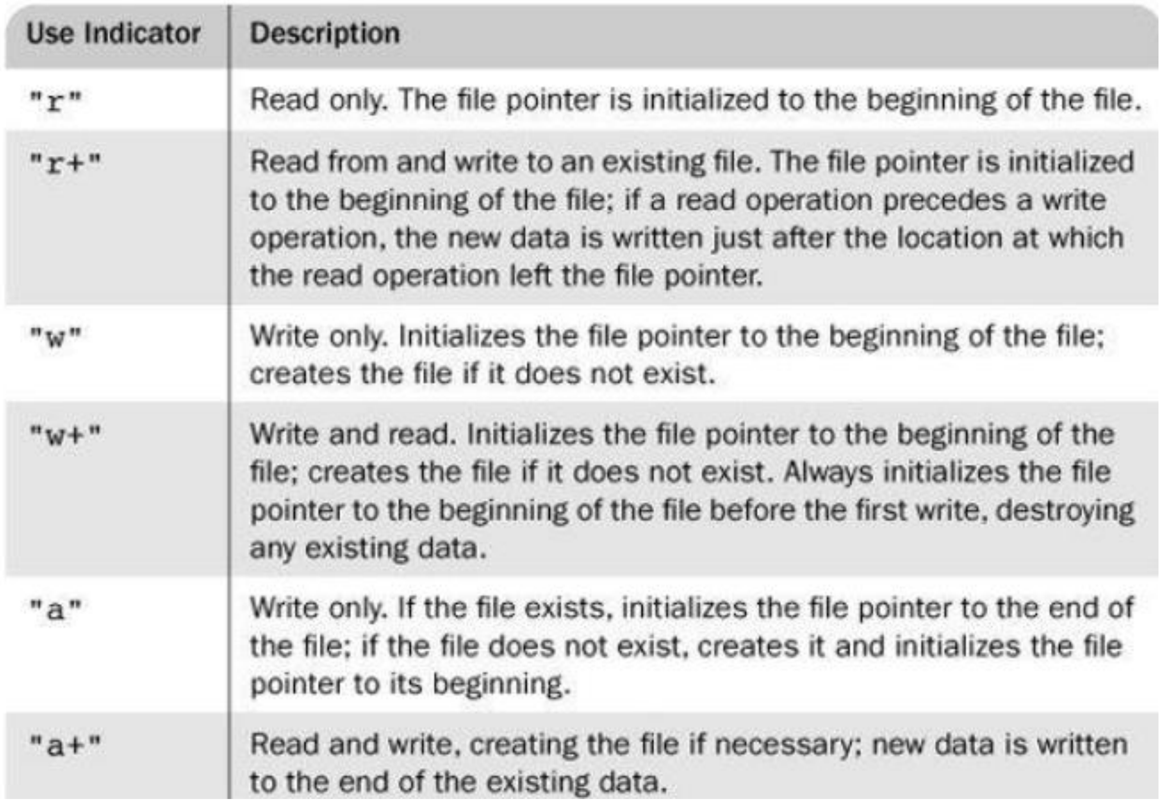
Files:

- Because PHP is a server-side technology, it is possible to create, read, and write files on the server system using it.
- In fact, PHP can deal with files residing on any server system on the Internet, using both HTTP and FTP protocols.

Opening and Closing Files:

- The first step in some file operations is to open the file, a process that prepares the file for use and associates a program variable with the file for future reference.
- This program variable is called the file variable.
- The `fopen` function performs these operations.
- It takes two parameters: the file name, including the path to it if it is in a different directory, and a use indicator, which specifies the operation or operations that will be performed on the file. Both parameters are given as strings.
- The `fopen` function returns the reference to the file for the file variable. Every open file has an internal pointer that is used to indicate where the next operation should take place within the file. It is called the file pointer.

File Use Indicators



Use Indicator	Description
"r"	Read only. The file pointer is initialized to the beginning of the file.
"r+"	Read from and write to an existing file. The file pointer is initialized to the beginning of the file; if a read operation precedes a write operation, the new data is written just after the location at which the read operation left the file pointer.
"w"	Write only. Initializes the file pointer to the beginning of the file; creates the file if it does not exist.
"w+"	Write and read. Initializes the file pointer to the beginning of the file; creates the file if it does not exist. Always initializes the file pointer to the beginning of the file before the first write, destroying any existing data.
"a"	Write only. If the file exists, initializes the file pointer to the end of the file; if the file does not exist, creates it and initializes the file pointer to its beginning.
"a+"	Read and write, creating the file if necessary; new data is written to the end of the existing data.

- It is possible for the `fopen` function to fail—for example, if an attempt is made to open a file for reading but no such file exists.
- The function would also fail if the file access permissions did not allow the requested use of the file.
- The `fopen` function returns `FALSE` if it fails.
- The `die` function produces a message and stops the interpretation process.
- It is often used with input and output operations, which sometimes fail. For example, the following statement attempts to open a file named `testdata.dat` for reading only, but calls `die` if the open operation fails:

```
$file_var = fopen("testdata.dat", "r") or
            die ("Error — testdata.dat cannot be opened");
```

- The problem of `fopen` failing because the specified file does not exist can be avoided by determining whether the file exists with `file_exists` before calling `fopen`.
- The `file_exists` function takes a single parameter: the file's name. It returns `TRUE` if the file exists, `FALSE` otherwise.
- A file is closed with the `fclose` function, which takes a file variable as its only parameter.

Reading from a File:

- The `fread` function reads part or all of a file and returns a string of what was read.

- This function takes two parameters: a file variable and the number of bytes to be read.
- The reading operation stops when either the end-of-file marker is read or the specified number of bytes has been read.
- A call to `filesize` is often used as the second parameter to `fread`, to find the size of the reading file.
- For example:

```
$file_string = fread($file_var,
                     filesize("testdata.dat"));
```

- One alternative to `fread` is `file`, which takes a file name as its parameter and returns an array of all of the lines of the file. One advantage of `file` is that the file open and close operations are not necessary.
- PHP has another file input function that does not require calling `fopen`: The function `file_get_contents` takes the file's name as its parameter. This function reads the entire contents of the file, as exemplified in the following call:
  $file_string = file_get_contents("testdata.dat");
- A single line of a file can be read with `fgets`, which takes two parameters: the file variable and a limit on the length of the line to be read. As an example, the statement:
  $line = fgets($file_var, 100);

  reads characters from the file whose file variable is `$file_var` until it finds a newline character, encounters the end-of-file marker, or has read 99 characters.

Writing a File:

- The `fwrite` function takes two parameters: a file variable and the string to be written to the file. The `fwrite` function returns the number of bytes written. The following is an example of a call to `fwrite`:
  $bytes_written = fwrite($file_var, $out_data);
  This statement writes the string value in `$out_data` to the file referenced with `$file_var` and places the number of bytes written in `$bytes_written`.
- The `file_put_contents` function is the counterpart of `file_get_contents`—it writes the value of its second parameter, a string, to the file specified in its first parameter. For example, consider the following call:
  file_put_contents("savedata.dat", $str);

Locking Files:

- If it is possible for more than one script to access a file at the same time, the potential interference of those accesses can be prevented with a file lock, which prevents any other access to the file while the lock is set.
- Scripts that use such files lock them before accessing them and unlock them when the access is completed. File locking
  is done in PHP with the `flock` function.

- The function takes two parameters: the file variable of the file and an integer that specifies the particular operation.
- A value of `1` specifies that the file can be read by others while the lock is set, a value of `2` allows no other access, and a value of `3` unlocks the file.

<u>PHP File Upload:</u>

Configure The "php.ini" File

First, ensure that PHP is configured to allow file uploads.

In your "php.ini" file, search for the `file_uploads` directive, and set it to On:

```
file_uploads = On
```

Create The HTML Form to choose file to upload:

```
<!DOCTYPE html>
<html>
<body>
<form action="upload.php" method="post" enctype="multipart/form-data">
   Select image to upload:
   <input type="file" name="fileToUpload" id="fileToUpload">
   <input type="submit" value="Upload Image" name="submit">
</form>

</body>
</html>
```

Some rules to follow for the HTML form above:

- Make sure that the form uses method="post"
- The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form

Without the requirements above, the file upload will not work.

Other things to notice:

- The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

Create The Upload File PHP Script

The "upload.php" file contains the code for uploading a file:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
   $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
   if($check !== false) {
      echo "File is an image - " . $check["mime"] . ".";
      $uploadOk = 1;
   } else {
      echo "File is not an image.";
      $uploadOk = 0;
   }
}
?>
```

PHP script explained:

- $target_dir = "uploads/" - specifies the directory where the file is going to be placed
- $target_file specifies the path of the file to be uploaded
- $uploadOk=1 is not used yet (will be used later)
- $imageFileType holds the file extension of the file (in lower case)
- Next, check if the image file is an actual image or a fake image

Check if File Already Exists

Now we can add some restrictions.

First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and $uploadOk is set to 0:

```
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
```

Limit File Size

The file input field in our HTML form above is named "fileToUpload".

Now, we want to check the size of the file. If the file is larger than 500KB, an error message is displayed, and $uploadOk is set to 0:

```
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
```

Limit File Type

The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting $uploadOk to 0:

```
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" &&
$imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are
allowed.";
    $uploadOk = 0;
}
```

The complete “upload.php”:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
   $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
   if($check !== false) {
      echo "File is an image - " . $check["mime"] . ".";
      $uploadOk = 1;
   } else {
      echo "File is not an image.";
      $uploadOk = 0;
   }
}
// Check if file already exists
if (file_exists($target_file)) {
   echo "Sorry, file already exists.";
   $uploadOk = 0;
}
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
   echo "Sorry, your file is too large.";
```

```
    $uploadOk = 0;
}
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $target_file)) {
        echo "The file ". basename( $_FILES["fileToUpload"]["name"]). " has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>
```

Session and Cookies:

- A session is the time span during which a browser interacts with a particular server.
- A session begins when a browser connects to the server.
- That session ends either when the browser is terminated or because the server terminated the session because of client inactivity.
- The length of time a server uses as the maximum time of inactivity is set in the configuration of the server. For example, the default maximum for the Tomcat server is 30 minutes.
- The HTTP protocol is essentially stateless: It includes no means to store information about a session that is available to a subsequent session.
- However, there are a number of different reasons that it is useful for the server to be capable of connecting a request made during a session to the other requests made by the same client during that session, as well as to requests made during previous and subsequent sessions.
- One of the most common needs for information about a session is to implement shopping carts on Web sites.
- An e-commerce site can have any number of simultaneous online customers. At any time, any customer can add an item to or remove an item from his or her cart. Each user's shopping cart is identified by a session identifier, which could be implemented as a cookie.
- So, cookies can be used to identify each of the customers visiting the site at a given time.
- Besides identifying customers, another common use of cookies is for a Web site to create profiles of visitors by remembering which parts of the site are perused by that visitor which is sometimes called as personalization.

- Later sessions can use such profiles to target advertising to the client in line with the client's past interests.
- Also, if the server recognizes a request as being from a client who has made an earlier request from the same site, it is possible to present a customized interface to that client.
- These situations require that information about clients be accumulated and stored.
- Storing session information is becoming increasingly important as more and more Web sites make use of shopping carts and personalization.
- Cookies provide a general approach to storing information about sessions on the browser system itself.
- The server is given this information when the browser makes subsequent requests for resources from the server.
- Note that some of the uses of cookies require them to be stored after the session in which they were created ends.
- A cookie is a small object of information that includes a name and a textual value.
- A cookie is created by some software system on the server.
- Every HTTP communication between a browser and a server includes a header, which stores information about the message.
- The header part of an HTTP communication can include cookies. So, every request sent from a browser to a server, and every response from a server to a browser, can include one or more cookies.
- At the time it is created, a cookie is assigned a lifetime. When the time a cookie has existed reaches its associated lifetime, the cookie is deleted from the browser's host machine.
- Every browser request includes all of the cookies its host machine has stored that are associated with the Web server to which the request is directed.
- Only the server that created a cookie can ever receive the cookie from the browser, so a particular cookie is information that is exchanged exclusively between one specific browser and one specific server.
- Because cookies allow servers to record browser activities, some consider them to involve privacy concerns.
- Accordingly, browsers allow the client to change the browser setting to refuse to accept cookies from servers. This is clearly a drawback of using cookies—they are useless when clients reject them.
- Cookies also can be deleted by the browser user, although the deletion process is different for different browsers.

<u>PHP Support for Cookie:</u>

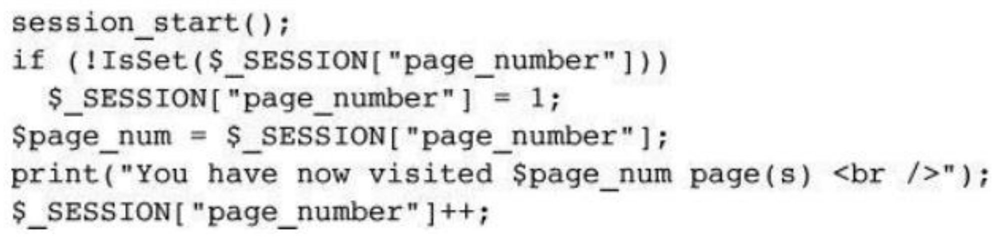
- A cookie is set in PHP with the `setcookie` function.
- This function takes one or more parameters. The first parameter, which is mandatory, is the cookie's name
  given as a string. The second, if present, is the new value for the cookie, also a string.
- If the value is absent, `setcookie` undefines the cookie.
- The third parameter, when present, is the expiration time in seconds for the cookie, given as an integer.

- The default value for the expiration time is zero, which specifies that the cookie is destroyed at the end of the current session.
- When specified, the expiration time is often given as the number of seconds in the UNIX epoch, which began on January 1, 1970.
- The `time` function returns the current time in seconds. So, the cookie expiration time is given as the value returned from `time` plus some number.
- For example, consider the following call to `setcookie`:
  setcookie("voted", "true", time() + 86400);
- This call creates a cookie named `"voted"` whose value is `"true"` and whose lifetime is one day (86,400 is the number of seconds in a day).
- Creating a cookie or setting a cookie to a new value is that it must be done before any other XHTML is created by the PHP document.
- It is because cookies are stored in the HTTP header of the document returned to the requesting browser. The HTTP header is sent before the body of the document is sent. The server sends the header when it receives the first character of the body of the document.
- If we create a cookie or change the value of a cookie after even a single character of document body has been generated, the cookie operation will not be successful.
- The other cookie operation is getting the cookies and their values from subsequent browser requests.
- In PHP, cookie values are treated much as are form values.
- All cookies that arrive with a request are placed in the implicit `$_COOKIES` array, which has the cookie names as keys and the cookie values as values.
- A PHP script can test whether a cookie came with a request by using the `IsSet` predicate function on the associated variable.
- It is to be noted that cookies cannot be dependent upon because some users set their browsers to reject all cookies. Furthermore, most browsers have a limit on the number of cookies that will be accepted from a particular server site.

Session Tracking:

- Rather than using one or more cookies, a single session array can be used to store information about the previous requests of a client during a session.
- In particular, session arrays often store a unique session ID for a session.
- One significant way that session arrays differ from cookies is that they can be stored on the server, whereas cookies are stored on the client.
- In PHP, a session ID is an internal value that identifies the session.
- Session IDs need not be known or handled in any way by PHP scripts.
- PHP is made aware that a script is interested in session tracking by calling the `session_start` function, which takes no parameters.
- The first call to `session_start` in a session causes a session ID to be created and recorded.
- On subsequent calls to `session_start` in the same session, the function retrieves the `$_SESSION` array, which
  stores any session variables and their values that were registered in previously executed scripts in that session.

- Session key—value pairs are created or changed by assignments to the `$_SESSION` array. They can be destroyed with the `unset` operator.
- For example:



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

- If this is not the first document visited that calls `session_start` and sets the `page_number` session variable, the script that it executes will produce the specified line with the last set value of `$_SESSION["page_number"]`.
- If no document that was previously visited in this session set `page_number`, the script sets `page_number` to `1`, produces the line,

You have now visited 1 page(s)

and increments `page_number`.

<u>Differences between Cookie and Session:</u>

Cookies

- Cookies are stored in browser as text file format.
- It is stored limit amount of data. It is only allowing 4kb[4096bytes]
- It is not holding the multiple variable in cookies.
- We can accessing the cookies values in easily. So it is less secure.
- The setcookie() function must appear BEFORE the tag.

Destroy Cookies:

- If we Closing the browsers at the time.
- Setting the cookie time to expire the cookie.

Example:

```
<?php

setcookie(name, value, expire, path, domain, secure,
httponly);
$cookie_uame = "codingslover";
$cookie_uvalue = "website";
```

```
//set cookies for 1 hour time
setcookie($cookie_uname, $cookie_uvalue, 3600, "/");

//expire cookies
setcookie($cookie_uname,"",-3600);

?>
```

Sessions

- Sessions are stored in server side.
- It is stored unlimited amount of data.
- It is holding the multiple variable in sessions.

Destroy Sessions :

- Using unset() session, we will destroyed the sessions.
- Using session_destory(), we we will destroyed the sessions.

Example:

```
<?php

session_start();

//session variable
$_SESSION['testvaraible'] = 'Codings';

//destroyed the entire sessions
session_destroy();

//Destroyed the session variable "testvaraible".
unset($_SESSION['testvaraible']);

?>
```

PHP and Database Access:

- PHP includes support for a wide variety of database systems.
- For each database system supported, there is an associated API. These APIs provide the interface to the
specific systems. For example, the MySQL API includes functions to connect to a database and apply SQL commands against the database.
- Web access to a database with the use of PHP is a natural architecture because PHP scripts are called through HTML documents from browsers.

MySQL Database System:

- MySQL is a free, efficient, widely used database system that implements SQL.
- It is available for all popular computing platforms.
- The first step in using MySQL is logging into the MySQL system, which is done with the following command (at the command line of the operating system):
  mysql [-h *host*] [-u *username*] [*database_name*] [-p]
- The parts of this command that are in square brackets are optional. The host is the name of the server running MySQL; if host is absent, MySQL assumes that it denotes the user's machine. If username is absent, MySQL assumes that the name you used to log onto the machine is the correct username. If database_name is given, that database is selected as the focus of MySQL, making it the object of subsequent commands. If `-p` is included, a password is required, and MySQL will ask for it.
- Once you have successfully logged into MySQL, it is ready to receive commands. Although it is called "logging on," what you are actually doing is starting execution of the MySQL system.
- If the database to be accessed already exists, but its name was not included in the logon to MySQL, the `use` command can be used to focus on the database of interest. For example, if we want to access a database named `cars`, the following command would be used:
  use cars;
- This is sometimes called making a specific database the "current" database for the MySQL server. The MySQL response to this command is as follows:
  Database changed
- Note the semicolon at the end of the `use` command; it is essential, as it is for all MySQL commands.
- If a command is given without a semicolon, MySQL will wait indefinitely for one. Until a semicolon is found, MySQL behaves as if the remainder of the command is yet to be typed.
- If a database is not specified in the logon to MySQL and a database command is given before `use` is used to focus on a database, the following error message will be issued:
  ERROR 1046: No Database Selected
- If a new database is to be created, the database itself must be created first and then the tables that will make up the database. A new database is created with the SQL `CREATE DATABASE` command:
  CREATE DATABASE cars;
- This command also elicits an odd response from MySQL:
  Query ok, 1 row affected (0.05 sec)
- The time given varies with the speed of the host machine and its current load.
- The tables of a database are created with the `CREATE TABLE` command, whose syntax is that of SQL. For example, in the `INT` and `UNSIGNED` parts of the `Equip_id` column indicate the data type.

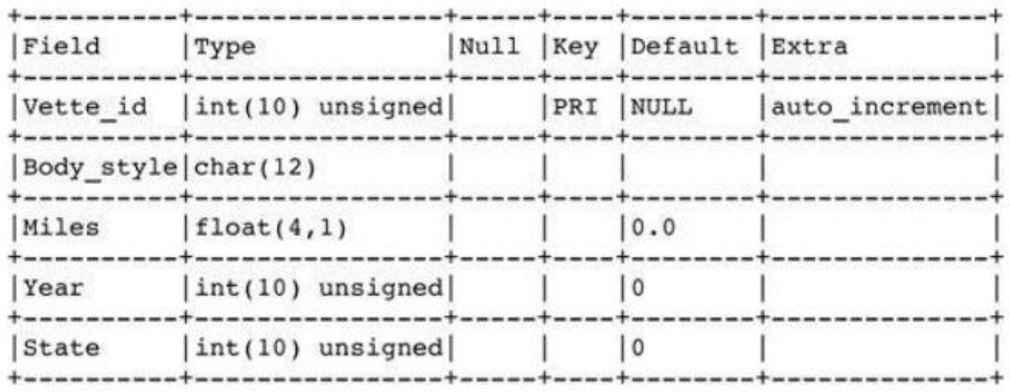
```
CREATE TABLE Equipment
   (Equip_id  INT  UNSIGNED  NOT NULL  AUTO_INCREMENT
              PRIMARY KEY,
    Equip  CHAR(10)
   );
```

- The `AUTO_INCREMENT` is a MySQL convenience. It specifies that the values of this column need not be given in populating the table. The values `1`, `2`, `3`, and so forth will be implicitly assigned.
- The value `NULL` is given in place of a value for a column so specified in populating the table with `INSERT`.
- A large number of different data types are possible for field values.
- The most common of these are `CHAR(`length`)`, `INT`, and `FLOAT(`total, fractional`)`, where total specifies the total
  number of characters, including both digits and the decimal point, and fractional gives the number of digits to the right of the decimal point.
- The `SHOW` command can be used to display the tables of the database:
  SHOW TABLES;

  If our sample database, `cars`, is the database of current focus, the preceding command produces the following output:

```
--------------
show
--------------
+---------------------+
|                     |
| Tables_in_cars      |
|                     |
+---------------------+
| Corvettes           |
| Corvettes_Equipment |
| Equipment           |
| States              |
+---------------------+
```

- The `DESCRIBE` command can be used to display the description of the structure of a table. For example,
  DESCRIBE Corvettes;
  produces the following table:



```
+----------+---------------+-----+----+--------+--------------+
|Field     |Type           |Null |Key |Default |Extra         |
+----------+---------------+-----+----+--------+--------------+
|Vette_id  |int(10) unsigned|    |PRI |NULL    |auto_increment|
+----------+---------------+-----+----+--------+--------------+
|Body_style|char(12)       |     |    |        |              |
+----------+---------------+-----+----+--------+--------------+
|Miles     |float(4,1)     |     |    |0.0     |              |
+----------+---------------+-----+----+--------+--------------+
|Year      |int(10) unsigned|    |    |0       |              |
+----------+---------------+-----+----+--------+--------------+
|State     |int(10) unsigned|    |    |0       |              |
+----------+---------------+-----+----+--------+--------------+
```

- ➢ The other MySQL commands that are needed here—`INSERT`, `SELECT`, `DROP`, `UPDATE`, and `DELETE`
- ➢ `Popularly, CRUD (CREATE, RETRIEVE, UPDATE & DELETE) operation are used for frequent database access by most of the server side scripting language.`

Database Access with PHP and MySQL:

- ➢ PHP access to a database is often done with two HTML documents: one to collect a user request for a database access and one to host the PHP code to process the request and generate the return HTML document.
- ➢ The user request collector is a simple HTML document.

Potential Problems with Special Characters:

- ➢ When a query is made on a database through a browser, the result of the query must be returned to the browser as HTML.
- ➢ Putting database field data into an HTML document creates a potential problem.
- ➢ A field retrieved from the database may contain characters that are special in HTML, namely >, <, ‟, or `&`.
- ➢ PHP includes a function, `htmlspecialchars`, that replaces all occurrences of these four special characters in its parameter with their corresponding entities. For example,
  consider the following code:
  $str = “Apples & grapes <raisins, too>“;
  $str = htmlspecialchars($str);

  After the interpretation of this code, the value of `$str` is as follows:
  “Apples &amp; grapes &lt;raisins, too&gt;”

  This string is now ready to be made the content of an HTML tag without causing any browser confusion

- Another problem with special characters can occur with PHP scripts that get values through `GET` or `POST` or from a cookie.
- Strings from these sources could include single quotes, double quotes, backslashes, and null characters, all of which could cause problems if they are used in other strings in a script.
- To avoid these problems, the PHP system has an implicit backslashing function named `magic_quotes_gpc`, which can be turned on or off in the `PHP.ini` file.
- When this function is enabled, which is the default, all values received in a script from `$_POST`, `$_GET`, and `$_COOKIE` have backslashes implicitly inserted in front of all single quotes, double quotes, backslashes, and null characters.
- This strategy avoids any problems that could be caused by those characters.
- For example, if the string `O'Reilly` is fetched from `$_POST`, it would be converted by `magic_quotes_gpc` to `O\'Reilly`.
- Unfortunately, this causes other problems. If the script compares the name with a nonslashed version, the comparison will fail. Furthermore, even displaying the name will show the backslash.
- This problem is relevant here because we want to have a PHP script get SQL commands from a text box in an XHTML document.
- For example, suppose `magic_quotes_gpc` is on and the value for a query obtained from a text box on a form is as follows:
  SELECT * FROM Corvettes WHERE Body_style = ‘coupe’

  If the name of the text box is `query`, its value is put in `$query` with the following statement:
  $query = $_POST[‘query’];
  The value of `$query` is converted to the following by `magic_quotes_gpc`:
  SELECT * FROM Corvettes WHERE Body_style = \’coupe\’

- Unfortunately, this string is not a valid SQL command (because of the backslashes). If it is sent to MySQL as a command, MySQL will reject it and report an error.
- Therefore, if complete SQL commands are to be collected from a form, `magic_quotes_gpc` must be disabled in `PHP.ini` to avoid the extra backslashes.
- The alternative to changing the value of `magic_quotes_gpc` is to remove the extra slashes in the PHP script with the predefined function `stripslashes`, as in the following statement:
  $query = stripslashes($query);

Connecting to MySQL and Selecting a Database:

Note: PHP 5 and later can work with a MySQL database using:

- **MySQLi extension** (the "i" stands for improved)
- **PDO (PHP Data Objects)**

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

PDO and MySQLi Comparison:

- PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.
- So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.
- Both are object-oriented, but MySQLi also offers a procedural API.
- Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

MySQL Connection:

Traditional Approach:

- The PHP function `mysql_connect` connects a script to a MySQL server.
- This function takes three parameters, all of which are optional. The first is the host that is running MySQL; the default is localhost (the machine on which the script is running). The second parameter is the username for MySQL; the default is the username in which the PHP process runs. The third parameter is the password for the database; the default is blank (which works if the database does not require a password).

  For example, if the default parameters were acceptable, the following statement could be used:
  $db = mysql_connect();

  Of course, the connect operation could fail, in which case the value returned would be `false` (rather than a reference to the database).
- Therefore, the call to `mysql_connect` usually is used in conjunction with `die`.
- The connection to a database is terminated with the `mysql_close` function.
- This function is not necessary when MySQL is used through a PHP script, because the connection will be closed implicitly when the script terminates.
- When running MySQL from the command line, a database must be selected as the current, or focused, database. This is also necessary when MySQL is used through PHP; it is accomplished with the `mysql_select_db` function, as in the following

```
call:
mysql_select_db("cars");
```

Modern Approach (using MySQLi):

Example (MySQLi Object-Oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

Close the connection:

The connection will be closed automatically when the script ends. To close the connection before, use the following:

```
$conn->close();
```

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

Close the connections:

```
mysqli_close($conn);
```

Create a MySQL Database

The CREATE DATABASE statement is used to create a database in MySQL.

The following examples create a database named "myDB":

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
   echo "Database created successfully";
} else {
   echo "Error creating database: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Create a MySQL Table under database

The CREATE TABLE statement is used to create a table in MySQL.

We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
```

```
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
   echo "Table MyGuests created successfully";
} else {
   echo "Error creating table: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

In the above example, after the data type, you can specify other optional attributes for each column:

- NOT NULL - Each row must contain a value for that column, null values are not allowed
- DEFAULT value - Set a default value that is added when no other value is passed
- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
- AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
- PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.

MySQL Insert Data:

After a database and a table have been created, we can start adding data in them.

Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

The INSERT INTO statement is used to add new records to a MySQL table:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
   echo "New record created successfully";
} else {
   echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Insert Multiple Records Into MySQL

Multiple SQL statements must be executed with the **`mysqli_multi_query()`** function.

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
```

```
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if (mysqli_multi_query($conn, $sql)) {
    echo "New records created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Alternative way to insert multiple data:

Query:
```
INSERT INTO TABLE1(COLUMN1, COLUMN2, ....) VALUES (VALUE1, VALUE2..)
```

Array Conversion to Statements:

```
$columns = implode(", ",array_keys($insData));
$escaped_values = array_map('mysqli_real_escape_string',
array_values($insData));
$values  = implode(", ", $escaped_values);
$sql = "INSERT INTO `fbdata`($columns) VALUES ($values)";
```

Where,

```
$insData = array(

        $key=>$value

);
```

Select Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

SELECT column_name(s) FROM table_name

or we can use the * character to select ALL columns from a table:

SELECT * FROM table_name

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
   // output data of each row
   while($row = mysqli_fetch_assoc($result)) {
      echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
   }
} else {
   echo "0 results";
}

mysqli_close($conn);
?>
```

Delete Data From a MySQL Table

The DELETE statement is used to delete records from a table:

```
DELETE FROM table_name
WHERE some_column = some_value
```

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30
3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if (mysqli_query($conn, $sql)) {
   echo "Record deleted successfully";
} else {
   echo "Error deleting record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

After the record is deleted, the table will look like this:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

PHP Update Data in MySQL

The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

The following examples update the record with id=2 in the "MyGuests" table:

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if ($conn->query($sql) === TRUE) {
  echo "Record updated successfully";
} else {
  echo "Error updating record: " . $conn->error;
}

$conn->close();
?>
```

After the record is updated, the table will look like this:

id	firstname	lastname	email	reg_date

1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Doe	mary@example.com	2014-10-23 10:22:30

Get ID of The Last Inserted Record

If we perform an INSERT or UPDATE on a table with an AUTO_INCREMENT field, we can get the ID of the last inserted/updated record immediately.

$last_id = mysqli_insert_id($conn);

Limit Data Selections From a MySQL Database

MySQL provides a LIMIT clause that is used to specify the number of records to return.

The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance.

Assume we wish to select all records from 1 - 30 (inclusive) from a table called "Orders". The SQL query would then look like this:

$sql = "SELECT * FROM Orders LIMIT 30";

SQL Join:

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Example:

- Table: **Orders**

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

- Table: **Customers**

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

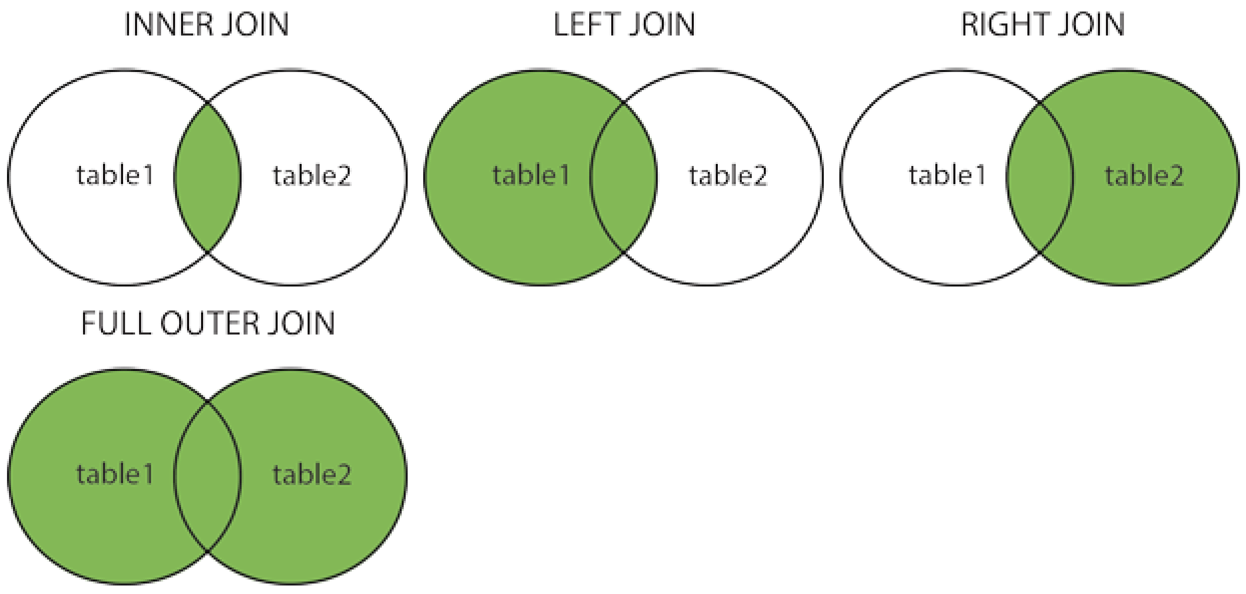
Will produce output:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table


INNER JOIN
table1
table2
LEFT JOIN
table1
table2
RIGHT JOIN
table1
table2
FULL OUTER JOIN
table1
table2