SEMESTER – VI - III BSc IT

18BIT61C - SCRIPTING TOOLS

Prepared by Dr.N.Thenmozhi

UNIT II: Working with string & numeric functions: using string functions, using numeric functions. Working with arrays: creating an array – Modifying array element – using arrays with forms-working with array functions-working with date and time-creating user defined functions. Working with forms: creating a simple Input form –Accessing from - combining HTML and PHP code on single page –using hidden fields to save state –redirecting the user – working with file uploads.

TEXT BOOKS

1. Vikram Vaswani, "PHP: A beginners guide", TMH Hill, 1stedition, 2010 (Unit-I to IV).
2. Steve Suehring, Tim Converse, Joyce Park , "PHP 6 and MySQL 6 Bible", Wiley India pvt. Ltd., Edition, 2009 (Unit – V).

REFERENCE BOOKS

1. Matt Doyle, " Beginning PHP 5.3", Wiley India pvt. Ltd, First edition, 2010.
2. Luke welling and Laura Thomson, "PHP and MySQL Web Development", 5th Edition, 2016.

2.1 Working with string & numeric functions

PHP has over 75 built-in string manipulation functions, supporting operations ranging from string repetition and reversal to comparison and search-and-replace.

2.1.1.Using String Functions

empty()	Tests if a string is empty
strlen()	Calculates the number of characters in a string
strrev()	Reverses a string
str_repeat()	Repeats a string
substr()	Retrieves a section of a string
strcmp()	Compares two strings
str_word_count()	Calculates the number of words in a string
str_replace()	Replaces parts of a string
trim()	Removes leading and trailing whitespace from a string
strtolower()	Lowercases a string
strtoupper()	Uppercases a string
ucfirst()	Uppercases the first character of a string
ucwords()	Uppercases the first character of every word of a string
addslashes()	Escapes special characters in a string with backslashes
stripslashes()	Removes backslashes from a string
htmlentities()	Encodes HTML within a string
htmlspecialchars()	Encodes special HTML characters within a string
nl2br()	Replaces line breaks in a string with <br/> elements
html_entity_decode()	Decodes HTML entities within a string
htmlspecialchars_decode()	Decodes special HTML characters within a string
strip_tags()	Removes PHP and HTML code from a string

Here are some examples:

```
<?php
// test if string is empty
// output: true
$str = '';
echo (boolean) empty($str);
// output: true
$str = null;
echo (boolean) empty($str);

// output: true
$str = '0';
echo (boolean) empty($str);
// output: true
unset($str);
echo (boolean)empty($str);
// calculate length of string
// output: 17
$str = 'Welcome to Xanadu';
echo strlen($str);
// reverse string
// output: 'pets llams enO'
$str = 'One small step';
echo strrev($str);
// repeat string
// output: 'yoyoyo'
$str = 'yo';
echo str_repeat($str, 3);
// extract substring
// output: 'come'
$str = 'Welcome to nowhere';
echo substr($str, 3, 4);
// extract substring
// output: 'come here'
$str = 'Welcome to nowhere';
echo substr($str, 3, 5) . substr($str, -4, 4);
// compare strings
$a = "hello";
$b = "hello";
$c = "hEllo";
// output: 0
echo strcmp($a, $b);
// output: 1
echo strcmp($a, $c);
// count words
// output: 5
$str = "The name's Bond, James Bond";
echo str_word_count($str);
// replace '@' with 'at'
// output: 'john at domain.net'
```

```
$str = 'john@domain.net';
echo str_replace('@', ' at ', $str);
// change string case
$str = 'the yellow brigands';
// output: 'The Yellow Brigands'
echo ucwords($str);
// output: 'The yellow brigands'
echo ucfirst($str);
// escape special characters
// output: 'You\'re awake, aren\'t you?'
$str = "You're awake, aren't you?";
echo addslashes($str);
// remove slashes
// output: 'John D'Souza says "Catch you later".'
$str = "John D\'Souza says \"Catch you later\".";
echo stripslashes($str);
// replace with HTML entities
// output: '&lt;div width=&quot;200&quot;&gt;
// This is a div&lt;/div&gt;'
$html = '<div width="200">This is a div</div>';
echo htmlentities($html);
// replace line breaks with <br/>s
// output: 'This is a bro<br />
// ken line'
$lines = 'This is a bro
ken line';
echo nl2br($lines);
// strip HTML tags from string
// output: 'Please log in again'
$html = '<div width="200">Please <strong>log in again</strong></div>';
echo strip_tags($html);
?>
```

2.1.2.Using Numeric Functions

The PHP language has over 50 built-in functions for working with numbers, ranging from simple formatting functions to functions for arithmetic, logarithmic, and trigonometric manipulations.

Function	What It Does
ceil()	Rounds a number up
floor()	Rounds a number down
abs()	Finds the absolute value of a number
pow()	Raises one number to the power of another
log()	Finds the logarithm of a number
exp()	Finds the exponent of a number
rand()	Generates a random number
bindec()	Converts a number from binary to decimal
decbin()	Converts a number from decimal to binary
decoct()	Converts a number from decimal to octal

dechex()	Converts a number from decimal to hexadecimal
hexdec()	Converts a number from hexadecimal to decimal
octdec()	Converts a number from octal to decimal
number_format()	Formats a number with grouped thousands and decimals
printf()	Formats a number using a custom specification

Common PHP Numeric Functions

Here is some examples :

```
<?php
// round number up
// output: 19
$num = 19.7;
echo floor($num);

// round number down
// output: 20
echo ceil($num);
// return absolute value of number
// output: 19.7
$num = -19.7;
echo abs($num);
// calculate 4 ^ 3
// output: 64
echo pow(4,3);
// calculate natural log of 100
// output: 2.30258509299
echo log(10);
// calculate log of 100, base 10
// output: 2
echo log(100,10);
// calculate exponent of 2.30258509299
// output: 9.99999999996
echo exp(2.30258509299);
?>
```

<u>Generating Random Numbers</u>

Generating random numbers with PHP is pretty simple too: the language's built-in rand() function automatically generates a random integer greater than 0. You can constrain it to a specific number range by providing optional limits as arguments. The following example illustrates:

```
<?php
// generate a random number

echo rand();
// generate a random number between 10 and 99
echo rand(10,99);
?>
```

<u>Converting Between Number Bases</u>

PHP comes with built-in functions for converting between binary, decimal, octal, and hexadecimal bases. Here's an example which demonstrates the bindec(), decbin(), decoct(), dechex(), hexdec(), and octdec() functions in action:

```
<?php
// convert to binary
// output: 1000
echo decbin(8);
// convert to hexadecimal
// output: 8
echo dechex(8);
// convert to octal
// output: 10
echo decoct(8);
// convert from octal
// output: 8
echo octdec(10);
// convert from hexadecimal
// output: 101
echo hexdec(65);
// convert from binary
// output: 8
echo bindec(1000);
?>
```

Formatting Numbers

When it comes to formatting numbers, PHP offers the number_format() function, which accepts four arguments: the number to be formatted, the number of decimal places to display, the character to use instead of a decimal point, and the character to use to separate grouped thousands (usually a comma). Consider the following example, which illustrates:

```
<?php
// format number (with defaults)
// output: 1,106,483
$num = 1106482.5843;
echo number_format($num);
// format number (with custom separators)
// output: 1?106?482*584
echo number_format($num,3,'*','?');
?>
```

For more control over number formatting, PHP offers the printf() and sprintf() functions. These functions, though very useful, can be intimidating to new users, and so the best way to understand them is with an example. Consider the next listing, which shows them in action:

```
<?php
// format as decimal number
// output: 00065
printf("%05d", 65);
// format as floating-point number
// output: 00239.000
printf("%09.3f", 239);
// format as octal number
// output: 10
```

```
printf("%4o", 8);
// format number
// incorporate into string
// output: 'I see 8 apples and 26.00 oranges'
printf("I see %4d apples and %4.2f oranges", 8, 26);
?>
```

Both functions accept two arguments, a series of *format specifiers* and the raw string or number to be formatted. The input is then formatted according to the format specifiers and the output either displayed with printf() or assigned to a variable with sprintf().

Specifier	What It Means
%s	String
%d	Decimal number
%x	Hexadecimal number
%o	Octal number
%f	Floating-point number

Format Specifiers for the printf() and sprintf() Functions

2.2. Working with arrays:

An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables it's easy to define an array of 100 lengths. PHP also supports *arrays,* which let you group and manipulate more than one value at a time.

<u>2.2.1 Storing Data in Arrays</u>

So far, all the variables you've used have held only a single value. Array variables are "special," because they can hold more than one value at a time. This makes them particularly useful for storing related values—for example, a set of fruit names, as in this example:

```
<?php
// define array
$fruits = array('apple', 'banana', 'pineapple', 'grape');
?>
```

The second way to create such an array is to set values individually using index notation. Here's an example, which is equivalent to the preceding one:

```
<?php
// define array
$cars[0] = 'Ferrari';
$cars[1] = 'Porsche';
$cars[2] = 'Jaguar';
$cars[3] = 'Lamborghini';
$cars[4] = 'Mercedes';

?>
```

<u>2.2.2. Assigning Array Values</u>

PHP's rules for naming array variables are the same as those for regular variables: variable names must be preceded with a dollar ($) symbol and must begin with a letter or underscore character, optionally followed by more letters, numbers, or underscore characters. Punctuation characters and spaces are not allowed within array variable names. Once you've decided a name for your array, there are two ways of assigning values to it. The first is the method you saw in the preceding section, where the values are all assigned at once, separated with commas. This method creates a standard, number indexed array. Here's an example:

```
<?php
// define array
$cars = array(
'Ferrari',
'Porsche',
'Jaguar',
'Lamborghini',
'Mercedes'
);
 ?>
```

2.2.3.Modifying Array Values

Modifying an array value is as simple as modifying any other variable value: simply assign a new value to the element using either its index or its key. Consider the following example, which modifies the second element of the $meats array to hold the value 'turkey' instead of 'ham':

```
<?php
// define array
$meats = array(
'fish',
'chicken',
'ham',
'lamb'
 );
 ?>
```

2.2.4.Retrieving Array Size

An important task when using arrays, especially in combination with loops, is finding out how many values the array contains. This is easily accomplished with PHP's count() function, which accepts the array variable as a parameter and returns an integer value indicating how many elements it contains. Here's an example:

```
<?php
// define array
 $data = array('Monday', 'Tuesday', 'Wednesday');
// get array size
echo 'The array has ' . count($data) . ' elements';
 ?>
```

2.2.5.Nesting Arrays

PHP also allows you to combine arrays, by placing one inside another to an unlimited depth. This can come in handy when dealing with structured, hierarchically arranged information. To illustrate, consider the following example:

```
<?php
// define nested array
$phonebook = array(
array(
'name' => 'Raymond Rabbit',
'tel' => '1234567',
'email' => 'ray@bunnyplanet.in',
),
array(
'name' => 'David Duck',
'tel' => '8562904',
'email' => 'dduck@duckpond.corp',
),
array(
'name' => 'Harold Horse',
'tel' => '5942033',
'email' => 'kingharold@farmersmarket.horsestuff.com',
)
);
 ?>
```

2.2.6. Array Types

There are three different kinds of arrays and each array value is accessed using an ID which is called array index.

- **Numeric array** – An array with a numeric index. Values are stored and accessed in linear fashion.
- **Associative array** – An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- **Multidimensional array** – An array containing one or more arrays and values are accessed using multiple indices

Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

Following is the example showing how to create and access numeric arrays.

Here we have used **array()** function to create array. This function is explained in function reference.

```
<html>
  <body>

    <?php
      /* First method to create array. */
      $numbers = array( 1, 2, 3, 4, 5);
       foreach( $numbers as $value ) {
          echo "Value is $value <br />";
      }
      /* Second method to create array. */
      $numbers[0] = "one";
      $numbers[1] = "two";
      $numbers[2] = "three";
      $numbers[3] = "four";
      $numbers[4] = "five";
       foreach( $numbers as $value ) {
         echo "Value is $value <br />";
      }
    ?>
  </body>
</html>
```

This will produce the following result:

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five
```

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE – Don't keep associative array inside double quote while printing otherwise it would not return any value.

```
<html>
  <body>

    <?php
      /* First method to associate create array. */
      $salaries = array("mohammad" => 2000, "qadir" => 1000, "zara" => 500);
      echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
      echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
      echo "Salary of zara is ".  $salaries['zara']. "<br />";
     /* Second method to create array. */
      $salaries['mohammad'] = "high";
      $salaries['qadir'] = "medium";
      $salaries['zara'] = "low";
      echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
```

```
        echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
        echo "Salary of zara is ".  $salaries['zara']. "<br />";
      ?>
   </body>
</html>
```

This will produce the following result :

```
Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low
```

Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple indexes.

In this example we create a two dimensional array to store marks of three students in three subjects. This example is an associative array, you can create numeric array in the same fashion.

```
<html>
   <body>
            <?php
             $marks = array(
                 "mohammad" => array (
                 "physics" => 35,
                 "maths" => 30,
                 "chemistry" => 39
            ),

            "qadir" => array (
               "physics" => 30,
               "maths" => 32,
               "chemistry" => 29
            ),

            "zara" => array (
               "physics" => 31,
               "maths" => 22,
               "chemistry" => 39
            )
         );

         /* Accessing multi-dimensional array values */
         echo "Marks for mohammad in physics : " ;
         echo $marks['mohammad']['physics'] . "<br />";
```

```
        echo "Marks for qadir in maths : ";
        echo $marks['qadir']['maths'] . "<br />";

        echo "Marks for zara in chemistry : " ;
        echo $marks['zara']['chemistry'] . "<br />";
    ?>
    </body>
</html>
```

This will produce the following result :

```
Marks for mohammad in physics : 35
Marks for qadir in maths : 32
Marks for zara in chemistry : 39
```

2.2.7.Using Arrays with Forms

Arrays are particularly potent when used in combination with form elements that support more than one value, such as multiple-selection list boxes or grouped checkboxes. To capture a user's input in an array, simply add square braces to the form element's 'name' to automatically convert it into a PHP array when the form is submitted. The easiest way to illustrate this is with an example. Consider the following form, which holds a multiple-selection list of popular music artists:

```
<form method="post" action="array-form.php">
Select your favourite artists: <br />
<select name="artists[]" multiple="true">
<option value="Britney Spears">Britney Spears</option>
<option value="Aerosmith">Aerosmith</option>
<option value="Black-Eyed Peas">Black-Eyed Peas</option>
<option value="Diana Ross">Diana Ross</option>
<option value="Foo Fighters">Foo Fighters</option>
</select>
<p>
<input type="submit" name="submit" value="Submit" />
</form>
```

Notice the 'name' attribute of the <select> element, which is named artists[]. This tells PHP that, when the form is submitted, all the selected values from the list should be converted into elements of an array. The name of the array will be $_POST['artists'], and it will look something like this:

```
Array
(
[0] => Britney Spears
[1] => Black-Eyed Peas
[2] => Diana Ross
)
```

2.2.8.Working with Array Functions

PHP has numerous built-in array manipulation functions, supporting operations ranging from array search and comparison to sorting and conversion operations.

Function	What It Does
explode()	Splits a string into array elements
implode()	Joins array elements into a string
range()	Generates a number range as an array
min()	Finds the smallest value in an array
max()	Finds the largest value in an array
shuffle()	Randomly rearranges the sequence of elements in an array
array_slice()	Extracts a segment of an array
array_shift()	Removes an element from the beginning of an array
array_unshift()	Adds an element to the beginning of an array
array_pop()	Removes an element from the end of an array
array_push()	Adds an element to the end of an array
array_unique()	Removes duplicate elements from an array
array_reverse()	Reverses the sequence of elements in an array
array_merge()	Combines two or more arrays
array_intersect()	Calculates the common elements between two or more arrays
array_diff()	Calculates the difference between two arrays
in_array()	Checks if a particular value exists in an array
array_key_exists()	Checks if a particular key exists in an array
sort()	Sorts an array
asort()	Sorts an associative array by value
ksort()	Sorts an associative array by key
rsort()	Reverse-sorts an array
krsort()	Reverse-sorts an associative array by value
arsort()	Reverse-sorts an associative array by key

Common PHP Array Functions

The following example shows sorting function

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);
ksort($age);
arsort($age);
?>
```

Another example is

```
<?php
// define array
$arr = array(7, 36, 5, 48, 28, 90, 91, 3, 67, 42);
// get min and max
// output: 'Minimum is 3 and maximum is 91'
```

```
echo 'Minimum is ' . min($arr) . ' and maximum is ' . max($arr);
?>
```

2.3. Working With Date And Time

In addition to allowing you to store multiple values in an array, PHP also comes with a full-featured set of functions for working with dates and times. Although the language does not have a dedicated data type for date and time values, it nevertheless allows programmers a great deal of flexibility when creating, formatting, and manipulating such values.

The PHP Date() Function

The PHP `date()` function formats a timestamp to a more readable date and time. The PHP `date()` function is used to format a date and/or a time.

Syntax

date(*format*,*timestamp*)

The parameter format - Required. Specifies the format of the timestamp,

Timestamp - Optional. Specifies a timestamp. Default is the current date and time

What is a TimeStamp?

A timestamp is a numeric value in seconds between the current time and value as at 1st January, 1970 00:00:00 Greenwich Mean Time (GMT).

Another way to define time stamp - A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

- The value returned by the time function depends on the default time zone.
- The default time zone is set in the php.ini file.
- It can also be set programmatically using date_default_timezone_set function.

The code below displays the current time stamp

```
<?php
        echo time();
?>
```

Get a Date

The required *format* parameter of the date() function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

- d - Represents the day of the month (01 to 31)
- m - Represents a month (01 to 12)
- Y - Represents a year (in four digits)
- l (lowercase 'L') - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

The example below formats today's date in three different ways:

Example

```
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
?>
```

Example : Use the `date()` function to automatically update the copyright year on your website:

```
&copy; 2010-<?php echo date("Y");?>
```

Get a Time

Here are some characters that are commonly used for times:

- H - 24-hour format of an hour (00 to 23)
- h - 12-hour format of an hour with leading zeros (01 to 12)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds with leading zeros (00 to 59)
- a - Lowercase Ante meridiem and Post meridiem (am or pm)

The example below outputs the current time in the specified format:

```
<?php
echo "The time is " . date("h:i:sa");
?>
```

Get Your Time Zone

If the time you got back from the code is not correct, it's probably because your server is in another country or set up for a different timezone. So, if you need the time to be correct according to a specific location, you can set the timezone you want to use.

The example below sets the timezone to "America/New_York", then outputs the current time in the specified format:

```
<?php
date_default_timezone_set("America/New_York");
echo "The time is " . date("h:i:sa");
?>
```

The PHP `mktime()` Function

The `mktime()` function is used to create the timestamp based on a specific date and time. If no date and time is provided, the timestamp for the current date and time is returned.

The syntax of the `mktime()` function can be given with:

```
mktime(hour, minute, second, month, day, year)
```

The following example displays the timestamp corresponding to 3:20:12 pm on May 10, 2014:

```
<?php
// Create the timestamp for a particular date
echo mktime(15, 20, 12, 5, 10, 2014);
?>
```

The above example produce the following output.

```
1399735212
```

Create a Date From a String With strtotime()

The PHP strtotime() function is used to convert a human readable date string into a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).

Syntax

strtotime(*time, now*)

The example below creates a date and time from the strtotime() function:

```
<?php
$d=strtotime("10:30pm April 15 2014");
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

PHP is quite clever about converting a string to a date, so you can put in various values:

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

Converting a Time Stamp with getdate()

The function **getdate()** optionally accepts a time stamp and returns an associative array containing information about the date. If you omit the time stamp, it works with the current time stamp as returned by time().

Following table lists the elements contained in the array returned by getdate().

Sr.No	Key & Description	Example
1	**seconds** Seconds past the minutes (0-59)	20
2	**minutes** Minutes past the hour (0 - 59)	29
3	**hours** Hours of the day (0 - 23)	22
4	**mday** Day of the month (1 - 31)	11
5	**wday** Day of the week (0 - 6)	4

6	**mon** Month of the year (1 - 12)	7
7	**year** Year (4 digits)	1997
8	**yday** Day of year (0 - 365)	19
9	**weekday** Day of the week	Thursday
10	**month** Month of the year	January
11	**0** Timestamp	948370048

Example

```
<?php
   $date_array = getdate();

   foreach ( $date_array as $key => $val ){
      print "$key = $val<br />";
   }

   $formated_date  = "Today's date: ";
   $formated_date .= $date_array['mday'] . "/";
   $formated_date .= $date_array['mon'] . "/";
   $formated_date .= $date_array['year'];

   print $formated_date;
?>
```

This will produce following result –

```
seconds = 10
minutes = 29
hours = 9
mday = 5
wday = 1
mon = 12
year = 2016
yday = 339
weekday = Monday
month = December
0 = 1480930150
Today's date: 5/12/2016
```

Converting a Time Stamp with date()

The **date()** function returns a formatted string representing a date. You can exercise an enormous amount of control over the format that date() returns with a string argument that you must pass to it.

Following table lists the codes that a format string can contain –

S.No.	Format & Description	Example
1	**A -**'am' or 'pm' lowercase	pm
2	**A -** 'AM' or 'PM' uppercase	PM
3	**D -** Day of month, a number with leading zeroes	20
4	**D -** Day of week (three letters)	Thu
5	**F -** Month name	January
6	**H -** Hour (12-hour format - leading zeroes)	12
7	**H -** Hour (24-hour format - leading zeroes)	22
8	**G -** Hour (12-hour format - no leading zeroes)	12
9	**G -**Hour (24-hour format - no leading zeroes)	22
10	**I -**Minutes (0 - 59)	23
11	**J -** Day of the month (no leading zeroes	20
12	**l (Lower 'L') -**Day of the week	Thursday
13	**L -**Leap year ('1' for yes, '0' for no)	1
14	**M -**Month of year (number - leading zeroes)	1

15	**M** -Month of year (three letters)	Jan
16	**R** -The RFC 2822 formatted date	Thu, 21 Dec 2000 16:01:07 +0200
17	**N** -Month of year (number - no leading zeroes)	2
18	**S** -Seconds of hour	20
19	**U** -Time stamp	948372444
20	**Y** -Year (two digits)	06
21	**Y** -Year (four digits)	2006
22	**Z** -Day of year (0 - 365)	206
23	**Z** -Offset in seconds from GMT	+5

For example

```
<?php
  print date("m/d/y G.i:s<br>", time());
  echo "<br>";
  print "Today is ";
  print date("j of F Y, \a\\t g.i a", time());
?>
```

This will produce following result –

```
12/05/16 9:29:47
Today is 5 2016f December 2016 at 9:29 am
```

<u>Useful Date and Time Functions</u>

PHP also supports many other date/time manipulation functions, which allow you to check if a date is valid or convert between time zones. Some of these functions.

Function	What It Does
checkdate()	Checks a date for validity
strtotime()	Creates timestamps from English-language descriptions
gmdate()	Expresses a timestamp in GMT

Common PHP Date Functions

Checking Date Validity

A useful built-in function is the checkdate() function, which accepts a month, day, and year combination and returns a true/false value indicating whether the corresponding date is valid. Consider the following example, which illustrates it by testing the date 30 February 2008:

```
<?php
// output: 'Date is invalid'
if (checkdate(2,30,2008)) {
echo 'Date is valid';
} else {
echo 'Date is invalid';
}
?>
```

Converting Strings to Timestamps

Another very useful function is PHP’s strtotime() function, which accepts a string value containing an embedded date or time and converts this into a UNIX timestamp. Here’s an example:

```
<?php
// define string containing date value
// convert it to UNIX timestamp
// re-format it using date()
// output: '07 Jul 08'
$str = 'July 7 2008';
echo date('d M y', strtotime($str));
?>
```

Calculating GMT Time

PHP’s gmdate() function works exactly like its date() function, except that it returns GMT representations of formatted date strings, instead of local time representations. To see this in action, consider the following examples, which return GMT equivalents for two local times:

```
<?php
// returns GMT time relative to 'now'
echo gmdate('H:i:s d-M-Y', mktime());
// returns GMT time relative to '00:01 1-Dec-2007'
// output: '18:31:00 30-Nov-2007'
echo gmdate('H:i:s d-M-Y', mktime(0,1,0,12,1,2007));
?>
```

2.4. Creating User Defined Functions

PHP - Functions

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value. You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.
There are two parts:

- Creating a PHP Function

- Calling a PHP Function

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

2.4.1 Creating PHP Function

Its very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below.

```
<html>

  <head>
    <title>Writing PHP Function</title>
  </head>
    <body>
        <?php
      /* Defining a PHP Function */
      function writeMessage() {
        echo "You are really a nice person, Have a nice time!";
      }
            /* Calling a PHP Function */
      writeMessage();
    ?>
      </body>
</html>
```

This will display following result –

You are really a nice person, Have a nice time!

2.4.2 PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

```
<html>
    <head>
    <title>Writing PHP Function with Parameters</title>
  </head>
    <body>
      <?php
         function addFunction($num1, $num2) {
                $sum = $num1 + $num2;
                        echo "Sum of the two numbers is : $sum";
         }
```

```
        addFunction(10, 20);
    ?>
  </body>
</html>
```

This will display following result –

```
Sum of the two numbers is : 30
```

2.4.3 Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value. Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Following example depicts both the cases.

```
<html>
    <head>
    <title>Passing Argument by Reference</title>
  </head>
    <body>
        <?php
         function addFive($num) {
                  $num += 5;
         }
               function addSix(&$num) {
                  $num += 6;
        }
          $orignum = 10;
         addFive( $orignum );
               echo "Original Value is $orignum<br />";
         addSix( $orignum );
        echo "Original Value is $orignum<br />";
        ?>
      </body>
</html>
```

This will display following result –

```
Original Value is 10
Original Value is 16
```

2.4.4 PHP Functions returning value

A function can return a value using the **return** statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.You can return more than one value from a function using **return array(1,2,3,4)**.

Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

```
<html>
```

```
    <head>
    <title>Writing PHP Function which returns value</title>
  </head>
    <body>
         <?php
         function addFunction($num1, $num2) {
                 $sum = $num1 + $num2;
                           return $sum;
                 }
        $return_value = addFunction(10, 20);
           echo "Returned value from the function : $return_value";
    ?>
     </body>
</html>
```

This will display following result –

```
Returned value from the function : 30
```

2.4.5 Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it. Following function prints NULL in case use does not pass any value to this function.

```
<html>
    <head>
    <title>Writing PHP Function which returns value</title>
  </head>
    <body>
        <?php
           function printMe($param = “*****”) {
                  print $param;
            }
            printMe("This is test");
         printMe();
        ?>
    </body>
</html>
```

This will produce following result –

```
This is test
*****
```

2.4.6 Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. Following example depicts this behaviour.

```
<html>
   <head>
    <title>Dynamic Function Calls</title>
  </head>
    <body>
```

```
            <?php
             function sayHello() {
                            echo "Hello<br />";
              }
             $function_holder = "sayHello";
             $function_holder();
            ?>
      </body>
</html>
```

This will display following result :

```
Hello
```

2.5.Working with forms:

2.5.1 What is the Form?

A Document that containing black fields, that the user can fill the data or user can select the data. Casually the data will store in the data base looks at strategies for acquiring and working with user input.

2.5.1.creating a simple Input form

2.5.2 Creating a Simple Input Form

Let's keep the HTML separate from the PHP code. The following listing builds a simple HTML form.

```
<!DOCTYPE html>
<html>
<head>
<title>A simple HTML form</title>
</head>
<body>
<form method="post" action="send_simpleform.php">
<p><label for="user">Name:</label><br/>
<input type="text" id="user" name="user"></p>
<p><label for="message">Message:</label><br/>
<textarea id="message" name="message" rows="5" cols="40"></textarea></p>
<button type="submit" name="submit" value="send">Send Message</button>
</form>
</body>
</html>
```

Put these lines into a text file called simpleform.html and place that file in your web server document root.

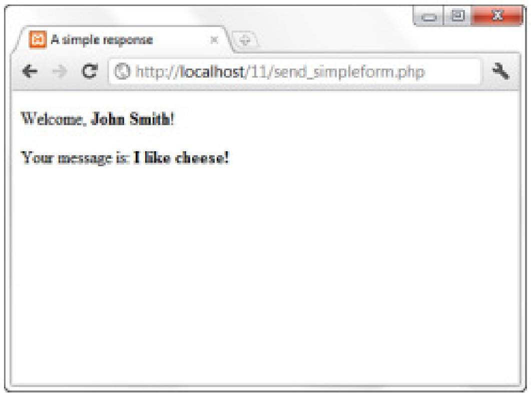
Reading Input from a Form

Creates the code that receives user input and displays it within the context of an HTML page.

```
<!DOCTYPE html>
```

```
<html>
<head>
<title>A simple response</title>
</head>
<body>
<p>Welcome, <strong><?php echo $_POST['user']; ?></strong>!</p>
<p>Your message is:
<strong><?php echo $_POST['message']; ?></strong></p>
</body>
</html>
```

The output is



Welcome, **John Smith!**

Your message is: **I like cheese!**

2.5.3. Accessing Form Input with User-Defined Arrays

Accesses two variables are : $_POST['user'] and $_GET

GET vs. POST

Both GET and POST create an array (e.g. array(key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$_GET is an array of variables passed to the current script via the URL parameters.

$_POST is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases. GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send. Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server. However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

2.5.4. Combining HTML and PHP code on single page

In some circumstances, we want to include the form-parsing PHP code on the same page as a hard-coded HTML form. Such a combination can prove useful if you need to present the same form to the user more than once. You would have more flexibility if you were to write the entire page dynamically,

For the following examples, imagine that you’re creating a site that teaches basic math to preschool children and have been asked to create a script that takes a number from form input and tells the user whether it’s larger or smaller than a predefined integer.

Combining HTML and PHP Code on a Single

```
1: <!DOCTYPE html>
2: <html>
3: <head>
4: <title>An HTML form that calls itself</title>
5: </head>
6: <body>
7: <form action=”<?php echo $_SERVER[‘PHP_SELF’]; ?>” method=”POST”>
8: <p><label for=”guess”>Type your guess here:</label> <br/>
9: <input type=”text” id=”guess” name=”guess” /></p>
10: <button type=”submit” name=”submit” value=”submit”>Submit</button>
11: </form>
12: </body>
13: </html>
```

The action of this script is $_SERVER[‘PHP_SELF’], as shown in line 7. This global variable represents the name of the current script. The following script does not produce any output, but if you upload the script to your web server, access the page, and view the source of the page, you will notice that the form action now contains the name of the script itself.

The following code will build up the PHP element of the page.

```
1: <?php
2: $num_to_guess = 42;
3: if (!isset($_POST[‘guess’])) {
4: $message = “Welcome to the guessing machine!”;
5: } elseif (!is_numeric($_POST[‘guess’])) { // is not numeric
6: $message = “I don’t understand that response.”;
7: } elseif ($_POST[‘guess’] == $num_to_guess) { // matches!
8: $message = “Well done!”;
9: } elseif ($_POST[‘guess’] > $num_to_guess) {
10: $message = $_POST[‘guess’].” is too big! Try a smaller number.”;
11: } elseif ($_POST[‘guess’] < $num_to_guess) {
12: $message = $_POST[‘guess’].” is too small! Try a larger number.”;
13: } else { // some other condition
14: $message = “I am terribly confused.”;
15: }
16: ?>
```

First, you must define the number that the user guesses, and this is done in line 2 when 42 is assigned to the $num_to_guess variable. Next, you must determine whether the form has been submitted. You can test for submission by looking for the existence of the variable $_POST[‘guess’], which is available only if the form script has been submitted (with or without a value in the field). If a value for $_POST[‘guess’] isn’t present, you can safely assume that the user arrived at the page without submitting a form. If the value *is* present, you can test the value it contains. The test for the presence of the $_POST[‘guess’] variable takes place on line 3. Lines 3 through 15 represent an if...elseif...else control structure. Only one of these conditions will be true at any given time, depending on what (if anything) was submitted from the form. Depending on the condition, a different value is assigned to the $message variable. That variable is then printed to the screen in line 23 in listing below is part of the HTML portion of the script.

Note : A PHP Number-Guessing Script (Continued)

```
17: <!DOCTYPE html>
18: <html>
19: <head>
20: <title>A PHP number guessing script</title>
21: </head>
22: <body>
23: <h1><?php echo $message; ?></h1>
24: <form action=”<?php echo $_SERVER[‘PHP_SELF’]; ?>” method=”POST”>
25: <p><label for=”guess”>Type your guess here:</label><br/>
26: <input type=”text” is=”guess” name=”guess” /></p>
27: <button type=”submit” name=”submit” value=”submit”>Submit</button>
28: </form>
29: </body>
30: </html>
```

2.5.5.Using Hidden Fields To Save State

If the script has no way of knowing how many guesses a user has made, but you can use a hidden field to keep track of this value. A hidden field behaves the same as a text field, except that the user cannot see it unless he views the HTML source of the document that contains it. Take the original numguess.php script and save a copy as numguess2.php. In the new version, add a line after the initial assignment of the $num_to_guess variable:

```
$num_tries = (isset($_POST[‘num_tries’])) ? $num_tries + 1 : 1;
```

This line initializes a variable called $num_tries and assigns a value to it. If the form has not yet been submitted (if $_POST[‘num_tries’] is empty), the value of the $num_tries variable is 1 because you are on your first attempt at guessing the number. If the form has already been sent, the new value is the value of

$_POST[‘num_tries’] plus 1.

The next change comes after the HTML level H1 heading:

```
<p><strong>Guess number:</strong> <?php echo $num_tries; ?></p>
```

This new line simply prints the current value of $num_tries to the screen. Finally, before the HTML code for the form submission button, add the hidden field. This field saves the incremented value of $num_tries:

```
<input type=”hidden” name=”num_tries” value=”<?php echo $num_tries; ?>”/>
```

The following script listing Saving State with a Hidden Field

```
<?php
$num_to_guess = 42;
$num_tries = (isset($_POST[‘num_tries’])) ? $num_tries + 1 : 1;
if (!isset($_POST[‘guess’])) {
$message = “Welcome to the guessing machine!”;
} elseif (!is_numeric($_POST[‘guess’])) { // is not numeric
$message = “I don’t understand that response.”;
} elseif ($_POST[‘guess’] == $num_to_guess) { // matches!
$message = “Well done!”;
} elseif ($_POST[‘guess’] > $num_to_guess) {
$message = $_POST[‘guess’].” is too big! Try a smaller number.”;
} elseif ($_POST[‘guess’] < $num_to_guess) {
$message = $_POST[‘guess’].” is too small! Try a larger number.”;
} else { // some other condition
$message = “I am terribly confused.”;
}
?>
<!DOCTYPE html>
<html>
<head>
<title>A PHP number guessing script</title>
</head>
<body>
<h1><?php echo $message; ?></h1>
<p><strong>Guess number:</strong> <?php echo $num_tries; ?></p>
<form action=”<?php echo $_SERVER[‘PHP_SELF’]; ?>” method=”POST”>
<p><label for=”guess”>Type your guess here:</label><br/>
<input type=”text” id=”guess” name=”guess” /></p>
```

```
<input type=”hidden” name=”num_tries” value=”<?php echo $num_tries; ?>”/>
<button type=”submit” name=”submit” value=”submit”>Submit</button>
</form>
</body>
</html>
```

Save the numguess2.php file and place it in your web server document root. Access the form a few times with your web browser and try to guess the number. The counter should increment by 1 each time you access the form.

2.5.6. Redirecting The User

Our simple script still has one major drawback: The form is reloaded whether or not the user guesses correctly. The fact that the HTML is hard-coded makes it difficult to avoid writing the entire page. You can, however, redirect the user to a congratulations page, thereby sidestepping the issue altogether. When a server script communicates with a client, it must first send some headers that provide information about the document to follow. PHP usually handles this for you automatically, but you can choose to send your own header lines with PHP’s header() function.

To call the header() function, you must be absolutely sure that no output has been sent to the browser. The first time content is sent to the browser, PHP sends out headers of its own, and it’s too late for you to send any more. Any output from your document, even a line break or a space outside your script tags, causes headers to be sent. If you intend to use the header() function in a script, you must make certain that nothing precedes the PHP code that contains the function call.

The following listing shows typical headers sent to the browser by PHP, beginning with line 3, in response to the request in line 1.

```
// Typical Server Headers Sent from a PHP Script
1: HTTP/1.1 200 OK
2: Date: Sun, 29 Jan 2012 15:50:28 PST
3: Server: Apache/2.2.21 (Win32) PHP/5.4.0
4: X-Powered-By: PHP/5.4.0
5: Connection: close
6: Content-Type: text/html
```

By sending a Location header rather than PHP’s default header, you can cause the browser to be redirected to a new page, such as the following:

```
header(“Location: http://www.samspublishing.com”);
```

Assuming that you’ve created a suitably upbeat page called congrats.html; we can amend the number-guessing script to redirect the user if she guesses correctly.

```
// Using header() to Redirect User
<?php
$num_to_guess = 42;
$num_tries = (isset($_POST[‘num_tries’])) ? $num_tries + 1 : 1;
if (!isset($_POST[‘guess’])) {
$message = “Welcome to the guessing machine!”;
} elseif (!is_numeric($_POST[‘guess’])) { // is not numeric
$message = “I don’t understand that response.”;
} elseif ($_POST[‘guess’] == $num_to_guess) { // matches!
header(“Location: congrats.html”);
exit;
} elseif ($_POST[‘guess’] > $num_to_guess) {
$message = $_POST[‘guess’].” is too big! Try a smaller number.”;
} elseif ($_POST[‘guess’] < $num_to_guess) {
$message = $_POST[‘guess’].” is too small! Try a larger number.”;
} else { // some other condition
$message = “I am terribly confused.”;
}
?>

<!DOCTYPE html>
<html>
<head>
<title>A PHP number guessing script</title>
</head>
<body>
<h1><?php echo $message; ?></h1>
<p><strong>Guess number:</strong> <?php echo $num_tries; ?></p>
<form action=”<?php echo $_SERVER[‘PHP_SELF’]; ?>” method=”POST”>
<p><label for=”guess”>Type your guess here:</label><br/>
<input type=”text” id=”guess” name=”guess” /></p>
<input type=”hidden” name=”num_tries” value=”<?php echo $num_tries; ?>”/>
<button type=”submit” name=”submit” value=”submit”>Submit</button>
</form>
</body>
</html>
```

2.5.7.Working With File Uploads

However, web browsers support file uploads, and so, information about the uploaded file becomes available to you in the $_FILES super global, which is indexed by the name of the upload field (or fields) in the form. The corresponding value for each of these keys is an associative array. These fields are described in Table 1, using file upload as the name of the form field used for the upload.

TABLE 1 File Upload Global Variables

Element	Contains	Example
$_FILES[‘fileupload’][‘name’]	Original name of uploaded	file test.gif
$_FILES[‘fileupload’]	Path to temporary file	/tmp/phprDfZvN [‘tmp_name’]
$_FILES[‘fileupload’][‘size’]	Size (in bytes) of uploaded file	6835
$_FILES[‘fileupload’][‘type’]	MIME type of uploaded file (where given by client)	image/gif

Creating the File Upload Form

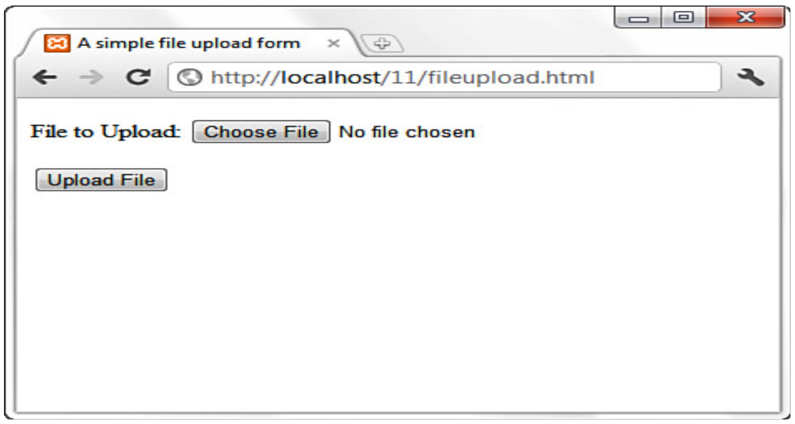
First, you must create the HTML form to handle the upload. HTML forms that include file upload fields must include an ENCTYPE argument:

enctype=”multipart/form-data”

PHP also works with an optional hidden field that can be inserted before the file upload field. This field must be called MAX_FILE_SIZE and should have a value representing the maximum size in bytes of the file that you’re willing to accept. The MAX_FILE_SIZE field is obeyed at the browser’s discretion, so you should rely on the php.ini setting, upload_max_filesize, to cap unreasonably large uploads. After the MAX_FILE_SIZE field has been entered, you are ready to add the upload field itself. This is simply an INPUT element with a TYPE argument of “file”. You can give it any name you want. The following example all this together into an HTML upload form.

```
//A Simple File Upload Form
1: <!DOCTYPE html>
2: <html>
3: <head>
4: <title>A simple file upload form</title>
5: </head>
6: <body>
7: <form action=”do_upload.php” enctype=”multipart/form-data” method=”POST”>
8: <input type=”hidden” name=”MAX_FILE_SIZE” value=”1048576” />
9: <p><label for=”fileupload”>File to Upload:</label>
10: <input type=”file” id=”fileupload” name=”fileupload” /></p>
11: <button type=”submit” name=”submit” value=”send”>Upload File</button>
12: </form>
13: </body>
14: </html>
```

File uploads are limited to 1MB (or 1048576 bytes) on line 8, and the name of the file upload field is file upload, as shown on line 8. Save this listing in a text file called fileupload.html and place that file in your web server document root. Use your web browser to access this form and you should see something like


A simple file upload form
http://localhost/11/fileupload.html
File to Upload:
Choose File
No file chosen
Upload File

This form calls the do_upload.php script, which you create next.

Creating the File Upload Script

If you remember the information regarding the $_FILES superglobal, you have all the information you need to write a simple file upload script. This script is the backend for the form created.

Program : A File Upload Script

```
1: <?php
2: $file_dir = “/path/to/upload/directory”;
3:
4: foreach($_FILES as $file_name => $file_array) {
5: echo “path: “.$file_array[‘tmp_name’].”<br/>\n”;
6: echo “name: “.$file_array[‘name’].”<br/>\n”;
7: echo “type: “.$file_array[‘type’].”<br/>\n”;
8: echo “size: “.$file_array[‘size’].”<br/>\n”;
9:
10: if (is_uploaded_file($file_array[‘tmp_name’])) {
11: move_uploaded_file($file_array[‘tmp_name’],
12: “$file_dir/”.$file_array[‘name’])
13: or die (“Couldn’t move file”);
14: echo “File was moved!”;
15: } else {
16: echo “No file found.”;
17: }
18: }
19: ?>
```

First create the $file_dir variable on line 2 to store path information. This path should be one that exists on your system, and the web server user (for example, httpd, www, nobody) must have write permissions for it. The path used in line 2 is a Linux/UNIX path. Windows users would use a path including the drive letter, such as the following:

```
$file_dir = “C:\Users\You\Desktop”;
```

Line 4 begins a foreach statement that loops through every element in the $_FILES array. A foreach loop is used rather than an if statement to make the script capable of scaling to deal with multiple uploads on the same page. The foreach loop beginning on line 4 stores the upload file’s name in the $file_name variable and the file information in the $file_array variable. You can then output the information you have about the upload.

Before moving the uploaded file from its temporary position to the location specified in line 2, first check that the file exists (has been uploaded). The code does so on line 10, using the is_uploaded_file() function. This function accepts a path to an uploaded file and returns true only if the file in question is a valid upload file. This function therefore enhances the security of your scripts.

Assuming that all is well, the file is copied from its temporary home to a new directory on lines 11 to 13. Another function, move_uploaded_file (), is used for this purpose. This function copies a file from one place to another, first performing the same security checks as those performed by is_uploaded_file(). The move_uploaded_file() function requires a path to the source file and a path to the destination. It returns true if the move is successful and false if the file isn’t a valid upload file or if the file couldn’t be found.