Lecture-21-22

Functions in PHP: Defining a Function , Calling A function, Variable Scope(Global variable& Static Variables)

Function

- A function is a piece of code in a larger program. The function performs a specific task. The advantages of using functions are:
- Reducing duplication of code
- Decomposing complex problems into simpler pieces
- Improving clarity of the code
- Reuse of code
- Information hiding

There are four basic types of functions.

- user-defined functions
- Function arguments
- Returning values
- Variable functions
- Internal (built-in) functions
- Anonymous functions

PHP User Defined Functions

- Besides the built-in PHP functions, we can create our own functions.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.

- Any valid PHP code may appear inside a function, even other functions and class definitions.
- Function names follow the same rules as other labels in PHP. A valid function name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

Defining functions

- A function is created with the function keyword.
- A function may be defined using syntax such as the following:

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

Conditional functions

- Functions need not be defined before they are referenced, *except* when a function is conditionally defined.
- When a function is defined in a conditional manner. Its definition must be processed *prior* to being called.

Example: Conditional functions

```
<?php

$makefunction = true;

/* We can't call fun() from here
   since it doesn't exist yet,
   but we can call bar() */
bar();
//fun();
if ($makefunction) {
  function fun()
  {
    echo "I don't exist until program execution reaches me <br>";
  }
```

```
/* Now we can safely call fun()
   since $makefunction evaluated to true */

if ($makefunction)
fun();

function bar()
{
  echo "I exist immediately upon program start.<br>";
}

?>
```

Example: Functions within functions

```
<?php
function abc()
{
echo "hello <br>";
  function xyz()
  {
    echo "I don't exist until abc() is called.\n";
  }
}

/* We can't call xyz() yet
   since it doesn't exist. */

abc();

/* Now we can call xyz(),
   abc()'s processing has
   made it accessible. */

xyz();

?>
```

Note:

- All functions and classes in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa.
- PHP does not support function overloading, nor is it possible to undefine or redefine previously-declared functions.

- Function names are case-insensitive, though it is usually good form to call functions as they appear in their declaration.
- Both variable number of arguments and default arguments are supported in functions.
- It is possible to call recursive functions in PHP. However avoid recursive function/method calls with over 100-200 recursion levels as it can smash the stack and cause a termination of the current script.

Example :Recursive functions

```
<?php
function recursion($a)
{
    if ($a < 20) {
        echo "$a\n";
        recursion($a + 1);
    }
}
recursion(1);
?>
```

Function arguments

- Information may be passed to functions via the argument list, which is a comma-delimited list of expressions.
- The arguments are evaluated from left to right.
- PHP supports passing arguments by value (the default), passing by reference, and default argument values.
- Variable-length argument lists are also supported.

Example: function with one argument

```
<?php
function familyName($fname)
{
echo "$fname Sharma.<br>";
}

familyName(“Aman");
familyName(“Ajay");
familyName(“Ankit");
familyName(“Anil");
familyName(“Amar");
?>
```

Example :function with two arguments

```
<?php
function familyName($fname,$year)
{
echo "$fname Sharma. Born in $year <br>";
}

familyName(“Aman","1975");
familyName(“Amar","1978");
familyName(“Ajay","1983");
?>
```

PHP Default Argument Value

```
<?php
   function  setHeight($minheight=50)
   {
   echo "The height is : $minheight <br>";
   }
   setHeight(350);
   setHeight(); // will use the default value of 50
   setHeight(135);
   setHeight(80);
   ?>
```

Example : Passing arrays to functions

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

function arguments are passed by value or Call by Value

```
<?php
$a=10;
$b=20;
echo "before swaping value is : $a  $b <br>";
function swap($i,$j)
{
$t=$i;
$i=$j;
$j=$t;
echo "after swaping $i $j";
}
swap($a, $b)
?>
```

Making arguments be passed by reference

- By default, function arguments are passed by value (so that if the value of the argument within the function is changed, it does not get changed outside of the function).
- To allow a function to modify its arguments, they must be passed by reference.
- To have an argument to a function always passed by reference, prepend an ampersand (&) to the argument name in the function definition:

Call By Reference

```
<?php
$a=10;
$b=20;
echo "before swaping value is : $a  $b <br>";
function swap(&$i ,& $j)
{
$t=$i;
$i=$j;
$j=$t;
echo "after swaping $i $j";
}
swap($a, $b)
?>
```

Example : Passing function parameters by reference

```
<?php
   function add_some_extra(&$string)
   {
      $string .= 'and something extra.';
   }
   $str = 'This is a string, ';
   add_some_extra($str);
   echo $str;    // outputs 'This is a string, and something e
   xtra.'
   ?>
```

Example: returning value from call by references

```
<?php
$a1="William";
$a2="henry";
$a3="gates";
echo $a1." ".$a2." ".$a3 ."<br/>" ;
fix_name($a1,$a2,$a3);
echo $a1." ".$a2." ".$a3."<br>";

function fix_name(&$n1,&$n2,&$n3)
{
$n1=strtoupper($n1);
$n2=strtoupper($n2);
$n3=strtoupper($n3);

}
?>
```

PHP also allows the use of arrays and the special type NULL as default values, for example:

```
<?php
function makecoffee($types = array("cappuccino"), $coffeeMaker =
   NULL)
{
   $device = is_null($coffeeMaker) ? "hands" : $coffeeMaker;
   echo"<br>";
  return "Making a cup of ".join(", ", $types)." with $device.<br>";
}
   echo makecoffee();

echo makecoffee(array("cappuccino", "lavazza"), "teapot");
?>
```

Note:

- The default value must be a constant expression, not (for example) a variable, a class member or a function call.
- when using default arguments, any defaults should be on the right side of any non-default arguments; otherwise, things will not work as expected.

Example: Incorrect usage of default function arguments

```
<?php
   function makeyogurt($type = "acidophilus", $flavour)
   {
      return "Making a bowl of $type $flavour.\n";
   }

   echo makeyogurt("raspberry");   // won't work as expected
   ?>
```

- **output:**

  Warning: Missing argument 2 for makeyogurt(), called in C:\wamp\www\suman\fun_Incorrect_default_null.php on line 8 and defined in **C:\wamp\www\suman\fun_Incorrect_default_null.php** on line **3**

  Notice: Undefined variable: flavour in **C:\wamp\www\suman\fun_Incorrect_default_null.php** on line **5**
  Making a bowl of berry

Example Correct usage of default function arguments

```
<?php
function makeyogurt($flavour, $type = "acid")
{
    return "Making a bowl of $type $flavour.\n";
}

echo makeyogurt("berry");   // works as expected
?>
```

output:

- Making a bowl of acid berry.

Note: **In PHP 5, arguments that are passed by reference may have a default value.**

- **Variable-length argument lists**
- PHP has support for variable-length argument lists in user-defined functions. This is really quite easy, using the func_num_args(), func_get_arg(), and func_get_args() functions.
- No special syntax is required, and argument lists may still be explicitly provided with function definitions and will behave as normal.

3. Returning values

- Values are returned by using the optional return statement. Any type may be returned, including arrays and objects. This causes the function to end its execution immediately and pass control back to the line from which it was called.
- **Note**: If the return is omitted the value **NULL** will be returned.

Example :Use of return

```
<?php
   function square($num)
   {
      return $num * $num;
   }
   echo square(4);   // outputs '16'.
   ?>
```

A function can not return multiple values, but similar results can be obtained by returning an array.

- **Example: Returning an array to get multiple values**

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

- To return a reference from a function, use the reference operator & in both the function declaration and when assigning the returned value to a variable:
- **Example :Returning a reference from a function**

```
<?php
function &returns_reference()
{
    return $someref;
}
$newref =& returns_reference();
?>
```

4: Variable functions

- PHP supports the concept of variable functions.
- This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it.
- Among other things, this can be used to implement callbacks, function tables, and so forth.
- Variable functions won't work with language constructs such as echo, print, unset(), isset(), empty(), include, require and the like. Utilize wrapper functions to make use of any of these constructs as variable functions.

Example :Variable function example

```
<?php
function abc() {
    echo "In abc()<br />\n";
}

function xyz($arg = '')
{
    echo "In xyz(); argument was '$arg'.<br />\n";
}

// This is a wrapper function around echo
function echoit($string)
{
    echo $string;
}

$func = 'abc';
$func();

$func = 'xyz';
$func('test');
$func = 'echoit';
$func('test');
?>
```

Object methods can also be called with the variable functions syntax.

Example :Variable method example

```
<?php
class Foo
{
    function Variable()
    {
        $name = 'Bar';
        $this->$name(); // This calls the Bar() method
    }

    function Bar()
    {
        echo "This is Bar";
    }
}

$foo = new Foo();
$funcname = "Variable";
$foo->$funcname();  // This calls $foo->Variable()

?>
```

When calling static methods, the function call is stronger than the static property operator:

- **Example: Variable method example with static properties**

```
<?php
   class Foo
   {
      static $variable = 'static property';
      static function Variable()
      {
         echo 'Method Variable called';
      }
   }

   echo Foo::$variable; // This prints 'static property'. It does need a $vari
   able in this scope.
   $variable = "Variable";
   Foo::$variable();  // This calls $foo-
   >Variable() reading $variable in this scope.

   ?>
```

Internal (built-in) functions

- PHP comes standard with many functions and constructs. There are also functions that require specific PHP extensions compiled in, otherwise fatal "undefined function" errors will appear.
- For example:

1. to use image functions such as imagecreatetruecolor(), PHP must be compiled to use mysql_connect(), PHP must be compiled with MySQL support.
2. There are many core functions that are included in every version of PHP, such as the string and variable functions.
3. A call to phpinfo() or get_loaded_extensions() will show which extensions are loaded into PHP. Also note that many extensions are enabled by default and that the PHP manual is split up by extension.

- For example, str_replace() will return the modified string while usort() works on the actual passed in variable itself. Each manual page also has specific information for each function like information on function parameters, behavior changes, return values for both success and failure, and availability information.
- **Note**: If the parameters given to a function are not what it expects, such as passing an array where a string is expected, the return value of the function is undefined. In this case it will likely return **NULL** but this is just a convention, and cannot be relied upon.

Anonymous functions

- Anonymous functions, also known as *closures*, allow the creation of functions which have no specified name. They are most useful as the value of **callback** parameters, but they have many other uses.
- **Example :Anonymous function example**
- <?php
  echo preg_replace_callback('~-([a-
  z])~', function ($match) {
      return strtoupper($match[1]);
  }, 'hello-world');   // outputs helloWorld
  ?>

Lecture-25
Include , Require, Date ,time and printf function

Include and require statement

- In PHP programming we start building a library of functions that we think we will need again.
- We can use inbuilt library function
- There is no need to copy and paste these functions into our code .We can save them and use command to pull.

There are two types of command

1. include statement
2. require statement

Both include and require statement are identical , except upon failure

- require will produce a fatal error (E_COMPILE_ERROR) and stop the script
- include will only produce a warning (E_WARNING) and the script will continue

Include() function

- Using include we can fetch a particular file and load all its contents
- Syntax:

  Include "filename”;

Example:

```
<html>
<body>
<?php include 'header.php'; ?>
<h1>Welcome to my home page!</h1>
<p>Some text.</p>
</body>
</html>
```

Example: we have an include file with some variables defined ("vars.php"):

```
<?php
$color='red';
$car='BMW';
?>
```

Then the variables can be used in the calling file

```
<html>
   <body>
<h1>Welcome to my home page.</h1>
   <?php include 'vars.php';
   echo "I have a $color $car";
   ?>
</body>
   </html>
```

require() Function

- The require() function takes all the text in a specified file and copies it into the file that uses the include function.
- If there is any problem in loading a file then the **require()** function generates a fatal error and halt the execution of the script.
- there is no difference in require() and include() except they handle error conditions.
- It is recommended to use the require() function instead of include(), because scripts should not continue executing if files are missing or misnamed.

Syntax:

require '*filename*';

- Example with require() function following two examples where file does not exist then we will get different results.

Example:

- <html>
  <body>
  <?php include("xxmenu.php"); ?>
  This is an example to show how to include wrong PHP file!
  </body>
  </html>
- This will produce following result

  This is an example to show how to include wrong PHP file!

Example:

```
<html>
<body>
<?php require("xxmenu.php"); ?>
<p>This is an example to show how to include wrong
PHP file!</p>
</body>
</html>
```

This time file execution halts and nothing is displayed.

NOTE: You may get plain warning messages or fatal error messages or nothing at all. This depends on our PHP Server configuration.

The PHP Date() Function

- The PHP date() function formats a timestamp to a more readable date and time.
- A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.
- **Syntax**

  date (*format*, *timestamp*)

The PHP Date() Function

Parameter	Description
format:	Required. Specifies the format of the timestamp
timestamp:	Optional. Specifies a timestamp. Default is the current date and time

Note:

- The date() optionally accepts a time stamp if omitted then current date and time will be used.
- Any other data we include in the format string passed to date() will be included in the return value

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

Format	Description	Example
a	'am' or 'pm' lowercase	Pm
A	'AM' or 'PM' uppercase	PM
d	Day of month, a number with leading zeroes	20
D	Day of week (three letters)	Thu
F	Month name	January
h	Hour (12-hour format – leading zeroes)	12
H	Hour (24-hour format – leading zeroes)	22
g	Hour (12-hour format – no leading zeroes)	12

G	Hour (24-hour format – no leading zeroes)	22
i	Minutes (0 – 59)	23
j	Day of the month (no leading zeroes	20
l (Lower 'L')	Day of the week	Thursday
L	Leap year ('1' for yes, '0' for no)	1
m	Month of year (number – leading zeroes)	1
M	Month of year (three letters)	Jan
r	The RFC 2822 formatted date	Thu, 21 Dec 2000 16:01:07 +0200
n	Month of year (number – no leading zeroes)	2
s	Seconds of hour	20
U	Time stamp	948372444
y	Year (two digits)	06
Y	Year (four digits)	2006
z	Day of year (0 – 365)	206
[illegible]	Offset in seconds from GMT	+5

Example:

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

result:

- 01/20/00 13.27:55
- Today is 20 of January 2000, at 1.27 pm

PHP Date() – Adding a Timestamp

- The optional *timestamp* parameter in the date() function specifies a timestamp. If we do not specify a timestamp, the current date and time will be used.
- The mktime() function returns the Unix timestamp for a date.
- The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

Syntax for mktime()

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

Example:

```
<?php
   $tomorrow =
   mktime(0,0,0,date("m"),date("d")+1,date("Y"));
   echo "Tomorrow is ".date("Y/m/d", $tomorrow);
   ?>
```

- The output of the code above could be something like this:

Tomorrow is 2009/05/12

PHP Time() Function

- PHP's **time()** function gives us all the information that we need about the current date and time.
- It requires no arguments but returns an integer.
- The integer returned by time() represents the number of seconds elapsed since midnight GMT on January 1, 1970. This moment is known as the UNIX epoch, and the number of seconds that have elapsed since then is referred to as a time stamp.

Example:

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

Output:

- 948316201

Note: This is something difficult to understand. But PHP offers excellent tools to convert a time stamp into a form that humans are comfortable with.

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 we 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().

Key	Description	Example
seconds	Seconds past the minutes (0-59)	20
minutes	Minutes past the hour (0 – 59)	29
hours	Hours of the day (0 – 23)	22
mday	Day of the month (1 – 31)	11
wday	Day of the week (0 – 6)	4
mon	Month of the year (1 – 12)	7
year	Year (4 digits)	1997
yday	Day of year (0 – 365)	19
weekday	Day of the week	Thursday
Month	Month of the year	January
[illegible]	Time Since Epoch	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;
?>
```

It will produce following result:

seconds = 27
minutes = 25
hours = 11
mday = 12
wday = 6
mon = 5
year = 2007
yday = 131
weekday = Saturday
month = May
0 = 1178994327
Today's date: 12/5/2007

Printf function

- Print and echo function simply output text to the browser.
- Printf function controls the format of the output by using special characters in a string.
- For each formatting character printf expects to pass an argument that it will display format.
- Example:

  printf(“there are %d items in your basket”, 3);

List of printf conversion specifier.

Specifier	Conversion action on argument argument
%	Display a % character(no argument is required)
b	Display argument as a binary integer
c	Display the ASCII character for the argument
d	Display arguments as a signed decimal integer
e	Display argument using scientific notation
f	Display argument as floating point
o	Display argument as an octal integer
s	Display argument as a string
u	Display argument in unsigned decimal
x	Display argument in lowercase hexadecimal
X	Display argument in uppercase hexadecimal

Example:

- Suppose we want a color that has a triplet value of 65 red,127 green and 245 blue the hexadecimal value for these are
- Printf(“<font color=“#%X%X%X “Hello ”,65,127,245);

Precision setting

- In php we can also set the precision of the displayed result. For example: amounts of currency are usually displayed with only two digits of precision. However after calculation a value may have a greater precision than this e.g. $123.42/12, which result in $10.285).
- If we want to displayed only two digit of precision we can insert the string ".2" between the % symbol and the conversion specifier:

  Printf("the result is :$%.2f",123.42/12);

  Output is $10.29

String Padding

- We can also pad string to required lengths select different padding characters and even choose between left and right justification.

```
<php
echo “<pre>”;
$h=“house”;
Printf(“[%s]\n”,          $h);
Printf(“[%10s]\n”,        $h);
Printf(“[%-10s]\n”,       $h);
Printf(“[%010s]\n”,       $h);
?>
```

Sprintf function

- Using sprintf function we can send the output to another variable rather than to the browser.

  $hex string = sprintf(“%X%X%X” ,65,127,245);

- Or we may wish to store the output in a variable

  $out= sprintf(“the result is :$%.2f”,123.42/12);

  echo $out;