UNIT – 3
FUNCTIONS IN PHP

UNIVERSITY
OF PETROLEUM
& ENERGY STUDIES

Dr. P S V S Sridhar

Assistant Professor (SS)

Centre for Information Technology

Functions

- A function is a self-contained block of statements that performs a specific task . Functions are most useful when you need to use the same code in more than one place. Reusing existing code reduces costs, increases reliability, and improves consistency. Ideally, combining existing reusable components, with a minimum of development from scratch creates a new project. If you find that your code files are getting longer, harder to understand, and more difficult to manage, however, it may be an indication that you should start wrapping some of your code up into functions.

Some of the properties of a function:

- has a unique name.
- is independent and it can perform its task without intervention from or interfering with other parts of the program .
- can take some inputs(i.e arguments) for performing a task.
- returns a value to the calling program. This is optional and depends upon the task your function is going to accomplish.

Functions

- Functions are the heart of a well-organized script, making code easy to read and reuse.
- The basic syntax for using (or *calling) a function is:*

  function_name(expression_1, expression_2, ..., expression_n)
- sqrt(9); // square root function, evaluates to 3
- rand(10, 10 + 10); // random number between 10 and 20
- strlen(“This has 22 characters”); // returns the number 22
- pi(); // returns the approximate value of pi
- strrev (" .dlrow olleH"); //returns Hellow world.
- str_repeat("Hip ", 2);   //returns Hip two times
- strtoupper("hooray!"); //returns in capital HOORAY
- phpinfo() function prints out the internal configuration capabilities of your particular PHP installation

Built-in Functions in PHP

- Every language has a set of built-in functions (for example, string functions). For example: echo(“PHP”);
- print(“It is a server side programming language”);
- **Some of the Array functions in PHP are:**
- array() ◦ To Create an array
- sort() ◦ Sorts an array
- array_unique() ◦ Removes duplicate values from an array
- count() ◦ Counts no of elements in an array
- array_reverse() ◦ Returns an array in the reverse order

Built-in Functions in PHP

- **Some of the character functions in PHP are:**
- **ctype_upper()** ◦ Checks if all of the characters in the provided string are uppercase characters, return as 1.
- **ctype_digit()** ◦ Checks if all of the characters in the provided string, text, are numerical.
- **ctype_alpha()** ◦ Checks if all of the characters in the provided string, text, are alphabetic
- **ctype_alnum()** - Check for alphanumeric character(s)
- **ctype_xdigit()** - Check for character(s) representing a hexadecimal digit
- **is_numeric()** - Finds whether a variable is a number or a numeric string
- **is_int()** - Find whether the type of a variable is integer
- **is_string()** - Find whether the type of a variable is string

User defined functions

- A function is a way of wrapping up a chunk of code and giving that chunk a name, so that you can use that chunk later in just one line of code. Functions are most useful when you will be using the code in more than one place, but they can be helpful even in one-use situations, because they can make your code much more readable.
- **function *function-name ($argument-1, $argument-2, ..)***

  { *statement-1; statement-2;* ... }
- **That is, function definitions have four parts:**
- The special word **function**
- The name that you want to give your function
- The function's parameter list — dollar-sign variables separated by commas
- The function body — a brace-enclosed set of statements

Function definition example

```
function better_deal ($amount_1, $price_1,$amount_2, $price_2)
{
    $per_amount_1 = $price_1 / $amount_1;
    $per_amount_2 = $price_2 / $amount_2;
     return($per_amount_1 < $per_amount_2);
}
$liters_1 = 1.0; $price_1 = 1.59; $liters_2 = 1.5; $price_2 = 2.09;
if (better_deal($liters_1, $price_1, $liters_2,  $price_2))
   print("The first deal is better!<BR>");
else
   print("The second deal is better!<BR>");
```

Creating functions

```
<?php
        display();
        function display()
        {
        echo " This text was displayed by the function";
        }
?>
```

Call by Value

- The argument variable within the function is an "alias" to the actual variable
- But even further, the alias is to a *copy* of the actual variable in the function call

```
function double($alias)
  {    $alias = $alias * 2;    return $alias;}
$val = 10;
$dval = double($val);
echo "Value = $val Doubled = $dval\n";
```

- **Output:**

Value = 10 Doubled = 20

Call by Value/Returning Values

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

Call by Reference

- Sometimes we want a function to change one of its arguments - so we indicate that an argument is "by reference" using (&)

```
function triple(&$alias)
{    $alias = $alias * 3;}
$val = 10;
triple($val);
echo "Triple = $val\n";
```

Output:

- **Triple = 30**

Call 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 extra.'
?>
```

Argument number mismatches

- **Too few arguments**
- If you supply fewer actual parameters than formal parameters, PHP will treat the unfilled formal parameters as if they were unbound variables. However, under the usual settings for error reporting in PHP6, you will also see a warning printed to the browser.
- **Too many arguments**
- If you hand too many arguments to a function, the excess arguments will simply be ignored, even when error reporting is set to E_ALL.

Passing Arrays to Functions

We can pass arrays to functions as easily as simple data items like string s or numbers

```
<?php
$scores = array(57,58,39,67,59);
average($scores);
function average($array)
{       $t = 0;
        foreach($array as $val)
        $t = $t + $val;
        if(count($array > 0))
                echo "The average is ", $t/count($array);
        else
                echo "No elements to average";
}                                                       //67.8
?>
```

Returning Arrays

```
<?php
$data1 = create_array(3);
print_r($data1);
$data2 = create_array(4);
print_r($data2);
function create_array($number)
{
        for($counter = 0; $counter < $number; $counter++)
        {       $array[] = $counter;    }
        return $array;
}
?>
```

Using Default Arguments

```
<?php
display(“The default argument is:”);
function display($greeting, $message = “hello there”)
{
        echo $greetings;
        echo $message;
}
?>
```

Using Default Arguments

```
<?php
function makecoffee($type = "cappuccino")
{
    return "Making a cup of $type.\n";
}
echo makecoffee();
echo makecoffee(null);
echo makecoffee("espresso");
?>
```

Making a cup of cappuccino.

Making a cup of .

Making a cup of espresso.

Returning a reference from a function

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

```
<?php


$t = &r_r(10);

echo $t;

function &r_r($s)

{

return($s);

}

?>
```

PHP Variable Scopes

- The scope of a variable is the part of the script where the variable can be referenced/used.
- PHP has four different variable scopes:
- local
- global
- static
- parameter

Functions and variable scope

- The scope of a variable defined inside a function is *local* by default, meaning that it has no connection with the meaning of any variables outside the function.
- The syntax of this declaration is simply the word **global**, followed by a comma-delimited list of the variables that should be treated that way, with a terminating semicolon.
- global $count1, $count2;

Ex1:

```
<?php
function myfunction()
{
   $GLOBALS["n1"] = 10;
}
$n1 = 20;
myfunction();
echo $n1;
?>
```

Ex2:

```
<?php
 $x=5; // global scope
 $y=10; // global scope

 function myTest()
 {
 global $x,$y;
 $y=$x+$y;
 }

 myTest();
 echo $y; // outputs 15
 ?>
```

Static variables

- The static keyword allows for an initial assignment, which has an effect only if the function has not been called before. The first time the variable executes initial value assigned.
- The second time the function is called, it has the value it had at the end of the last execution.

```
<?php
 function myTest()
{
   static $x=0;
   echo $x;
   $x++;
}
myTest();
myTest();
myTest();
?>
```

Output:

012

Example

```
function SayMyABCs3 ()
{
static $count = 0; //assignment only if first time called
$limit = $count + 10;
while ($count < $limit)
{
print(chr(ord('A') + $count));  // chr () converts ASCII  to char  ord() returns ASCII value of char
$count = $count + 1;
}
print("<BR>Now I know $count letters<BR>");
}
$count = 0;
SayMyABCs3();
$count = $count + 1;
print("Now I've made $count function call(s).<BR>");
SayMyABCs3();
$count = $count + 1;
print("Now I've made $count function call(s).<BR>");
```

Output:

ABCDEFGHIJ
Now I know 10 letters
Now I've made 1 function call(s).
KLMNOPQRST
Now I know 20 letters
Now I've made 2 function call(s).

Parameter Scope

- A parameter is a local variable whose value is passed to the function by the calling code.
- Parameters are declared in a parameter list as part of the function declaration:
- <?php

  function myTest($x)
  {
  echo $x;
  }

  myTest(5);

  ?>

Include and require

- It's very common to want to use the same set of functions across a set of web site pages, and the usual way to handle this is with either include or require, both of which import the contents of some other file into the file being executed. Using either one of these forms is vastly preferable to *cloning* your function definitions (that is, repeating them at the beginning of each page that uses them).
- For example, at the top of a PHP code file we might have lines like:

include "basic-functions.inc"; (.inc means include)

include "advanced-function.inc";

- Both **include** and **require** have the effect of splicing in the contents of their file into the PHP code at the point that they are called. The only difference between them is how they fail if the file cannot be found. The **include** construct will cause **a warning** to be printed, **but processing of the script will continue**; **require**, on the other hand, will cause a **fatal error** if the file cannot be found.

- include "header.php"; - Pull the file in here
- include_once "header.php"; - Pull the file in here unless it has already been pulled in before
- require "header.php"; - Pull in the file here and die if it is missing
- require_once "header.php"; - You can guess what this means...
- These can look like functions - require_once("header.php");

```
<?php
define("premium", 12.09);
?>
```

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

Missing functions

- Sometimes depending on the version or configuration of a particular PHP instance, some functions may be missing. We can check that.

```
if (function_exists("array_combine"))
    {   echo "Function exists";}
else
    {   echo "Function does not exist";}
```

Variable functions

In PHP we can assign variable value as a function name.

```
<?php
$function_variable = "red";
$function_variable();
$function_variable = "white";
$function_variable("In white() now");
function red()
{
        echo "In red() now";
}
function white($argument)
{
        echo $argument;
}
?>
```

Nested functions

```
<?php
outer_function();
inner_function();
function outer_function()
{
        echo "Outer function";
function inner_function()
{
        echo "inner function";
}
}
?>
```

or

```
<?php
outer_function();
function outer_function()
{
        echo "Outer function";
         inner_function();
}
function inner_function()
{
        echo "inner function";
}
?>
```

Passing Variable Number of Arguments

- **func_num_args** – Returns the number of arguments passed
- **func_get_arg –** Returns a single argument
- **func_get_args –** Returns all arguments in an array

```
<?php
connector('How','are','things');
function connector()
{
$data = '';
$arguments = func_get_args();
for ($loop_index = 0; $loop_index <func_num_args(); $loop_index++)
$data .= $arguments[$loop_index] . ' ';
echo $data;
}
?>
```

output:

How are things

Returning Data from functions

```
<?php
echo connector('How','are','things');


function connector()
{
$data = “”;
$arguments = func_get_args();
for ($loop_index = 0; $loop_index <func_num_args(); $loop_index++)
$data .= $arguments[$loop_index] . ' ';
}
return $data;
?>
```

output:

How are things

Thanks