Unit: III
Lecture: 5
Functions in PHP (Part-II)

Setting Default argument values:
Default values are used in the event the function invocation is missing some arguments. i.e. PHP allows us to set default argument values for function parameters. If we do not pass any argument for a parameter with default value then PHP will use the default set value for this parameter in the function call.

Example:

```
<?php

// function with default parameter
function defAge($str, $num=12)
{
echo "$str is $num years old \n";
}

// Calling the function
defAge("Ram", 15);

// In this call, the default value 12
// will be considered
defAge("Divya");

?>
```

Output:

Ram is 15 years old

Divya is 12 years old

In the above example, the parameter $num has a default value 12, if we do no pass any value for this parameter in a function call then this default value 12 will be considered.

Parameters passing to Functions:

PHP allows us two ways in which an argument can be passed into a function:

- **Pass by Value:** On passing arguments using pass by value, the value of the argument gets changed within a function, but the original value outside the function remains unchanged. i.e. a duplicate of the original value is passed as an argument.

- **Pass by Reference:** on passing arguments as pass by reference, the original value is passed. Therefore, the original value gets altered. In pas by reference, we actually pass the address of the value, where it is stored using ampersand sign (&).

Example:

```
<?php
// pass by value
function valGeek($num) {
$num += 2;
return $num;
}
// pass by reference
function refGeek(&$num) {
$num += 10;
return $num;
}
$n = 10;
valGeek($n);
echo "The original value is still $n \n";
refGeek($n);
echo "The original value changes to $n";
?>
```

Output:

The original value is still 10

The original value changes to 20

Scope of variables in PHP:

The extent of a variable's visibility within the space of a PHP program is called the variable scope.

By default, variables used within a function are local- their impact is restricted to the function space alone and they cannot be viewed or manipulated from outside the function in which they exist.

e.g.

```
<?php
//function definition
//change the value of $score
function changeScore() {
$score=25;
}
//define a variable in the main program
//print its value
$score=11;
echo 'Score is: ' .$score;  //output: 11
//run the changeScore() function
changeScore();
//print $score again
echo 'Score is: ' .$score; //output:11
?>
```

Here, the variable $score is defined in the main program, and the changeScore() function contains code to change the value of this variable. However, after running this function, the value of $score remains at its original setting, because the changes made to $score within the changeScore() function remain "local" to the function and do not reflect in the main program.

If you want to import a variable from the main program into a function or vice versa, then PHP offers the global keyword.

When global keyword is applied to a variable inside a function, this keyword turns the variable into a global variable, making it visible both inside and outside the function.

e.g.

```
<?php
//function definition
//change the value of $score
function changeScore() {
global $score;
$score=25;
}
// define a variable in the main program
//print its value
$score =11;
echo 'Score is: ' . $score; //output: 11
//run the changeScore() function
changeScore();
//print $score again
echo 'Score is: ' .$score; //output: 25
```

In the above example, the global keyword used with the $score variable within the changeScore() function changes the scope of the variable, increasing its scope to encompass the entire program. As a result, changes made to the variable within the function will reflect in the main program (and vice versa).