LESSON 5:

USING PRE-DEFINED AND CREATING USER-DEFINED FUNCTIONS IN PHP

OBJECTIVES:

At the end of the lesson, students shall be able to:

- Define PHP function;
- State the importance of functions;
- Describe the use of different PHP built-in functions;
- Use the core PHP functions; and
- Develop and call user-defined functions.

DURATION: **4hour**

ASSESS YOURSELF

Name: ___________________________________ **Date:** ___________________
Course & Section: _________________________ **Score:** ___________________

TRUE or FALSE. Write the capital letter T if the given statement is correct and capital letter F if not on the space provided.

__________1. PHP functions are not similar to other programming languages
__________2. There are more than 1000 of built-in library functions created in PHP for diverse area and you only need to call them in accordance of your requirement.
__________3. PHP date() function create a timestamp to a more lucid date and time outputs.
__________4."**H**" is 24-hour format of an hour (00 to 23)
__________5. PHP user-defined function declaration begins with the word declare.

What are the Functions?

A PHP function is a self-contained block of code that is used to perform a specific task. It has a huge collection of built-in functions that you can call directly within your PHP scripts to execute a specific task, like the date(), print_r(), mktime(), var_dump, and many more.

When it comes to built-in functions, PHP also allows the programmers to define their PHP functions. It could be a way to make reusable code bundles that perform various tasks and can be stored and maintained separately by the main program.

REMEMBER A function must begin with a letter or maybe an underscore. Function names are NOT that case-sensitive.

Date and Time Functions

The Date() Function

The PHP **date()** function creates a timestamp to a more lucid date and time outputs. PHP Language structure of date() is: **date(format, timestamp)**

REMEMBER A timestamp is a grouping of characters, signifying the date and time at which a certain occasion happened.

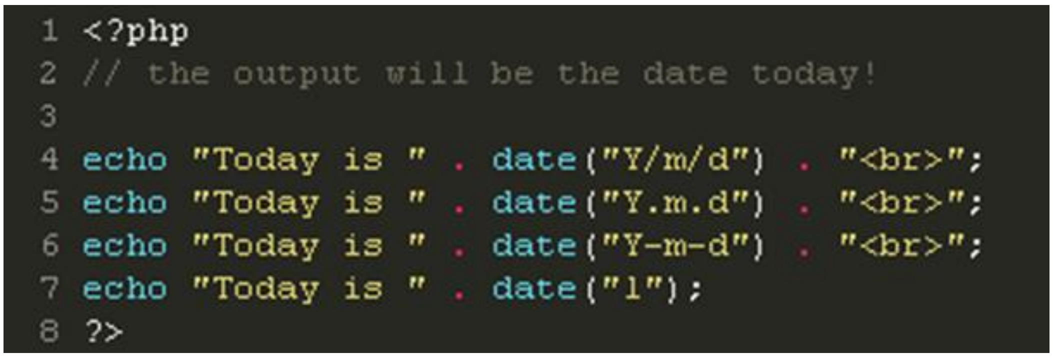
Getting a Date

The desired parameter of the date() function indicates how to arrange the date or time. Here are a few characters that are commonly utilized for dates:

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

There are other characters like"/", ".", or "-" it can be embedded between the characters to include extra formatting.

Example:



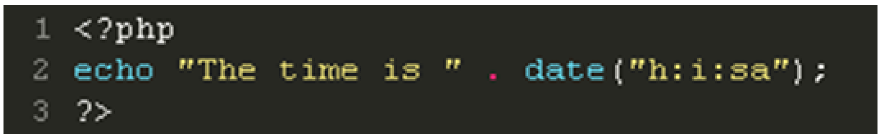
```php
<?php
// the output will be the date today!

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");
?>
```

Getting the Time

Here are a few characters that are commonly utilized for times:

- H - represents the 24-hour format of an hour (it's from 00 to 23)
- h - represents the 12-hour format of an hour (from 01 to 12)
- i - represents minutes with driving zeros (00 to 59)
- s - represents seconds with driving zeros (00 to 59)
- a - Lowercase a represents ante meridiem and post meridiem (am or pm)

Example:



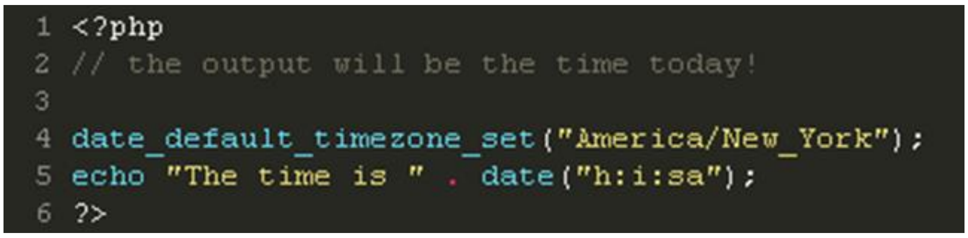
```php
<?php
echo "The time is " . date("h:i:sa");
?>
```

REMEMBER PHP function for the date() will return from the current date and time of the server!

Setting A Time Zone

On the off chance that the time you got back from the code is not correct, it's probably since your server is in another nation or set up for a diverse time zone. If you would like the time to rectify concurring to a particular area, you'll set the time zone you need to utilize.

Example:



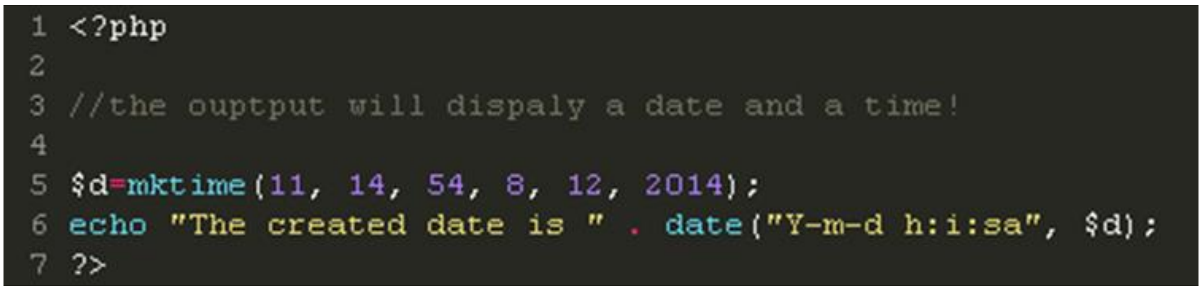
```php
<?php
// the output will be the time today!

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

Create A Date Using mktime() function

The timestamp parameter within the **date()** function indicates a timestamp. In case overlooked, the current date and time will be utilized (as within the examples).

The PHP **mktime()** returns the Unix timestamp for a date. The Unix timestamp encompasses the number of seconds between the Unix Age (January 1, 1970, 00:00:00 GMT) and the time indicated. The language structure of mktime() is: **mktime(hour, minute, second, month, day, year)**

Example:



```php
<?php

//the ouptput will dispaly a date and a time!

$d=mktime(11, 14, 54, 8, 12, 2014);
echo "The created date is " . date("Y-m-d h:i:sa", $d);
?>
```

Include and Require Functions

PHP permits us to form different functions and different elements that are utilized by numerous times on different pages. Scripting the same function in different pages could be an errand of incredible exertion and would devour time. It could be kept a strategic distance from if we take after and use the concept of file inclusion, which makes a difference to include various files, including text and codes, into a single file that saves the effort of composing the total function or code multiple times. It also gives another advantage. If we need to alter any code at that point rather than altering it in all the files, we just have to be compelled to edit the source record, and all codes will be consequently changed.

PHP has two ways which help us to include files:

1. function **include()** - will only produce a warning (E_WARNING), and the script will continue.
2. function **require()** - will produce a fatal error (E_COMPILE_ERROR) and stop the script.

The basic language structure of the include() and require() statements can be given with:

```
include("path/to/filename"); -Or- include "path/to/filename";
require("path/to/filename"); -Or- require "path/to/filename";
```

Just like print and echo statements, you can exclude the parentheses while utilizing the include and require functions.

The include() Function

This function is utilized to duplicate all the data of a record called within the function, content shrewd into a file from which it is called. It happens before the server executes the block of code. The following illustration will illustrate how to include the common header, footer, and menu codes, which are put away in separate 'header.php,' 'footer.php,' and 'menu.php' files individually, within all the pages of your site. Utilizing this strategy, you'll be able to update all pages of the site at once by doing the changes to just a sole file, this saves a lot of monotonous work.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>My Sample Script</title>
</head>
<body>
<?php include "header.php"; ?>
<?php include "menu.php"; ?>
    <h1>Welcome to BULSU!</h1>
    <p>Bulacan State University.</p>
<?php include "footer.php"; ?>
</body>
</html>
```

Comparing include() function and require() function

You may be guessing if we can include files using the include() function and why we should need to use the require() function. Typically the require() function operates just like the include().

The only distinction is, the include() function will only produce a PHP warning but permit script execution to proceed if the file to be included is not found. At the same time, the require() function will create a fatal error and stops the script execution.

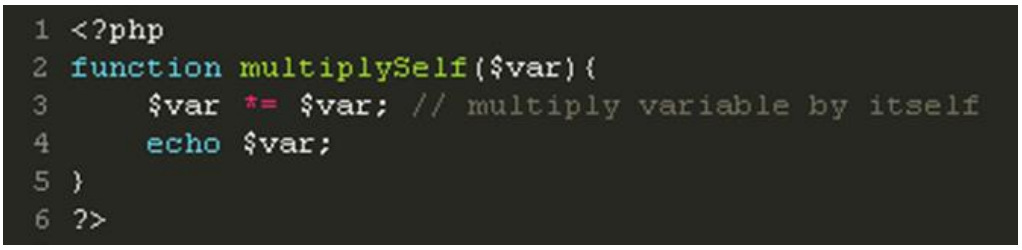
TIPS & TRICKS

It is suggested to use the require() function if you're including the library files or files containing the functions and configuration variables that are fundamental for running your application, such as database setup file.

Using include_once() and require_once()

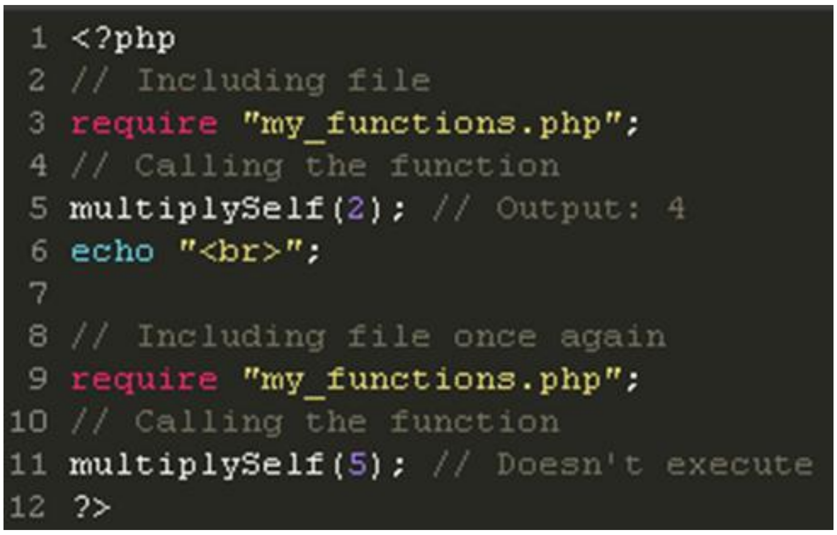
In case you inadvertently include the same file more than one time within your code using the include or require functions, it may cause crashes. To avoid this circumstance, PHP gives include_once and require_once statements. These statements carry on within the same way as to include and require functions with one exception.

The include_once and require_once statements will only include the file once even if inquired about including it a second time, in case the required file has already been included in a previous statement, the file isn't included once more. To way better get it how it works, let's check out an illustration. Assume we have a file named my_functions.php with the following code:



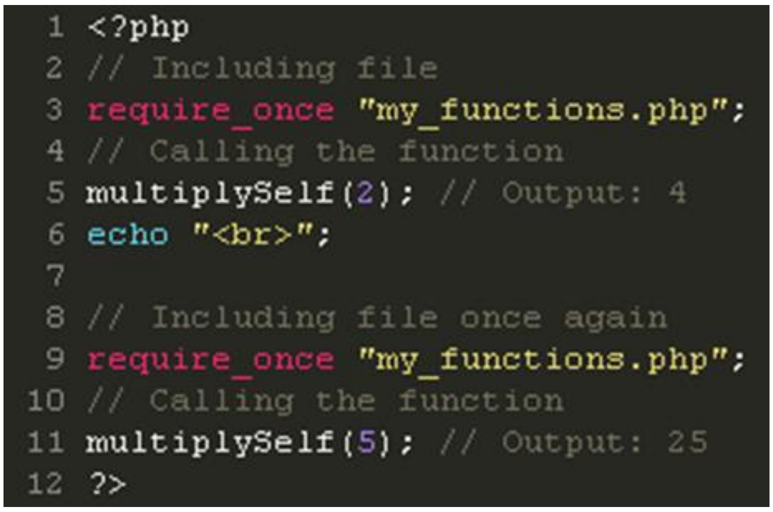
```php
<?php
function multiplySelf($var){
    $var *= $var; // multiply variable by itself
    echo $var;
}
?>
```

Here is the PHP script within which we've included the 'my_functions.php' file:



```php
<?php
// Including file
require "my_functions.php";
// Calling the function
multiplySelf(2); // Output: 4
echo "<br>";

// Including file once again
require "my_functions.php";
// Calling the function
multiplySelf(5); // Doesn't execute
?>
```

When you try to run the above sample script, you will see an error message saying: "**Fatal error: Cannot redeclare multiplySelf()**." It happens due to the 'my_functions.php' included twice. It means the function multiplySelf() is defined twice, which caused the PHP to stop the script execution and then generate a fatal error. You can rewrite the above example with the use of require_once.



```
<?php
// Including file
require_once "my_functions.php";
// Calling the function
multiplySelf(2); // Output: 4
echo "<br>";

// Including file once again
require_once "my_functions.php";
// Calling the function
multiplySelf(5); // Output: 25
?>
```

As in the figures shown, by using require_once rather than require, the script works as expected.

Creating and Calling User-defined Functions

Other than built-in PHP functions, it's possible to create your functions.

- A PHP function is a block or piece of statements that can be used multiple times in a program.
- A PHP function does not execute systematically when a page loads.
- A PHP function will be executed by calling it.

Creating a User-defined Function

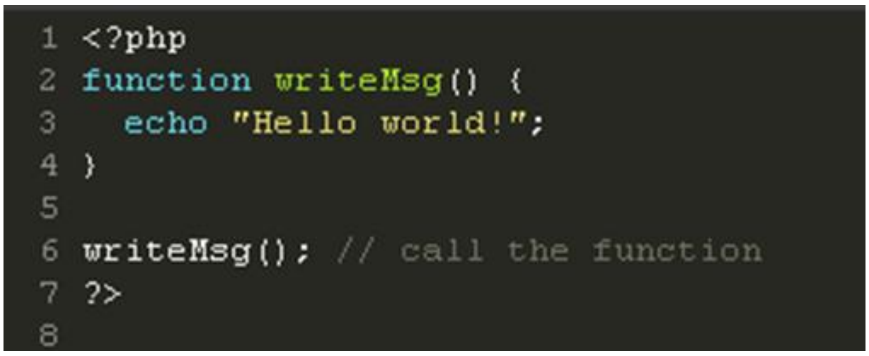
In declaring a PHP user-defined function the declaration must begin with the word function. Language Structure of a function is:

```
function functionName() {
        place code to be executed;
}
```

Always remember that a PHP function name must begin with a letter or an underscore. PHP Function names are NOT that case-sensitive.

In the given example below, we compose a function named "writeMsg()". The opening bracket({) specifies the start of the function code, and the closing brachet (}) specifies the ending of the function.

The function displays "Hello world!" To call a function, just simply write its function name followed by parentheses(**()**).



```php
<?php
function writeMsg() {
  echo "Hello world!";
}

writeMsg(); // call the function
?>
```

PHP functions are not that much different from other programming languages. A function is a piece or block of code that takes one or more input in the form of a parameter and makes some processing and then returns a value.

There are two partitions which must be clear to you:

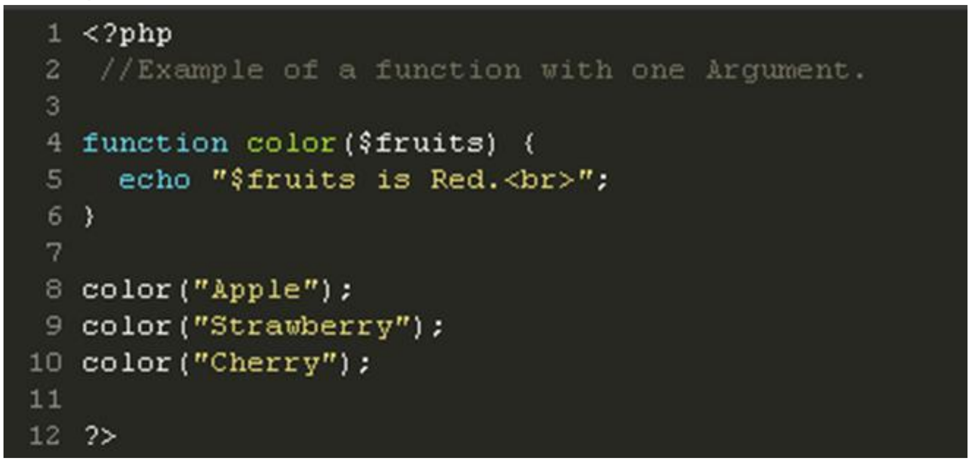
1. **Creating a Function** – It is very helpful to make your PHP function. Assume that you need a PHP function that can write a simple message on the browser after you call it. The following example defines the writeMsg() function and calls it after its creation.
2. **Calling a Function** – In reality, you hardly got to create your PHP function since there are now more than 1000 built-in library functions created for a diverse range, and you just have to call them according to what you want.

Give your function a name that represents what the function does!

Functions with Arguments

In PHP, an argument is like a variable. All the information can be passed to a function using these so-called Arguments. Arguments are indicated after the function name inside the open and close parentheses. You can add many arguments as many as you need but remember to separate them with a comma.

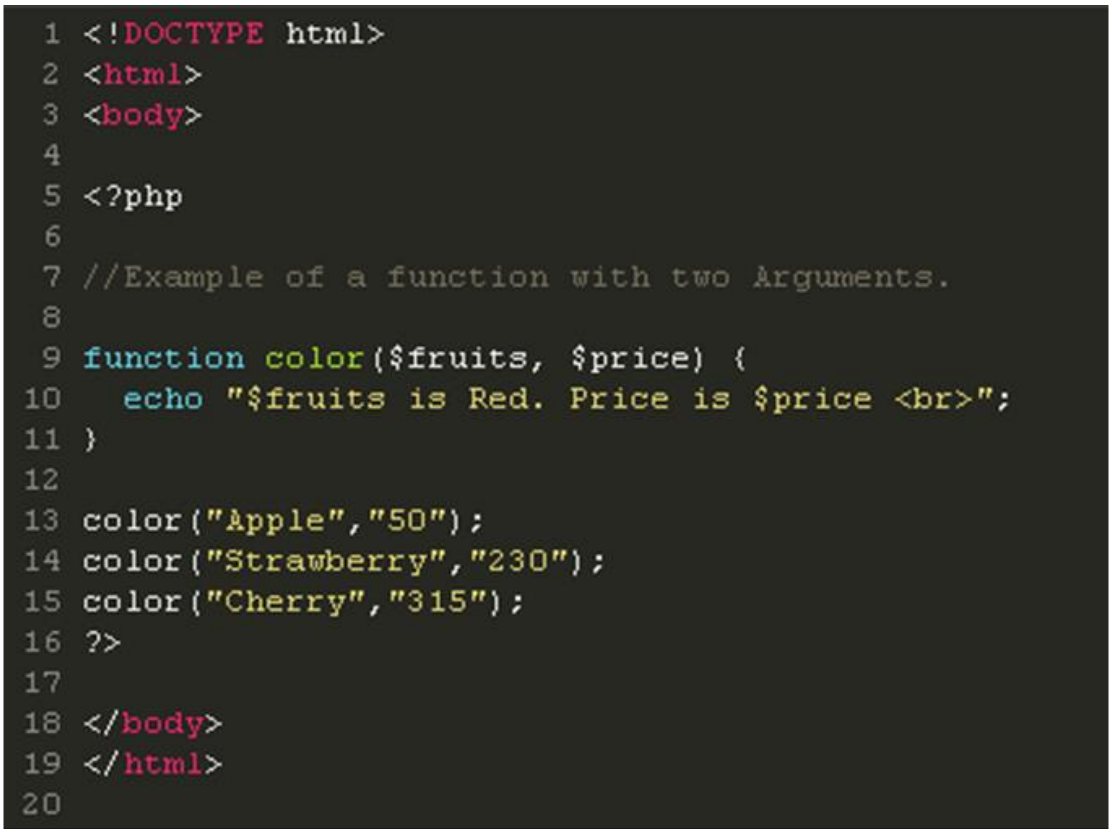
Example:



```php
<?php
 //Example of a function with one Argument.

function color($fruits) {
  echo "$fruits is Red.<br>";
}

color("Apple");
color("Strawberry");
color("Cherry");

?>
```

The example above has a function with one argument ($fruits). When the color() function is called, it passes along a fruit (e.g., Apple), and the **$fruits variable** is used inside the function, which outputs several different fruits, but the same color.

Example:



```
<!DOCTYPE html>
<html>
<body>

<?php

//Example of a function with two Arguments.

function color($fruits, $price) {
  echo "$fruits is Red. Price is $price <br>";
}

color("Apple","50");
color("Strawberry","230");
color("Cherry","315");
?>

</body>
</html>

```

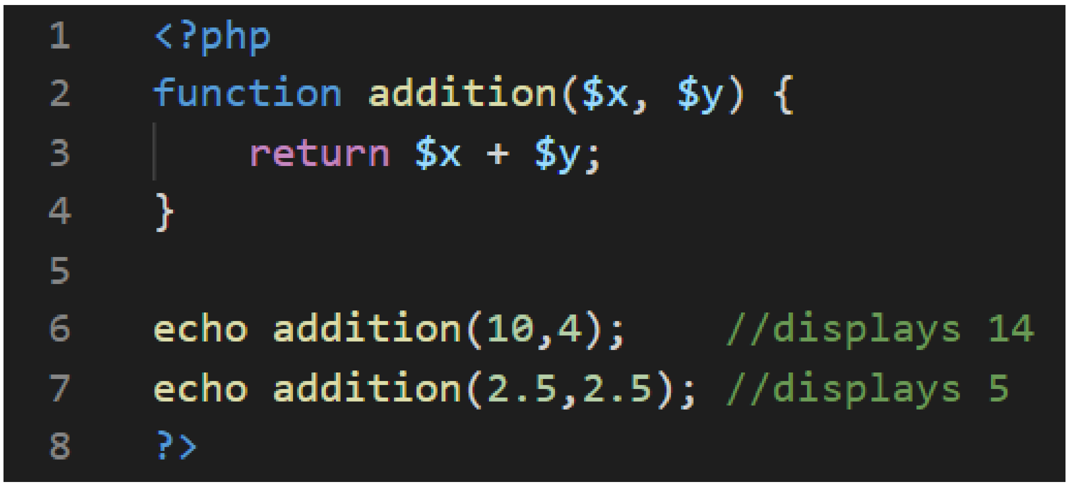
This next example has a function with two arguments ($fruits and $price).

PHP Function Return Values

The return statement permits a function to return a value to its caller. To create functions with return values, add the **return** statement at your function.

Once your function encountered the **return** statement, any code or script below the return statement will no longer be executed. Position your return statement wisely in your scripts.

Example:

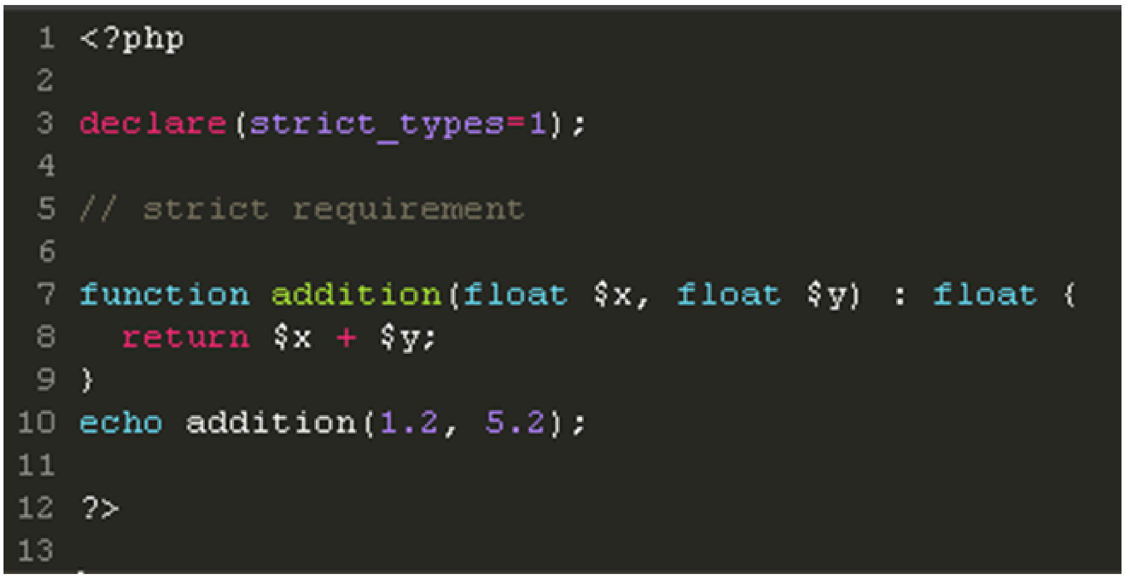


```
<?php
function addition($x, $y) {
    return $x + $y;
}

echo addition(10,4);     //displays 14
echo addition(2.5,2.5); //displays 5
?>
```

PHP also supports **Type Declarations** for the return statement. Same with type declaration in function arguments, by enabling the strict requirement, it will display a "Fatal Error" on a type mismatch.

To declare a type for return function, remember to add a **colon (:)** and the **type** right before the **opening brace ({)** when declaring function.

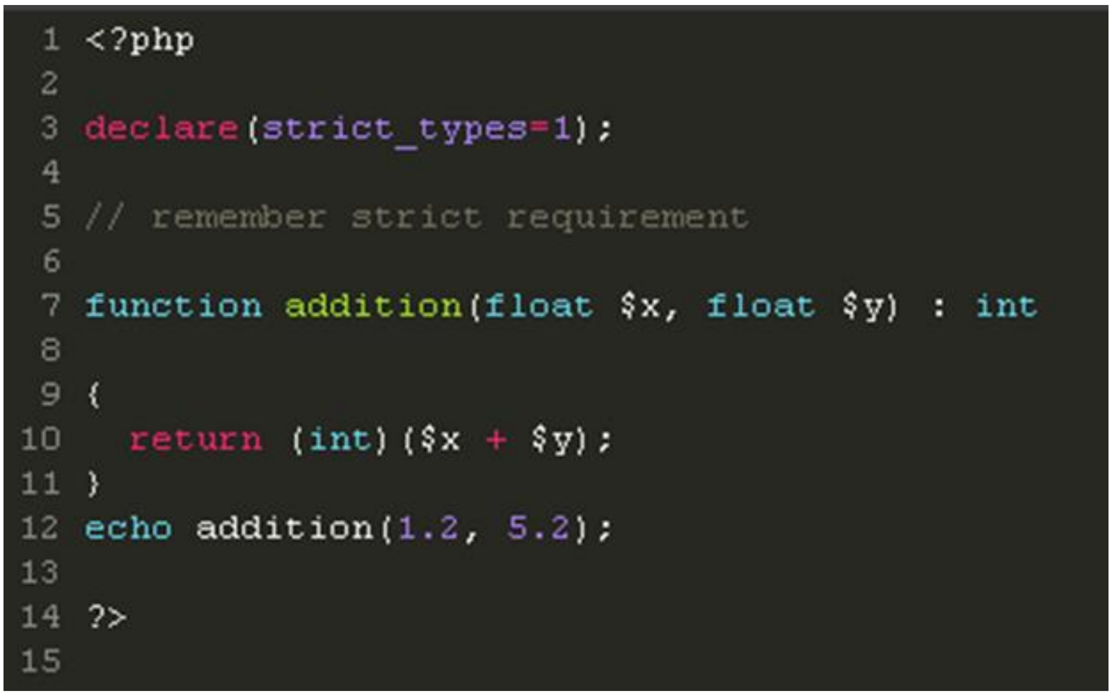
Example:



```php
<?php

declare(strict_types=1);

// strict requirement

function addition(float $x, float $y) : float {
  return $x + $y;
}
echo addition(1.2, 5.2);

?>

```

The example above specifies the return type (float) for the function **addition()**. In the next example, we could set its return type to a different one.

Example:



```php
<?php

declare(strict_types=1);

// remember strict requirement

function addition(float $x, float $y) : int

{
  return (int)($x + $y);
}
echo addition(1.2, 5.2);

?>

```