# Unit-4

# Introduction to PHP

## Content

✓Introduction to PHP

✓Datatypes

✓Control Statements

✓ Loops

✓ Functions

✓Embedding PHP in HTML & MySQL

## What is PHP?

**PHP is a server side scripting language, and a powerful tool for making dynamic and interactive Web pages**

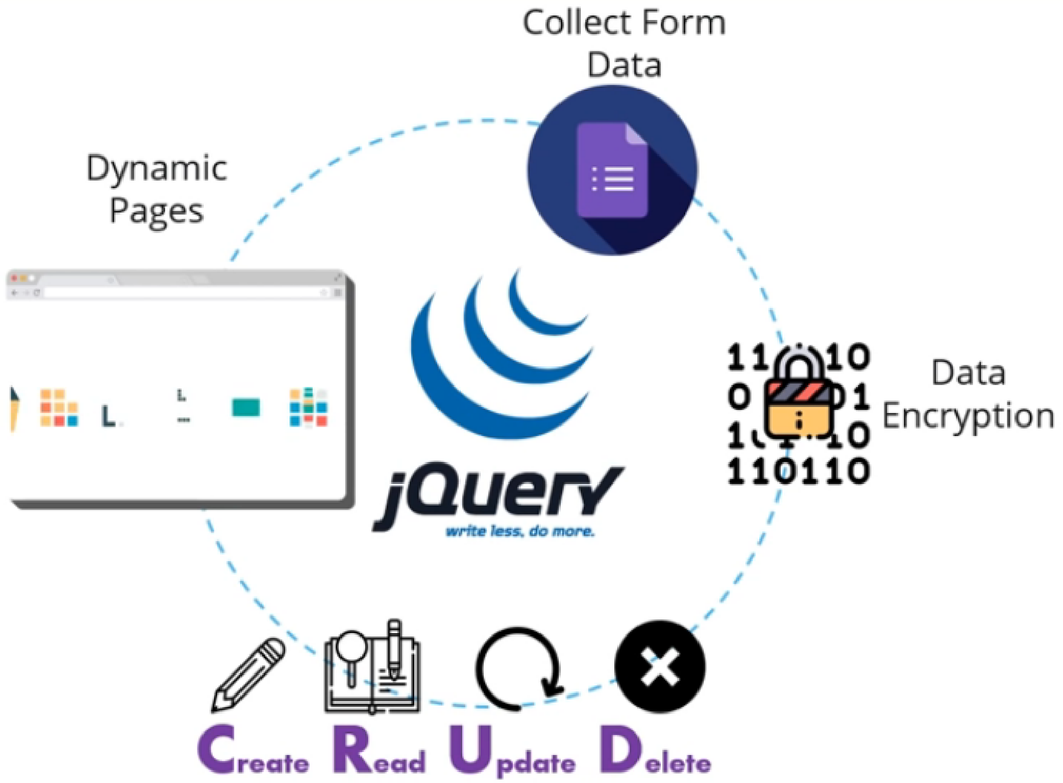
Hypertext Pre-processor

Used for server side logic

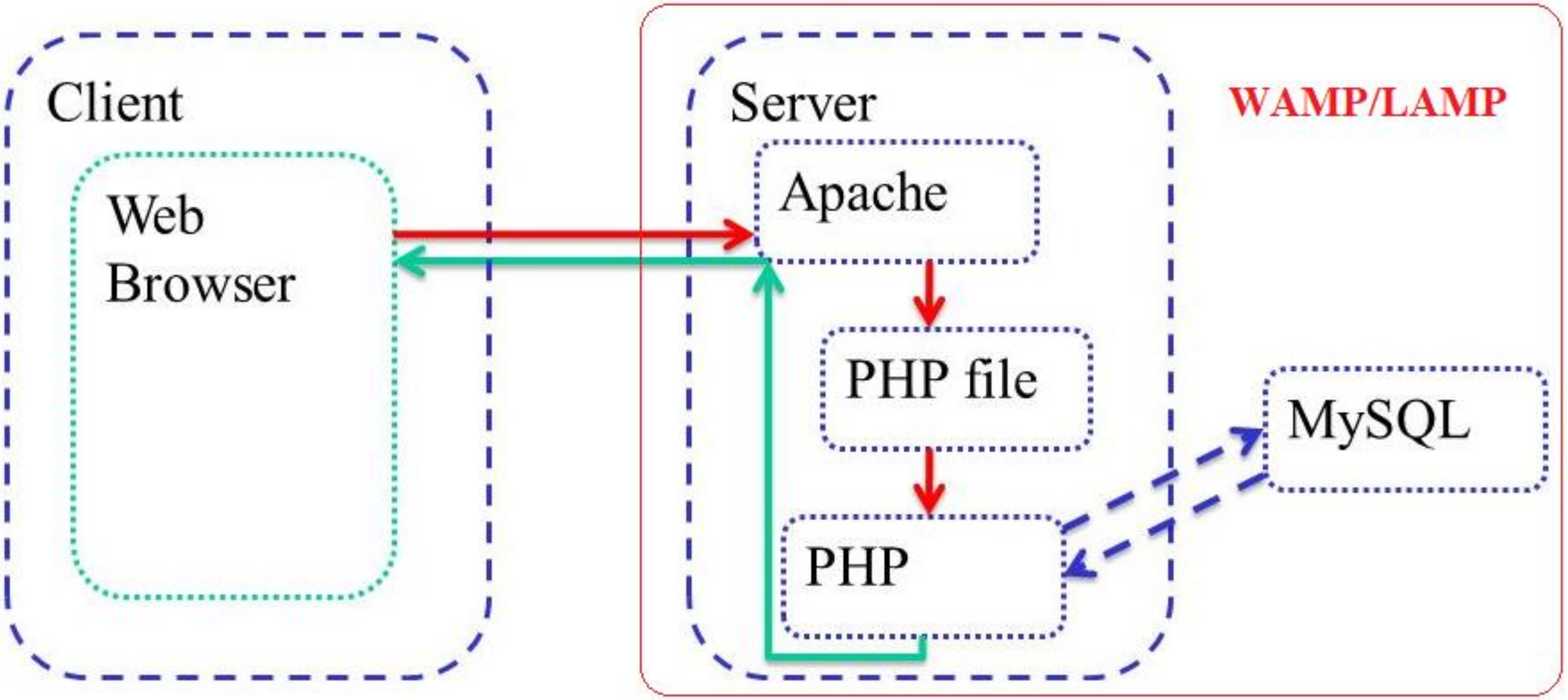
## What can PHP do?

PHP Features

- Easy to Learn
- Supports different types of DB
- Cross Platform
- Supports multiple server types



# PHP Architecture



This is often referred to as a LAMP (or WAMP) stack:

**Linux (or Windows) Apache MySQL PHP**

# What is PHP?

- → The PHP Hypertext Pre-processor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases.
- → PHP is basically used for developing web based software applications.
- → PHP was written in the C programming language by Rasmus Lerdorf in 1994
- → Rasmus used PHP for monitoring his online resume and related personal information.
- → **Originally PHP was an acronym for "Personal Home Page".**
- → **PHP processor has two modes of operations:**

  **1. Copy mode 2. interpret mode**

## Environment setup

→ PHP Parser-Parser must be installed to generate HTML output that can be sent to the Web Browser.

→ Database-Oracle and Sybase but most commonly used is freely available MySQL database.

→ Web Server-all Web Server software, including (IIS) but then most often used is freely available *Apache Server*.

## Where to add PHP script?

→ All PHP code must be included inside one of the three special markup tags are recognised by the PHP Parser.

→ <?php PHP code goes here ?>

→<script language="php"> PHP code goes here </script>

## PHP: Print construct

→ echo – Used to print message/string on console or web page.

Example: echo "Hello by PHP echo";

→ If more than one parameter is used for displaying output, then use echo with parenthesis

Syntax: void echo ( string $arg1 [, string $... ] )

→ print() – printing multi line or single line string with or without parenthesis

→ printf() – Printing multi line or single line string with parenthesis

→ escaping characters

→ echo “Hello!\“Welcome to PHP \” scripting language”

→ print “Hello!\“Welcome to PHP \” scripting language”

→ printf (“Hello!\“Welcome to PHP \” scripting language”)

## PHP Variable Types

The main way to store information in the middle of a PHP program is by using a variable.

Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign ($).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

PHP has a total of eight data types which we use to construct our variables:

## PHP Variables

→ A variable is declared using $ sign followed by variable name.
Syntax: $variablename=value;

→ PHP variables must start with letter or underscore only.

→ **PHP variable can't be start with numbers and special symbols.**

→ Example:

```
<?php
$a="hello";//letter (valid)
$_b="hello";//underscore (valid)

echo "$a <br/> $_b";
?>
```

## Variable scope

→PHP has three different variable scopes:
- →local
- →global
- → static

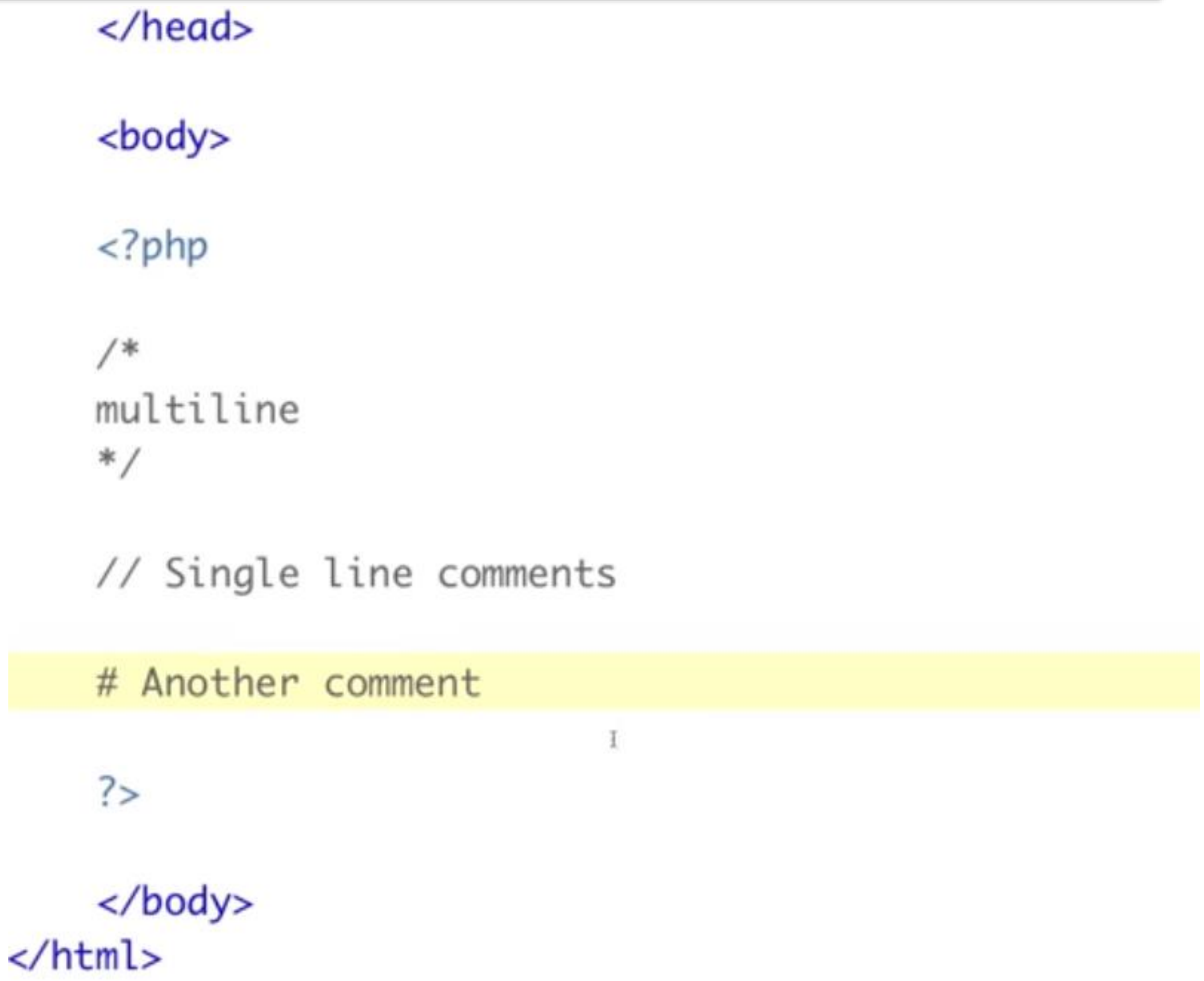
```
<?php
    $str="hello string";
    $a=23;
    $b=33.333;
    echo "string is: $str <br/>";
    echo "integer is: $a <br/>";
    echo "float is: $b <br/>";
?>
```

```
<?php
    $x=5;
    $y=6;
    $z=$x+$y;
    echo $z;
?>
```

## Variable Types

- **Integers:** are whole numbers, without a decimal point, like 4195.
- **Doubles:** are floating-point numbers, like 3.14159 or 49.1.
- **Booleans:** have only two possible values either true or false.
- **NULL:** is a special type that only has one value: NULL.
- **Strings:** are sequences of characters, like 'PHP supports string operations.'
- **Arrays:** are named and indexed collections of other values.
- **Objects:** are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources:** are special variables that hold references to resources external to PHP (such as database connections).

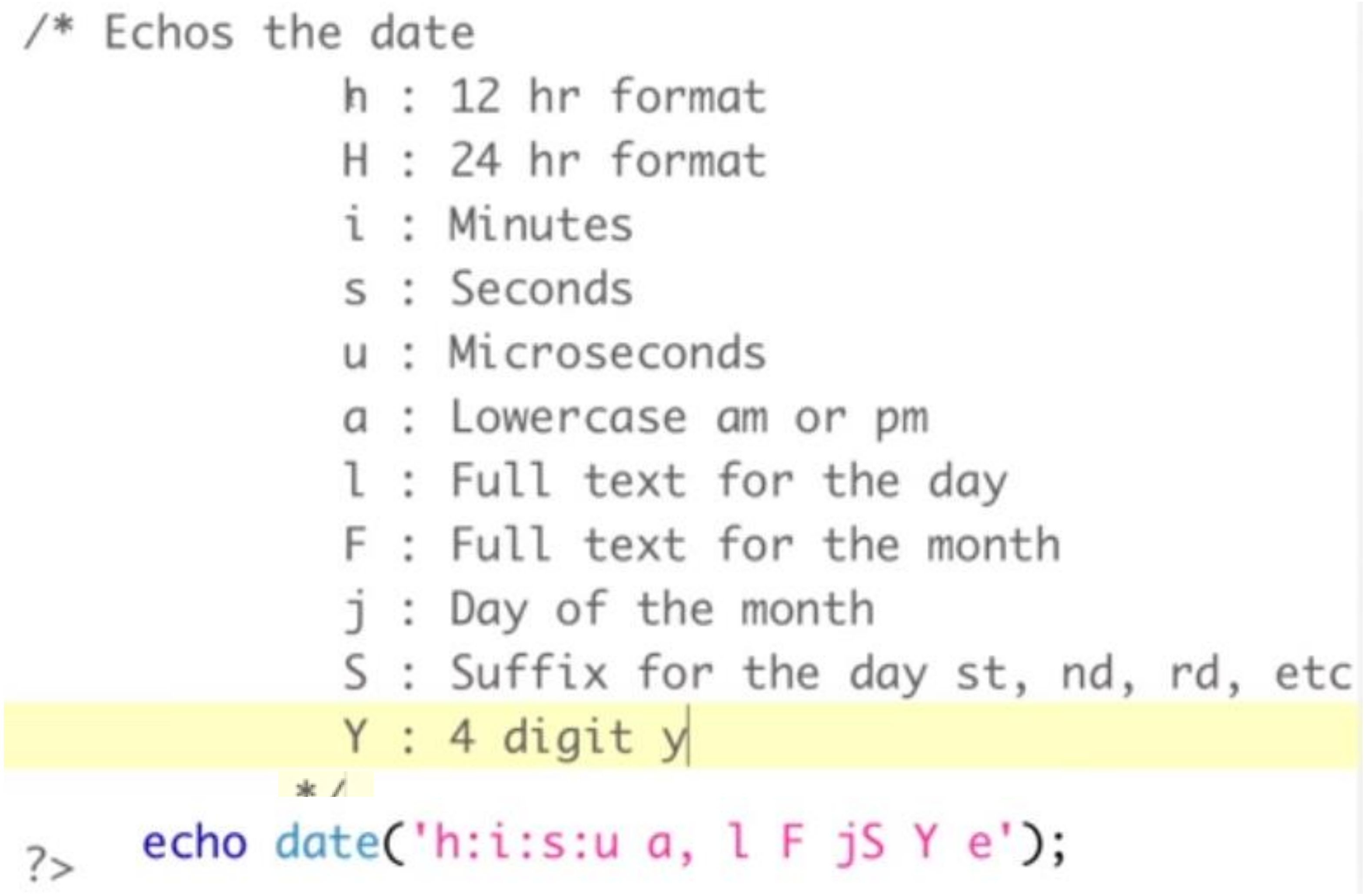
# Comments in PHP



```
    </head>

    <body>

    <?php

    /*
    multiline
    */

    // Single line comments

    # Another comment

    ?>

    </body>
</html>
```

## Date example

```
<?php date_default_timezone_set('UTC');
```



```
/* Echos the date
            h : 12 hr format
            H : 24 hr format
            i : Minutes
            s : Seconds
            u : Microseconds
            a : Lowercase am or pm
            l : Full text for the day
            F : Full text for the month
            j : Day of the month
            S : Suffix for the day st, nd, rd, etc
            Y : 4 digit y
          */
?>   echo date('h:i:s:u a, l F jS Y e');
```

## PHP is case sensitive:

```
<html>
<body>
<?
$capital = 67;
print("Variable capital is $capital<br>");
print("Variable CaPiTaL is $CaPiTaL<br>");
?>
</body>
</html>
```

This will produce following result:

```
Variable capital is 67
Variable CaPiTaL is
```

## Contd.

→Statements or expressions terminated by semicolons:

Example:

```
$greeting = “Welcome to php programming” ;
```

## PHP Operator Types

- Arithmetic Operators
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

| Operator | Name | Example | Result |
|---|---|---|---|
| + | Addition | $x + $y | Sum of $x and $y |
| - | Subtraction | $x - $y | Difference of $x and $y |
| * | Multiplication | $x * $y | Product of $x and $y |
| / | Division | $x / $y | Quotient of $x and $y |
| % | Modulus | $x % $y | Remainder of $x divided by $y |
| ** | Exponentiation | $x ** $y | Result of raising $x to the $y'th power (Introduced in PHP 5.6) |

## Contd…

| Assignment | Same as... | Description |
|---|---|---|
| x = y | x = y | The left operand gets set to the value of the expression on the right |
| x += y | x = x + y | Addition |
| x -= y | x = x - y | Subtraction |
| x *= y | x = x * y | Multiplication |
| x /= y | x = x / y | Division |
| x %= y | x = x % y | Modulus |

## Contd…

| Operator | Name | Example | Result |
|---|---|---|---|
| == | Equal | $x == $y | Returns true if $x is equal to $y |
| === | Identical | $x === $y | Returns true if $x is equal to $y, and they are of the same type |
| != | Not equal | $x != $y | Returns true if $x is not equal to $y |
| <> | Not equal | $x <> $y | Returns true if $x is not equal to $y |
| !== | Not identical | $x !== $y | Returns true if $x is not equal to $y, or they are not of the same type |
| > | Greater than | $x > $y | Returns true if $x is greater than $y |
| < | Less than | $x < $y | Returns true if $x is less than $y |
| >= | Greater than or equal to | $x >= $y | Returns true if $x is greater than or equal to $y |
| <= | Less than or equal to | $x <= $y | Returns true if $x is less than or equal to $y |

## Contd…

| Operator | Name | Description |
|---|---|---|
| ++$x | Pre-increment | Increments $x by one, then returns $x |
| $x++ | Post-increment | Returns $x, then increments $x by one |
| --$x | Pre-decrement | Decrements $x by one, then returns $x |
| $x-- | Post-decrement | Returns $x, then decrements $x by one |

## Logical Operators:

| Operator | Description | Example |
|---|---|---|
| and | Called Logical AND operator. If both the operands are true then then condition becomes true. | (A and B) is true. |
| or | Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true. | (A or B) is true. |
| && | Called Logical AND operator. If both the operands are non zero then then condition becomes true. | (A && B) is true. |
| \|\| | Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true. | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is false. |

## String operator

| Operator | Name | Example | Result |
|---|---|---|---|
| . | Concatenation | $txt1 . $txt2 | Concatenation of $txt1 and $txt2 |
| .= | Concatenation assignment | $txt1 .= $txt2 | Appends $txt2 to $txt1 |

## PHP Decision Making

The if, elseif ...else and switch statements are used to take decision based on the different condition.

You can use conditional statements in your code to make your decisions. PHP supports following threedecision making statements:

- **if...else statement** - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- **elseif statement** - is used with the if...else statement to execute a set of code if **one** of several condition are true
- **switch statement** - is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

**Syntax**

```
if (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

## Syntax

```
if (condition)
  code to be executed if condition is true;
elseif (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

## Syntax

```
switch (expression)
{
case label1:
  code to be executed if expression = label1;
  break;
case label2:
  code to be executed if expression = label2;
  break;
default:
  code to be executed
  if expression is different
  from both label1 and label2;

}
```

## PHP Loop Types

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **for** - loops through a block of code a specified number of times.
- **while** - loops through a block of code if and as long as a specified condition is true.
- **do...while** - loops through a block of code once, and then repeats the loop as long as a special condition is trur.
- **foreach** - loops through a block of code for each element in an array.

### Syntax

```
for (initialization; condition; increment)
{
  code to be executed;
}
```

### Syntax

```
while (condition)
{
    code to be executed;
}
```

## Syntax

```
do
{
    code to be executed;
}while (condition);
```

## The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

### Syntax

```
foreach (array as value)
{
    code to be executed;

}
```

## Arrays:

→ An array is a data structure that stores one or more similar type of values in a single variable.

→ An array stores multiple values in one single variable.

→ Types:

1. Numeric array: Array index will be numeric
2. Associative array: Array index will be string value
3. Multidimensional array: Multiple array index with numerical values

## Declaration of Array

→ Two ways to create an array in PHP

→First Method:

Empty Array declaration:

```
//Array with size 5
$MyArray[] = 5;
```

→Second Method:

```
// Array with unknown size
        $Arr = array();
```

# array() function

→ array() function is used to create array variable.

→ Example:

```
//Numerical array;
            $Num_arr = array( 1, 2, 3, 4, 5);
                        $Num_arr[0] // Used to access array value
// Associative array;
            $MyArray = array("College” => 'sdmcet', 'City' => 'Dharwad', 'Code' =>'2SD');
            $Assc_arr2 = array(“Dharwad”=>0836, “Bangaluru”=>080)
            $ MyArray ["College” ] // Used to access array value
            $Assc_arr2 [“Dharwad”]
//Multidimensional array;
            $Mult_arr1 = array(array(“pen drive”, ”hard disk”, “CD-ROM”), array(111, 222,333),
array(“PPT”, “docx”))
            $Mult_arr[2][1];
            $Mult_arr2  = array(“abc” => array(“web”=>45, “oomd”=> 35, “CN” => 55), “def” =>
array(“c”=>34, “phy”=>44))
            $Mult_arr2[abc][web]
```

## Cont'd…

→Functions to print array elements:

i) print_r(array_values(ArrayName))

The array_values() function returns an array containing all the values of an array.

→The Length of an Array - The count() Function

i) The count() function is used to return the length (the number of elements) of an array

## Examples: Arrays

```
<?php
    $num = array(10,20,30,40,50);
     for($i=0;$i<count($num);$i++)
         echo "Element at $i=".$num[$i]."<br>";
?>
```

```
<?php
 //Printing String array elements using foreach loop
    $colors = array("Red", "Green", "Blue", "Yellow", "Orange");
     // Loop through colors array
       foreach($colors as $value){
                    echo $value . "<br>";
           }
?>
```

## Examples: Arrays

```
//Loop through Associative array
<?php
      $StrArray = array("David"=>34,"John"=>56,"rick"=>65);
          foreach ($StrArray as $i => $i_value) {
              echo "Key=".$i.", Value=". $i_value."<br />";
                    }
?>
```

```
<?php
     $num = array(10,20,30,40,50);
      print_r($num);
?>
```

## Examples: Arrays

```
<?php
//printing Multidimensional Array elements
$M_AscArray =
array(array(1,2,3),array(11,22,33,44),array(10,20,30,40));
          for($row=0; $row<3;$row++)
                    for($col=0;$col<3;$col++)
                              echo    $M_AscArray[$row][$col]."<br/>";
          ?>
```

| Function | Description |
|---|---|
| array() | Creates an array |
| array_change_key_case() | Changes all keys in an array to lowercase or uppercase |
| array_chunk() | Splits an array into chunks of arrays |
| array_column() | Returns the values from a single column in the input array |
| array_combine() | Creates an array by using the elements from one "keys" array and one "values" array |
| array_count_values() | Counts all the values of an array |
| array_diff() | Compare arrays, and returns the differences (compare values only) |
| array_diff_assoc() | Compare arrays, and returns the differences (compare keys and values) |
| array_diff_key() | Compare arrays, and returns the differences (compare keys only) |
| array_diff_uassoc() | Compare arrays, and returns the differences (compare keys and values, using a user-defined key comparison function) |

| | |
|---|---|
| array_diff_ukey() | Compare arrays, and returns the differences (compare keys only, using a user-defined key comparison function) |
| array_fill() | Fills an array with values |
| array_fill_keys() | Fills an array with values, specifying keys |
| array_filter() | Filters the values of an array using a callback function |
| array_flip() | Flips/Exchanges all keys with their associated values in an array |
| array_intersect() | Compare arrays, and returns the matches (compare values only) |
| array_intersect_assoc() | Compare arrays and returns the matches (compare keys and values) |
| array_intersect_key() | Compare arrays, and returns the matches (compare keys only) |
| array_intersect_uassoc() | Compare arrays, and returns the matches (compare keys and values, using a user-defined key comparison function) |
| array_intersect_ukey() | Compare arrays, and returns the matches (compare keys only, using a user-defined key comparison function) |
| array_key_exists() | Checks if the specified key exists in the array |
| array_keys() | Returns all the keys of an array |
| array_map() | Sends each value of an array to a user-made function, which returns new values |
| array_merge() | Merges one or more arrays into one array |
| array_merge_recursive() | Merges one or more arrays into one array recursively |
| array_multisort() | Sorts multiple or multi-dimensional arrays |

| | |
|---|---|
| array_pad() | Inserts a specified number of items, with a specified value, to an array |
| array_pop() | Deletes the last element of an array |
| array_product() | Calculates the product of the values in an array |
| array_push() | Inserts one or more elements to the end of an array |
| array_rand() | Returns one or more random keys from an array |
| array_reduce() | Returns an array as a string, using a user-defined function |
| array_replace() | Replaces the values of the first array with the values from following arrays |
| array_replace_recursive() | Replaces the values of the first array with the values from following arrays recursively |
| array_reverse() | Returns an array in the reverse order |
| array_search() | Searches an array for a given value and returns the key |
| array_shift() | Removes the first element from an array, and returns the value of the removed element |
| array_slice() | Returns selected parts of an array |
| array_splice() | Removes and replaces specified elements of an array |
| array_sum() | Returns the sum of the values in an array |
| array_udiff() | Compare arrays, and returns the differences (compare values only, using a user-defined key comparison function) |

| | |
|---|---|
| array_udiff_assoc() | Compare arrays, and returns the differences (compare keys and values, using a built-in function to compare the keys and a user-defined function to compare the values) |
| array_udiff_uassoc() | Compare arrays, and returns the differences (compare keys and values, using two user-defined key comparison functions) |
| array_uintersect() | Compare arrays, and returns the matches (compare values only, using a user-defined key comparison function) |
| array_uintersect_assoc() | Compare arrays, and returns the matches (compare keys and values, using a built-in function to compare the keys and a user-defined function to compare the values) |
| array_uintersect_uassoc() | Compare arrays, and returns the matches (compare keys and values, using two user-defined key comparison functions) |
| array_unique() | Removes duplicate values from an array |

| | |
|---|---|
| array_unshift() | Adds one or more elements to the beginning of an array |
| array_values() | Returns all the values of an array |
| array_walk() | Applies a user function to every member of an array |
| array_walk_recursive() | Applies a user function recursively to every member of an array |
| arsort() | Sorts an associative array in descending order, according to the value |
| asort() | Sorts an associative array in ascending order, according to the value |
| compact() | Create array containing variables and their values |
| count() | Returns the number of elements in an array |
| current() | Returns the current element in an array |
| each() | Returns the current key and value pair from an array |
| end() | Sets the internal pointer of an array to its last element |
| extract() | Imports variables into the current symbol table from an array |
| in_array() | Checks if a specified value exists in an array |
| key() | Fetches a key from an array |

| | |
|---|---|
| krsort() | Sorts an associative array in descending order, according to the key |
| ksort() | Sorts an associative array in ascending order, according to the key |
| list() | Assigns variables as if they were an array |
| natcasesort() | Sorts an array using a case insensitive "natural order" algorithm |
| natsort() | Sorts an array using a "natural order" algorithm |
| next() | Advance the internal array pointer of an array |
| pos() | Alias of current() |
| prev() | Rewinds the internal array pointer |
| range() | Creates an array containing a range of elements |
| reset() | Sets the internal pointer of an array to its first element |
| rsort() | Sorts an indexed array in descending order |
| shuffle() | Shuffles an array |
| sizeof() | Alias of count() |
| sort() | Sorts an indexed array in ascending order |

| | |
|---|---|
| uasort() | Sorts an array by values using a user-defined comparison function |
| uksort() | Sorts an array by keys using a user-defined comparison function |
| usort() | Sorts an array using a user-defined comparison function |

## Strings

→ Strings are sequence of characters.

→ Example:

```
$MyStr = “SDMCET”;
echo ‘$MyStr ’;
echo “$MyStr ”;
```

→ escape-sequence characters:

\n is replaced by the newline character
\r is replaced by the carriage-return character
\t is replaced by the tab character
\$ is replaced by the dollar sign itself ($)
\" is replaced by a single double-quote (")
\\ is replaced by a single backslash (\)

→ String Concatenation: dot(.) is used to concatenate two strings.

→ strlen(), strpos(), strcmp(), strrev()….. etc

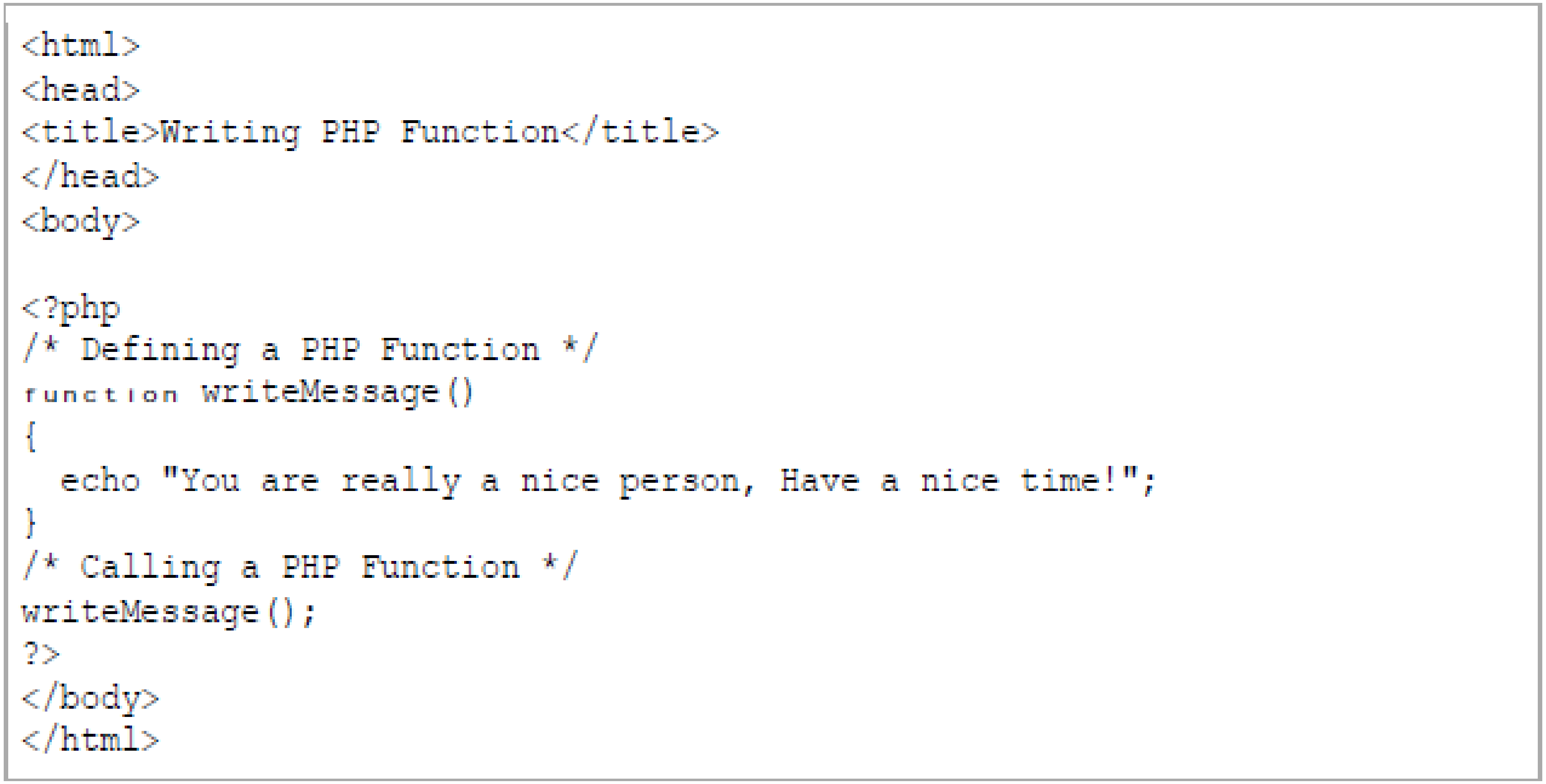
## PHP Functions

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.

There are two parts which should be clear to you:

- 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.



```
<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>
```

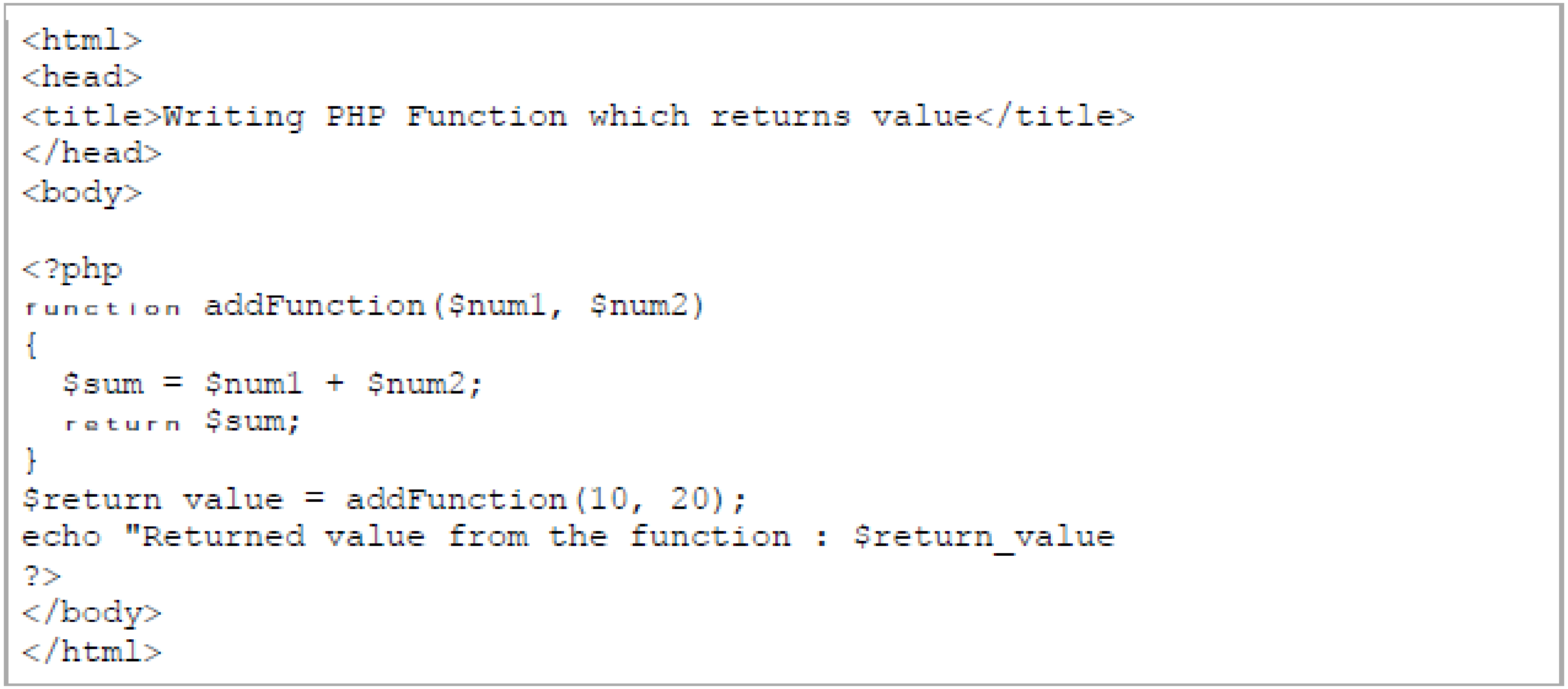
## PHP Functions with Parameters:

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>
```

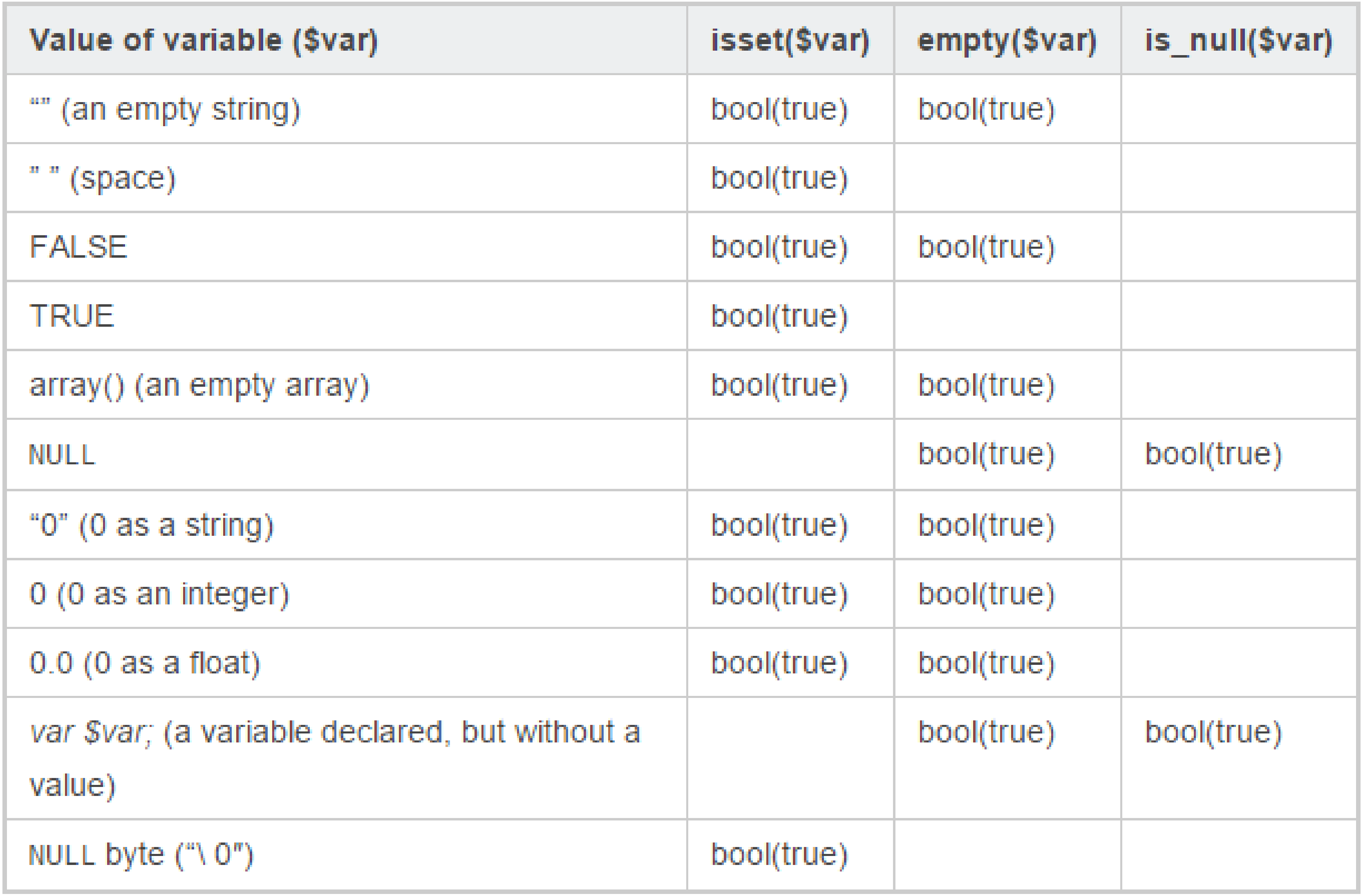
## PHP Functions returning value:

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.



```
<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>
```

# Functions used during Form data collections



| Value of variable ($var) | isset($var) | empty($var) | is_null($var) |
|---|---|---|---|
| "" (an empty string) | bool(true) | bool(true) | |
| " " (space) | bool(true) | | |
| FALSE | bool(true) | bool(true) | |
| TRUE | bool(true) | | |
| array() (an empty array) | bool(true) | bool(true) | |
| NULL | | bool(true) | bool(true) |
| "0" (0 as a string) | bool(true) | bool(true) | |
| 0 (0 as an integer) | bool(true) | bool(true) | |
| 0.0 (0 as a float) | bool(true) | bool(true) | |
| *var $var;* (a variable declared, but without a value) | | bool(true) | bool(true) |
| NULL byte ("\ 0") | bool(true) | | |

## Collection of Form Data

## PHP Global Variables - Superglobals

→Predefined variables in PHP are "superglobals",
→Always accessible, regardless of scope
→Accessed from any function, class or file without having to do anything special.

1. $GLOBALS
2. $_SERVER
3. $_REQUEST
4. $_POST
5. $_GET
6. $_FILES
7. $_ENV
8. $_COOKIE
9. $_SESSION

## PHP Global Variables

| Variable name | Definition |
|---|---|
| $_POST | Used to collect form data after submitting an HTML form with method="post" |
| $_GET | Used to collect form data after submitting an HTML form with method="get".<br>$_GET can also collect data sent in the URL. |
| $_REQUEST | Used to collect data after submitting an HTML form. |
| $_SERVER | PHP super global variable which holds information about headers, paths, and script locations. |
| $_SERVER['HTTPS'] | Is the script queried through a secure HTTP protocol |
| $_SERVER['SERVER_PORT'] | Returns the port on the server machine being used by the web server for communication (such as 80) |

## Examples

1. Design a web application to read two strings from client application, send data to a PHP script to concatenate two strings & print the result.

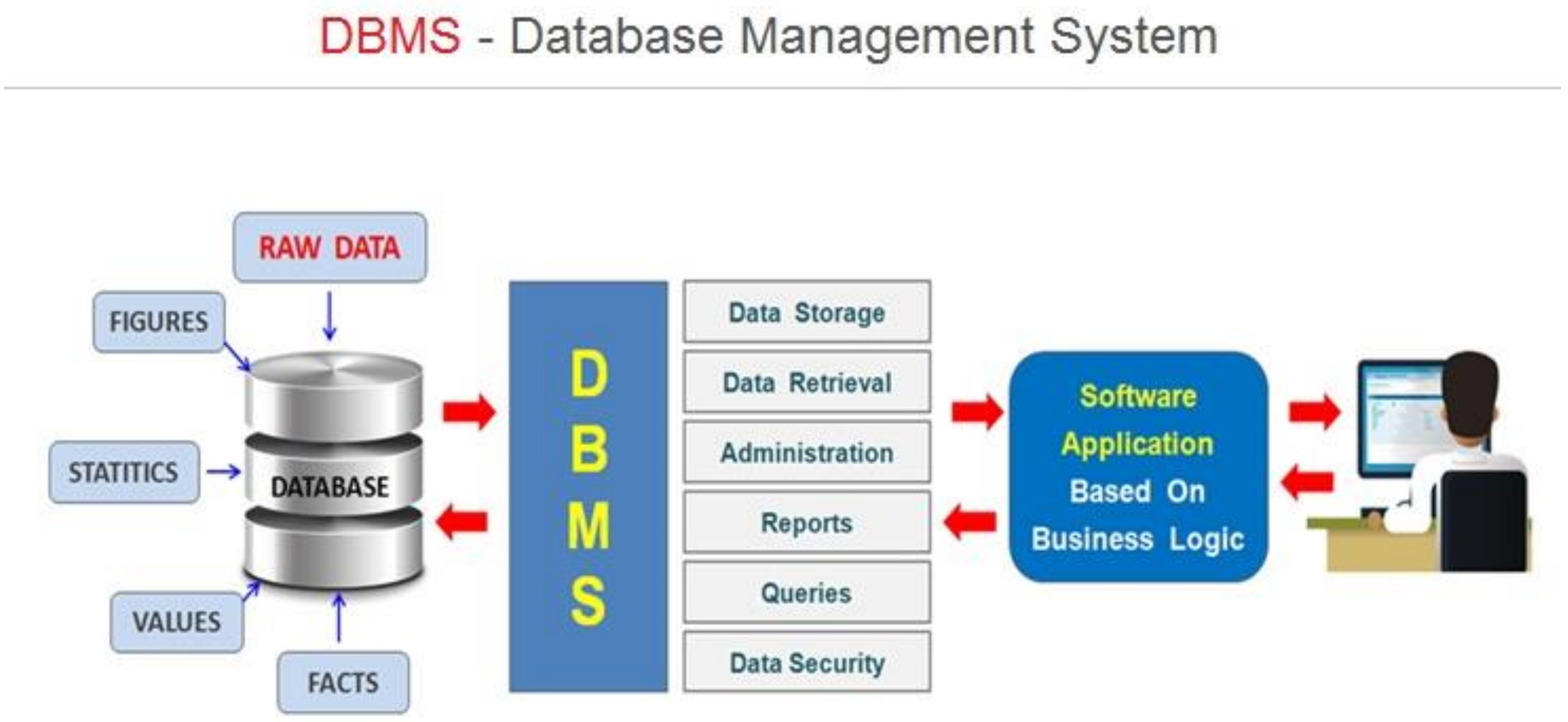
(Condition: Client application must be created in a different location i.e. not in the XAMP/WWW folder)

# PHP with Database (MySQL)

Part-II

## What is Database?

- A database is an organized collection of data that is stored and accessed electronically. It is designed to efficiently store, retrieve, and manage large amounts of structured data, ensuring data integrity and minimizing redundancy.



## What is Database?

- In computing, a database is an organized collection of data or a type of data store based on the use of a database management system (DBMS), the software that interacts with end users, applications, and the database itself to capture and analyze the data.

# SQL queries for CRUD operations

**Shema:**
**id** (integer, primary key, auto-increment)
**first_name** (varchar)
**last_name** (varchar)
**email** (varchar)
**salary** (decimal)

1. **INSERT INTO employees** (first_name, last_name, email, salary) **VALUES** ('John', 'Doe', 'john.doe@example.com', 60000);

**C**

2. **SELECT * FROM** employees;

**R**

3. **UPDATE** employees **SET** salary = 65000 **WHERE** id = 1;

**U**

4. **DELETE FROM** employees **WHERE** id = 1;

**D**

**I.** **DELETE FROM** employees;
**II.** **DROP TABLE** employees;
**III.** **DROP DATABASE** EMP_INFO

# MySQL vs. MySQLi in PHP

| MySQL | MySQLi |
|---|---|
| MySQL extension added in PHP version 2.0. and deprecated as of PHP 5.5.0. | MySQLi extension added in PHP 5.5 and will work on MySQL 4.1.3 or above. |
| Does not support prepared statements. | MySQLi supports prepared statements. |
| MySQL provides the procedural interface. | MySQLi provides both procedural and object-oriented interface. |
| MySQL extension does not support stored procedure. | MySQLi supports store procedure. |
| MySQL extension lags in security and other special features, comparatively. | MySQLi extension is with enhanced security and improved debugging. |
| Transactions are handled by SQL queries only. | MySQLi supports transactions through API. |
| Extension directory: ext/mysql. | Extension directory: ext/mysqli. |

## MySQLi Advantages

- MySQLi function mysqli_query() allows to enforce error prone queries and prevents bugs like SQL injection.
- Using MySQLi data fetch, we can get buffered or unbuffered based on server resource size.
- MySQLi API allows executing multiple queries with single expression using multi_query() function.

## MySQLi

**Example:-**

```
$mysqli = new mysqli("localhost",
"username", "password", "database");

if ($mysqli->connect_error) {
   die("Connection failed: " . $mysqli->
connect_error);
}

$result = $mysqli->query("SELECT * FROM
users");

while ($row = $result->fetch_assoc()) {
   echo "User: " . $row['username'] .
"<br>";
}

$mysqli->close();
```

## MySQL

**Example:-**

```
$mysqli = mysqli_connect("localhost",
"username", "password", "database");

if (!$mysqli) {
   die("Connection failed: " .
mysqli_connect_error());
}

$result = mysqli_query($mysqli, "SELECT *
FROM users");

while ($row = mysqli_fetch_assoc($result))
{
   echo "User: " . $row['username'] .
"<br>";
}
mysqli_close($mysqli);
```

## PHP functions to connect to Database

mysql_affected_rows — Get number of affected rows in previous MySQL operation
mysql_client_encoding — Returns the name of the character set
mysql_close — Close MySQL connection
mysql_connect — Open a connection to a MySQL Server
mysql_create_db — Create a MySQL database
mysql_data_seek — Move internal result pointer
mysql_db_name — Retrieves database name from the call to mysql_list_dbs
mysql_db_query — Selects a database and executes a query on it
mysql_drop_db — Drop (delete) a MySQL database
mysql_errno — Returns the numerical value of the error message from previous MySQL operation
mysql_error — Returns the text of the error message from previous MySQL operation
mysql_escape_string — Escapes a string for use in a mysql_query
mysql_fetch_array — Fetch a result row as an associative array, a numeric array, or both
mysql_fetch_assoc — Fetch a result row as an associative array
mysql_fetch_field — Get column information from a result and return as an object
mysql_fetch_lengths — Get the length of each output in a result
mysql_fetch_object — Fetch a result row as an object
mysql_fetch_row — Get a result row as an enumerated array
mysql_field_flags — Get the flags associated with the specified field in a result
mysql_field_len — Returns the length of the specified field
mysql_field_name — Get the name of the specified field in a result
mysql_field_seek — Set result pointer to a specified field offset

mysql_field_table — Get name of the table the specified field is in
mysql_field_type — Get the type of the specified field in a result
mysql_free_result — Free result memory
mysql_get_client_info — Get MySQL client info
mysql_get_host_info — Get MySQL host info
mysql_get_proto_info — Get MySQL protocol info
mysql_get_server_info — Get MySQL server info
mysql_info — Get information about the most recent query
mysql_insert_id — Get the ID generated in the last query
mysql_list_dbs — List databases available on a MySQL server
mysql_list_fields — List MySQL table fields
mysql_list_processes — List MySQL processes
mysql_list_tables — List tables in a MySQL database
mysql_num_fields — Get number of fields in result
mysql_num_rows — Get number of rows in result
mysql_pconnect — Open a persistent connection to a MySQL server
mysql_ping — Ping a server connection or reconnect if there is no connection
mysql_query — Send a MySQL query
mysql_select_db — Select a MySQL database
mysql_tablename — Get table name of field
mysql_thread_id — Return the current thread ID
mysql_unbuffered_query — Send an SQL query to MySQL without fetching and buffering the result rows.

| Function | Description |
|---|---|
| mysqli_affected_rows() | Returns the number of affected rows in the previous MySQL operation |
| mysqli_close() | Closes a previously opened database connection |
| mysqli_connect() | Opens a new connection to the MySQL server |
| mysqli_errno() | Returns the last error code for the most recent function call |
| mysqli_error() | Returns the last error description for the most recent function call |
| mysqli_fetch_all() | Fetches all result rows as an associative array, a numeric array, or both |
| mysqli_fetch_array() | Fetches a result row as an associative, a numeric array, or both |
| mysqli_fetch_assoc() | Fetches a result row as an associative array |
| mysqli_fetch_row() | Fetches one row from a result-set and returns it as an enumerated array |
| mysqli_free_result() | Frees the memory associated with a result |
| mysqli_num_rows() | Returns the number of rows in a result set |
| mysqli_query() | Performs a query against the database |
| mysqli_real_escape_string() | Escapes special characters in a string for use in an SQL statement |
| mysqli_select_db() | Changes the default database for the connection |

# Steps to Connect to MySQL database

Step-1: Establishing Connection with Server.

```
$connection=mysql_connect("localhost", "root", "");
```

Step-2: Selecting Database

```
$db = mysql_select_db("MyDatabase", $connection);
```

Step-3: Preparing SQL query

```
$sql = insert into table_name(col1,col2)
 values(val1,val2);
```

Step-4: Executing SQL query

```
$query = mysql_query($sql,$connection)
```

# Functions used in querying data

→ mysql_connect — Open a connection to a MySQL Server

→ Example:

→ $link = mysql_connect('localhost', 'mysql_user', 'mysql_password');

→ mysql_select_db — Select a MySQL database

```
// make foo the current db
$db_selected = mysql_select_db('foo', $link);
if (!$db_selected) {
    die ('Can\'t use foo : ' . mysql_error());
}
```

## Contd…

→mysql_query — Send a MySQL query

```
$result = mysql_query('SELECT * WHERE 1=1');
if (!$result) {
    die('Invalid query: ' . mysql_error());
}
```

→ mysql_fetch_array — Fetch a result row as an associative array, a numeric array, or both

```
while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
    printf("ID: %s  Name: %s", $row[0], $row[1]);
}
```

## Contd…

→ die – Equivalent to *exit*

→die doesn't prevent destructors from being run, so the script doesn't exit immediately, it still goes through cleanup routines.

→ mysql_error – function only returns the error text from the most recently executed MySQL function (not including mysql_error() and mysql_errno())

## mysqli_connect

```php
<?php
$link = mysqli_connect("127.0.0.1", "my_user", "my_password", "my_db");

if (!$link) {
    echo "Error: Unable to connect to MySQL." . PHP_EOL;
    echo "Debugging errno: " . mysqli_connect_errno() . PHP_EOL;
    echo "Debugging error: " . mysqli_connect_error() . PHP_EOL;
    exit;
}

echo "Success: A proper connection to MySQL was made! The my_db database is great." . PHP_EOL;
echo "Host information: " . mysqli_get_host_info($link) . PHP_EOL;

mysqli_close($link);
?>
```

```php
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");

/* check connection */
if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
}

/* Create table doesn't return a resultset */
if (mysqli_query($link, "CREATE TEMPORARY TABLE myCity LIKE City") === TRUE) {
    printf("Table myCity successfully created.\n");
}

/* Select queries return a resultset */
if ($result = mysqli_query($link, "SELECT Name FROM City LIMIT 10")) {
    printf("Select returned %d rows.\n", mysqli_num_rows($result));

    /* free result set */
    mysqli_free_result($result);
}
```

```
/* If we have to retrieve large amount of data we use MYSQLI_USE_RESULT */
if ($result = mysqli_query($link, "SELECT * FROM City", MYSQLI_USE_RESULT)) {

    /* Note, that we can't execute any functions which interact with the
       server until result set was closed. All calls will return an
       'out of sync' error */
    if (!mysqli_query($link, "SET @a:='this will not work'")) {
        printf("Error: %s\n", mysqli_error($link));
    }
    mysqli_free_result($result);
}

mysqli_close($link);
?>
```

## mysqli_query() – return type

→mysqli_query()

- ❑ On successful of these queries INSERT, UPDATE, and DELETE, mysqli_query() will return TRUE.

- ❑ If the query fails, mysqli_query() will return FALSE.

## Web application to keep track of Student Information

Example

# JSON

→ It is a textual way to represent objects by using two structures: collections of name-value pairs and arrays of values.

→JSON is a way to represent JavaScript objects as strings.

→Objects are unordered sets of property–value pairs.

→The property–value pairs in an object are separated by commas.

→ It is an alternative to the XML for returning data from the server in response to an Ajax request.

→Why JSON?

  → Reason to use JSON instead of XML is to eliminate the complexity of parsing.

## PHP with JSON

→A common use of JSON is to read data from a web server and display the data on a web page.

→Objective:

- → Encoding and Decoding of JSON objects using PHP.
- →json_decode- Decodes a JSON string
- → json_encode - Returns the JSON representation of a value.
- → It uses an associative array

## PHP with JSON

→JSON Encode

→In PHP, json_encode() is used to convert PHP supported data type into JSON formatted string to be returned as a result of JSON encode operation.

→JSON Decoding

→This is the reverse operation of JSON encode, obviously used to convert JSON encoded data into its original PHP data type from where it is encoded initially.

## Contd…

Example: json_encode

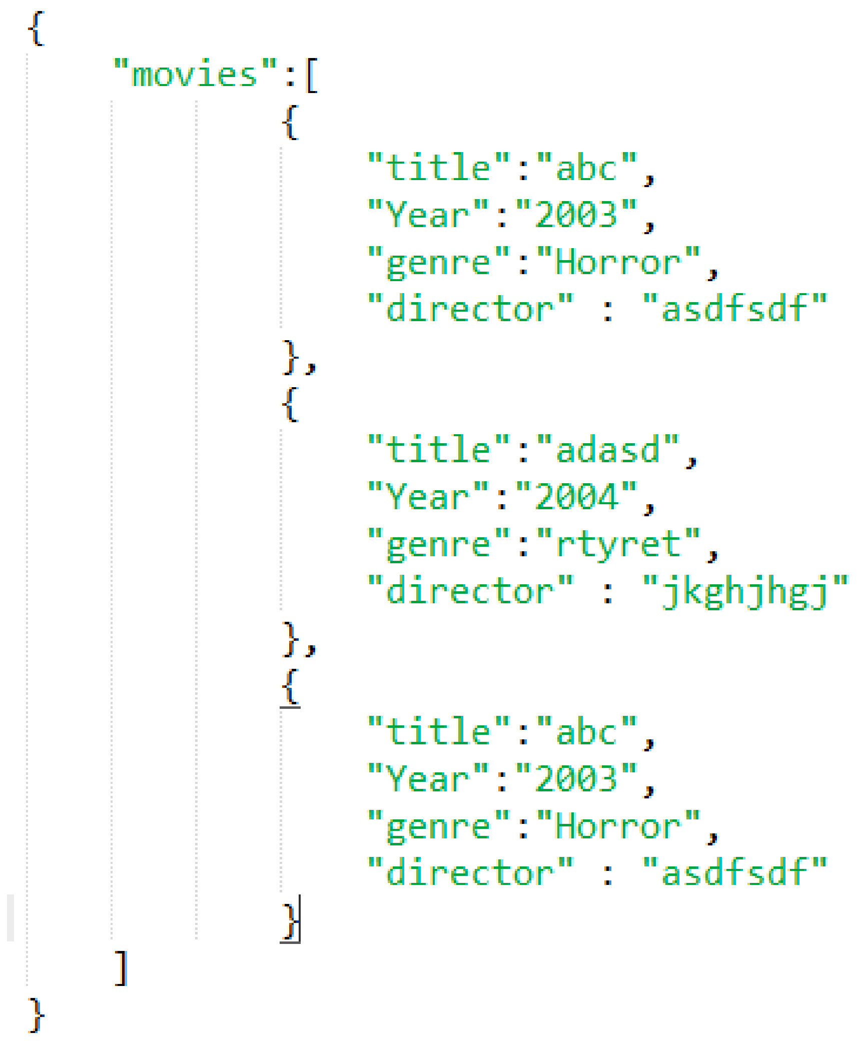
```
<?php
    $arrayObj = array("College"=>"SDM", "Code" =>"2sd","location“
                                        =>"Dharwad");
       echo json_encode($arrayObj);
?>
```

Json_decode:

```
<?php
   $j_obj = '{"college":"sdm" , "Code" : "2sd", "place" : "Dharwad"}';

   var_dump(json_decode($j_obj));
?>
```

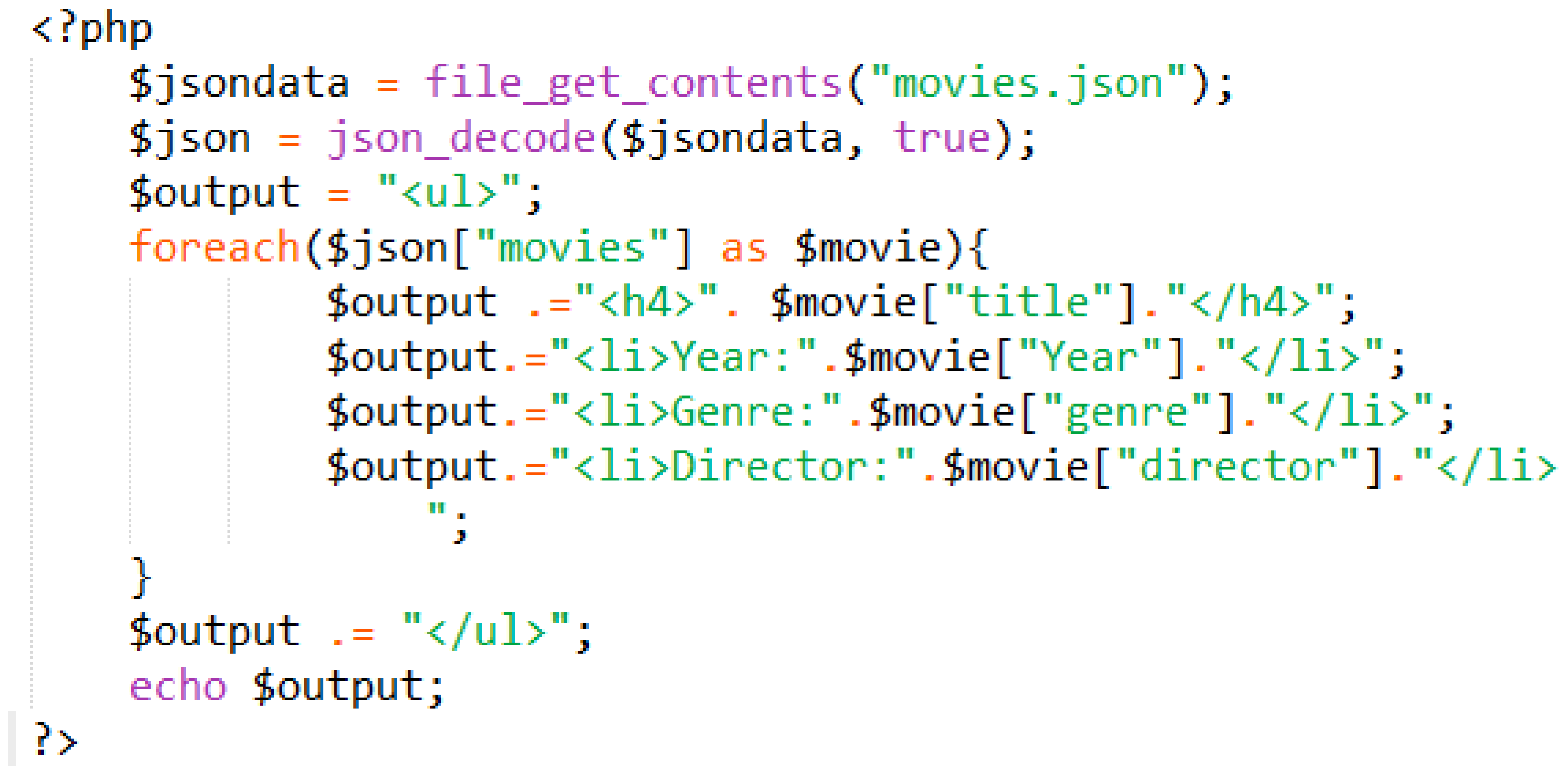
The var_dump() function is used to display structured information (type and value) about one or more variables.

## Example



```
{
    "movies":[
        {
            "title":"abc",
            "Year":"2003",
            "genre":"Horror",
            "director" : "asdfsdf"
        },
        {
            "title":"adasd",
            "Year":"2004",
            "genre":"rtyret",
            "director" : "jkghjhgj"
        },
        {
            "title":"abc",
            "Year":"2003",
            "genre":"Horror",
            "director" : "asdfsdf"
        }
    ]
}
```

## Contd…



```php
<?php
    $jsondata = file_get_contents("movies.json");
    $json = json_decode($jsondata, true);
    $output = "<ul>";
    foreach($json["movies"] as $movie){
            $output .="<h4>". $movie["title"]."</h4>";
            $output.="<li>Year:".$movie["Year"]."</li>";
            $output.="<li>Genre:".$movie["genre"]."</li>";
            $output.="<li>Director:".$movie["director"]."</li>
                ";
    }
    $output .= "</ul>";
    echo $output;
?>
```