4.0 Introduction to PHP

PHP *stands for* ***PHP Hypertext Processor*** *— **A recursive definition!**.*

PHP was originally created by Rasmus Lerdorf in 1995. The main implementation of PHP is now produced by The PHP Group and serves as the formal reference to the PHP language. PHP originally stood for "***Personal Home Page***", it is now said to stand for **"*PHP: Hypertext Preprocessor*"**, *a recursive acronym.*

PHP is a server-side scripting language. This means that the script is run on your web server, not on the user's browser, so you do not need to worry about compatibility issues. PHP is relatively new (compared to languages such as Perl (CGI) and Java) but is quickly becoming one of the most popular scripting languages on the internet. PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, GenericODBC, etc.) PHP is open source software PHP is free to download and use.

PHP file can contain text, HTML tags and scripts. PHP files are returned to the browser as plain HTML.PHP files have a file extension of ".php", ".php3", or ".phtml". PHP enables you to build large, complex, and dynamic websites.

PHP can also increase your productivity enormously, both in development time and maintenance time.

Using PHP, you can build websites that do things such as:
- Query a database
- Allow users to upload files
- Create/read files on the server (for example, the files that your users upload)
- Have a "member's area" (i.e. via a login page)
- Have a shopping cart
- Present a customized experience (for example, based on users' browsing history)
- Much, much more

The ***LAMP*** architecture has become popular in the Web industry as a way of deploying Web applications. PHP is commonly used as the P in this bundle alongside Linux, Apache and MySQL, although the P may also refer to Python or Perl or some mix of the three. Similar packages are also available for Windows and OS X, then called **WAMP**, **WIMP** and **MAMP**, with the first letter standing for the respective operating system. **XAMPP** architecture for any operating system.

Characteristics of PHP:

The main characteristics of PHP are:
- ❖ PHP is web-specific and open source
- ❖ Scripts are embedded into static HTML files
- ❖ Fast execution of scripts
- ❖ Fast access to the database tier of applications
- ❖ Supported by most web servers and operating systems
- ❖ Supports many standard network protocols libraries available for IMAP, NNTP, SMTP,POP3
- ❖ Supports many database management systems libraries available for UNIX DBM, MySQL, Oracle
- ❖ Dynamic Output any text, HTML XHTML and any other XML file
- ❖ Also Dynamic Output images, PDF files and even Flash movies
- ❖ Text processing features, from the POSIX Extended or Perl regular expressions to parsing XML documents
- ❖ A fully featured programming language suitable for complex systems development

Three main uses of PHP
1. ***Server-side scripting*** -– This is the most traditional and main target field for PHP. You need three things to make this work:
   - ❖ The PHP parser.

❖ A web server
❖ A web browser

2. ***Command line scripting*** -- You can make a PHP script to run without any server or browser. You only need the PHP parser to use it this way. These scripts can also be used for simple text processing tasks

3. ***Writing client-side GUI applications*** -- PHP is probably not the very best language to write windowing applications, but PHP-GTK (PHP Graphics Tool Kit) can be used to write such programs.

Advantages of PHP
❖ It is included inside HTML pages.
This means that:
─ All your work can be done in the same directory, and
─ A single file can contain HTML and PHP code
─ Much easier to edit/maintain web pages.
─ This greatly improves tasks like
★ Dynamic page processing.
★ Checking and doing simple HTML Form based tasks.
★ Database connectivity

4.1 How to create and run a PHP Script
Step 1: Open any simple Text editor like notepad, gedit etc.,
Step 2: Type the PHP Code and save it with ".php" extension.
Step 3: Place file in the same directory as HTML pages (*ex: htdocs*)
Step 4: Start the Web server if not
Step 5: Run files by accessing through Web browser– **Must be run via web server URL**

Step1:
Goto Start → Accessories→Run then type Notepad.
Step2:
A Simple PHP Script Example
Here is first complete PHP script which is embedded in HTML: We create a level one header with the PHP output text. This file is called *hello.php*

```
<html>
<head>
        <title>Hello world</title>
</head>
<body>
<h1>
   <?php echo("Welcome to PHP World....!"); ?>
 </h1>
</body>
<html>
```

Step 3:
Open the "*htdocs*" folder under *C:/XAMPPP/htdocs/appl-root/* and place your *".php"* file
Step 4:
Goto Start Menu → XAMPP → XAMPP Control panel, then start *"Apache"* Server
Step5
Open any Web Browser like IE, Chrome or Firefox etc., and type following URL:
http://localhost/appl_root/hello.php

4.2 Data Types
PHP provides eight types of values, or data types. They are divided into three categories:
1. Scalar (single-value) types: integers, floating-point numbers, strings, and Booleans
2. Compound (collection) types: arrays and objects.
3. Special types: resource and NULL.

2

4.2.1 Scalar Types

Integers

Integers are whole numbers, such as 1, 12, and 256. They are the simplest type. They correspond to simple whole numbers, both positive and negative. The range of acceptable values typically extends from −2,147,483,648 to +2,147,483,647. Integers can be assigned to variables, or they can be used in expressions, like so:

$int_var = 12345;
$another_int = -12345 + 12345;

Integer literals can be written in decimal, octal, hexadecimal, or binary. Examples of **decimal** integers include the following:

1998   −641   +33

Octal numbers consist of a leading 0 and a sequence of digits from 0 to 7. Here are some example octal values and their equivalent decimal values:

0755   // decimal 493
+010   // decimal 8

Hexadecimal values begin with 0x, followed by a sequence of digits (0–9) or letters (A–F). The letters can be upper- or lowercase but are usually written in capitals. Some example hexadecimal values and their equivalent decimal values:

0xFF   // decimal 255
0x10   // decimal 16
-0xDAD1   // decimal −56017

Binary numbers begin with 0b, followed by a sequence of digits (0 and 1). Some example binary values and their equivalent decimal values:

0b01100000   // decimal 1
0b00000010   // decimal 2
-0b10   // decimal −2

If you try to store a variable that is too large to be stored as an integer or is not a whole number, it will automatically be turned into a floating-point number.

Floating-Point Numbers

Floating-point numbers (often referred to as real numbers) represent numeric values with decimal digits. Usually, this allows numbers between 1.7E−308 and 1.7E+308 with 15 digits of accuracy. PHP recognizes floating-point numbers in two different formats.

First is normal format:

3.14   0.017   -7.1

Second is scientific notation:

0.314E1   // 0.314*10^1, or 3.14
1 7.0E-3   // 17.0*10^(-3), or 0.017

Strings

A string is a sequence of characters of arbitrary length. String literals are delimited by either single or double quotes:

'big dog'   *"fat hog"*

Variables are expanded (interpolated) within double quotes, while within single quotes they are not:

$name = "Vijay";
echo "Hi, $name\n";   // Hi, Vijay
echo 'Hi, $name';   // Hi, $name

Double quotes also support a variety of string escapes, listed in below Table

Escape sequence	Character represented
\"	Double quotes
\n	Newline
\r	Carriage return
\t	Tab
\\	Backslash
\$	Dollar sign
\{	Left brace

\}	Right brace
\[	Left bracket
\]	Right bracket

Booleans

A Boolean value represents a "truth value"—it says whether something is true or false. PHP defines some values as true and others as false. In PHP, the following values all evaluate to false:

- ❖ The keyword false
- ❖ The integer 0
- ❖ The floating-point value 0.0
- ❖ The empty string ("") and the string "0"
- ❖ An array with zero elements
- ❖ An object with no values or functions
- ❖ The NULL value
- ❖ A value that is not false is true, including all resource values.
- ❖ PHP also provides true and false keywords for clarity

Examples:

```
$x = 5;          // $x has a true value
$x = true;       // clearer way to write it
$y = "";         // $y has a false value
$y = false;      // clearer way to write it
```

4.2.2 Compound Types
Arrays

An array is collection of homogeneous elements grouped under logical name. An array holds a group of values, which you can identify by position (a number, with zero being the first position) or some   identifying   name   (a   string),   called   an   associative   index. Example:

```
$person[0] = "Edison";
$person[1] = "Wankel";
$person[2] = "Crapper";
$creator['Light bulb'] = "Edison";
$creator['Rotary Engine'] = "Wankel";
$creator['Toilet'] = "Crapper";
```

The array() construct can also be used to creates an array. For example:

```
        $person = array("Edison", "Wankel", "Crapper");
        $creator = array('Light bulb' => "Edison",
                        'Rotary Engine' => "Wankel",
                        'Toilet' => "Crapper");
```

Objects

PHP also supports object-oriented programming (OOP). Objects are the run-time instances of class. Once class is defined, any number of objects can be made from it with the new keyword, and the   object's   properties   and   methods   can   be   accessed   with   the   ->   construct: Example

```
$ed = new Person;
$ed->name('Edison');
echo "Hello, {$ed->name}\n";
Hello, Edison
```

4.2.3 Special Types
NULL

There's only one value of the NULL data type. That value is available through the case-insensitive keyword NULL. The NULL value represents a variable that has no value.

```
$alpha = "beta";
$alpha = null; // variable's value is gone
$alpha = Null; // same
$alpha = NULL; // same
```

Resources

Resources are special types. They are used database connectivity.

4.3 PHP Variables

The main way to store information in the PHP program is by using a variable. The most important points 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, be declared before assignment.
- ❖ A variable may hold a value of any type. There is no compile-time or runtime type checking on variables.
- ❖ 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.
- ❖ There is no size limit for variables.
- ❖ PHP does a good job of automatically converting types from one to another when necessary.

Rules for naming a variable are:

- ❖ Variable names must begin with a letter or underscore character.
- ❖ A variable name can consist of numbers, letters, underscores but you cannot use characters like +, - , %, (,). & , etc

Example

$name
$Age
$_debugging
$MAXIMUM_IMPACT
$day = 60 * 60 * 24;
echo "There are {$day} seconds in a day.\n";   //There are 86400 seconds in a day.

4.3.1 Variable Scope

The *scope* of a variable determines those parts of the program that can access it.

(or)

Scope can be defined as the range of availability a variable has to the program in which it is declared. There are four types of variable scope in PHP:

- ❖ Local
- ❖ Global
- ❖ Static and
- ❖ Function parameters

Local Variables

Variable declared inside a function is local to that function. That is, it is visible only to code in that function (including nested function definitions); it is not accessible outside the function. For example, here's a function that updates a local variable instead of a global variable:

```
function updateCount()
{
        $count=0;                // Local variable
        $count++;
        echo "$count";
}
$count = 10;                // Global variable
updateCount();        //1
echo $count;                //10
```

The *$count* inside the function is local to that function, the function increments its private **$count** variable, which is destroyed when the subroutine ends. The global **$count** remains set at 10.

Global scope

Variables declared outside a function are global. That is, they can be accessed from any part of the program. By default, they are not available inside functions. To allow a function to access a global variable, you can use the global keyword inside the function to declare the variable within the function. Here's how we can rewrite the updateCount() function to allow it to access the global $count variable:

```
function updateCount()
{
        global $count;
        $count++;
}
$count = 10;
updateCount();
echo $count;                        //11
```

Static variables

A static variable retains its value between calls to a function but is visible only within that function. You declare a variable static with the static keyword.
For example:

```
function updateCount()
{
        static $count = 0;
        $count++;
        echo "Static counter is now {$count}\n";
}
$count = 10;
updateCount();                              // Static counter is now 1
updateCount();                              // Static counter is now 2
echo "Global counter is {$counter}\n";   // Global counter is 10
```

Function parameters

A function definition can have named parameters. Function parameters are local, meaning that they are available only inside their functions.

```
function greet($name)
{
        echo "Hello, {$name}\n";
}
greet("Vijay");                    // Hello, Janet
```

In this case, $name is inaccessible from outside greet().

4.3.2 PHP Constants

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

To define a constant you have to use **define()** function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a $.

constant() example:

```
<?php
        define("MINSIZE", 50);
        echo MINSIZE;
?>
```

Only scalar data (boolean, integer, float and string) can be contained in constants.

Differences between constants and variables are:

❖ There is no need to write a dollar sign ($) before a constant, where as in Variable one has to write a dollar sign.
❖ Constants cannot be defined by simple assignment, they may only be defined using the define() function.
❖ Constants may be defined and accessed anywhere without regard to variable scoping rules.
❖ Once the Constants have been set, may not be redefined or undefined.

Valid and invalid constant names:

```
// Valid constant names
        define("ONE", "first thing");
        define("TWO2", "second thing");
        define("THREE_3", "third thing")
// Invalid constant names
        define("2TWO", "second thing");
        define("__THREE__", "third value");
```

4.4 PHP Operators: PHP language supports following type of operators.

1. Arithmetic Operators
2. Comparison Operators
3. Logical (or Relational) Operators
4. Assignment Operators
5. Conditional (or ternary) Operator

Arithmetic Operators:

Arithmetic operators are used to perform arithmetic operations like addition, subtraction, division, etc., The following arithmetic operators supported by PHP language.

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Logical Operators

There are following logical operators supported by PHP language. Assume variable A holds 10 and variable B holds 20 then:

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

Assignment Operators

There are following assignment operators supported by PHP language

Operator	Description	Example

=	Simple assignment operator, Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to the left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Conditional Operator

There is one more operator called conditional operator. This first evaluates an expr1 for a true or false value and then execute one of the two given expressions depending upon the result of the evaluation. The conditional operator has this syntax:

$result = expr1 ? expr2 : expr3;

Operator	Description	Example
?:	Conditional Expression	If expr1 is true ? Then expr2 : Otherwise expr3

4.5 PHP Decision Making

The **if**, **elseif ...else** and **switch** statements are used to take decisions based on the different condition. You can use conditional statements in your code to make your decisions. PHP supports following three decision making statements:

1. **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 false.
2. **elseif statement -** is used with the if...else statement to execute a set of code if one of several condition are true
3. **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.

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

Syntax

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

Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

> *<html>*
>> *<body>*
>>> *<?php*
>>>> *$d=date("D");*
>>>> *if ($d=="Fri")*
>>>>> *echo "Have a nice weekend!";*

8

```
            else
                echo "Have a nice day!";
        ?>
    </body>
</html>
```

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

```
<html>
    <body>
        <?php
            $d=date("D");
            if ($d=="Fri")
            {
                        echo "Hello!<br />";
                        echo "Have a nice weekend!";
                        echo "See you on Monday!";
            }
        ?>
    </body>
</html>
```

The If...else...if Statement

If you want to execute some code if one of several conditions are true use the elseif statement

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;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
    <body>
        <?php
            $d=date("D");
            if ($d=="Fri")
                echo "Have a nice weekend!";
            elseif ($d=="Sun")
                echo "Have a nice Sunday!";
            else
                echo "Have a nice day!";
        ?>
    </body>
</html>
```

The Switch Statement

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

```
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;
      }
```

Example

The switch statement works in an unusual way. First it evaluates given expression then seeks a label to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the labels match then statement will execute any specified default code.

```html
<html>
   <body>
      <?php
            $d=date("D");
            switch ($d)
            {
                case "Mon":
                    echo "Today is Monday";
                    break;
                case "Tue":
                    echo "Today is Tuesday";
                    break;
                case "Wed":
                    echo "Today is Wednesday";
                    break;
                case "Thu":
                    echo "Today is Thursday";
                    break;
                case "Fri":
                    echo "Today is Friday";
                    break;
                case "Sat":
                    echo "Today is Saturday";
                    break;
                case "Sun":
                    echo "Today is Sunday";
                    break;
                default:
                    echo "Wonder which day is this ?";
            }
      ?>
   </body>
</html>
```

4.6 PHP Loops

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

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

10

The for loop statement

The for statement is used when you know the exact number of times to execute a statement or a block of statements.

Syntax

for (initialization; condition; increment/decrement)
{
        Code to be executed;
}

The initializer is used to set the start value for the counter of the number of loop iterations.

Example

The following example makes ten iterations and prints 1 to 10 numbers:

```
<html>
    <body>
        <?php
          for( $i=1; $i<=10; $i++ )
          {
                echo ("$i"."    " );
          }
        ?>
    </body>
</html>
```

This will produce following result:
1 2 3 4 5 6 7 8 9 10

The while loop statement

The while statement will execute a block of code if and as long as the condition is true.

Syntax

w*hile (condition)*
{
        code to be executed;
}

Example

This example prints the reverse of a given number.

```
<html>
 <body>
   <?php
       $n = 1234;
       $t = $n;
       $rev=0;
       while($n>0)
       {
         $rem = $n%10;
         $rev = $rev*10+$rem;
         $n = ($n-$rem)/10;
    }
       echo "Reverse of $t is $rev ";
   ?>
 </body>
</html>                     }
```

This will produce following result:
Reverse of 1234 is 4321

The do...while loop statement

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

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

Example

The following example will check whether the given number is palindrome or not?

```
<html>
   <body>
      <?php
         $n = 858;
         $t = $n;
         $rev=0;
         do
         {
           $rem = $n%10;
           $rev = $rev*10+$rem;
           $n = ($n-$rem)/10;
         }while($n>0);
         if($rev== $t)
             echo "$t IS PALLINRDROME ";
         else
           echo "$t is not a PALLINRDROME ";
      ?>
 </body>
</html>
```

This will produce following result:

858 IS PALLINRDROME

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;
}
```

Example

The following example list out the values of an array.

```
<html>
   <body>
      <?php
         $array = array( 1, 2, 3, 4, 5);
         foreach( $array as $value )
         {
              echo "Value is $value <br />";
         }
      ?>
   </body>
</html>
```

This will produce following result:

Value is 1
Value is 2
Value is 3
Value is 4

Value is 5

The break statement

      The PHP **break** keyword is used to terminate the execution of a loop prematurely. The **break** statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.

Example

      In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>
   <body>
      <?php
         $i = 0;
         while( $i < 10)
         {
               $i++;
               if( $i == 3 )break;
         }
          echo ("Loop stopped at i = $i" );
      ?>
   </body>
</html>
```

This will produce following result: **Loop stopped at i = 3**

The continue statement

      The PHP **continue** keyword is used to halt the current iteration of a loop and place the loop in next iteration. But it does not terminate the loop.

      Just like the break statement the continue statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering continue statement, rest of the loop code is skipped and next pass starts.

Example

      In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```
<html>
   <body>
      <?php
         $array = array( 1, 2, 3, 4, 5);
         foreach( $array as $value )
         {
               if( $value == 3 )continue;
               echo "Value is $value <br/>";
         }
      ?>
   </body>
</html>
```

This will produce following result
Value is 1
Value is 2
Value is 4
Value is 5

4.7 PHP Arrays

      An array is a data structure that stores one or more similar type of values in a single value. There are three different kind of arrays and each array value is accessed using an ID which is called array index.

1. **Numeric array** - An array with a numeric index. Values are stored and accessed in linear fashion
2. **Associative array** - An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
3. **Multidimensional array** - An array containing one or more arrays and values are accessed using multiple indices

4.7.1 Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

Example

Following is the example showing how to create and access numeric arrays. Here we have used **array()** function to create array.

```
<html>
  <body>
      <?php
          $numbers = array(1,2,3,4,5); /* First method to create array. */
          foreach($numbers as $value)
              echo "Value is $value <br />";
          $numbers[0] = "one";  /* Second method to create array. */ $numbers[1] = "two";
          $numbers[2] = "three";
          $numbers[3] = "four";
          $numbers[4] = "five";
          foreach( $numbers as $value )
              echo "Value is $value <br />";
      ?>
  </body>
</html>
```

This will produce following result:

Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five

4.7.2 Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

Example

```
<html>
  <body>
   <?php
          /* First method to associate create array. */
          $salaries = array("raju" => 2000,
                            "anand" => 1000,
                            "kiran" => 500 );

          echo "Salary of raju is ". $salaries['raju'] . "<br/>";
          echo "Salary of anand is ". $salaries['anand']. "<br/>";
          echo "Salary of kiran is ". $salaries['kiran]. "<br/>";
          /* Second method to create array. */
```

```
                $salaries['raju'] = "high";
                $salaries['anand'] = "medium";
                $salaries['kiran'] = "low";
                echo "Salary of raju is ". $salaries['raju'] . "<br/>";
                echo "Salary of anand is ". $salaries['anand']. "<br/>";
                echo "Salary of kiran is ". $salaries['kiran']. "<br/>";
         ?>
      </body>
   </html>
```
This will produce following result:
Salary of raju is 2000
Salary of anand is 1000
Salary of kiran is 500
Salary of raju is high
Salary of anand is medium
Salary of kiran is low

4.7.3 Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed   using   multiple   index.

Example

In this example we create a two dimensional array to store marks of three students in three subjects. This example is an associative array

```
        <html>
         <body>
           <?php
                $marks=array("raju"=>array("physics"=>35,"maths"=>30,"chemistry"=>39),
                        "koti"=>array("physics"=>30,"maths"=>32,"chemistry"=>29),
                        "rani"=>array("physics"=>31,"maths"=>22,"chemistry"=>39));
                /* Accessing multi-dimensional array values */
                echo "Marks for raju in physics : " ;
                echo $marks['raju']['physics'] . "<br />";
                echo "Marks for koti in maths : ";
                echo $marks['koti']['maths'] . "<br />";
                echo "Marks for rani in chemistry : " ;
                echo $marks['rani']['chemistry'] . "<br />";
           ?>
         </body>
        </html>
```
This will produce following result:
Marks for raju in physics : 35
Marks for koti in maths : 32
Marks for rani in chemistry : 39

4.8 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.

   ❖ Creating a PHP Function
   ❖ Calling a PHP Function

Creating PHP Function:

Its very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.
Note that while creating a function its name should start with keyword function and all the PHP code should be put inside { and } braces as shown in the following example below:

```
        <html>
```

```
<head>
        <title>Writing PHP Function</title>
</head>
<body>
        <?php
                /* Defining a PHP Function */
        function welcome()
        {
                echo "You have created a simple PHP function!";
        }
                /* Calling a PHP Function */
        welcome();
        ?>
    </body>
</html>
```

This will display following result:
You have created a simple PHP function!

PHP Functions with Parameters (Passing Arguments by Value)

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters you 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 add($num1, $num2)
                {
                        $sum = $num1 + $num2;
                        echo "Sum of the two numbers is : $sum";
                }
                add(10, 20);
        ?>
    </body>
</html>
```

This will display following result:
Sum of the two numbers is: 30

Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value. Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition. Following example depicts both the cases.

```
<html>
    <head>
        <title>Passing Argument by Reference</title>
    </head>
    <body>
        <?php
            function addFive(&$num)
            {
                $num += 5;
            }
            $orignum = 10;
            addFive( $orignum );
```

*echo "Original Value is $orignum
";*

?>

</body>

</html>

This will display following result:

Original Value is 15

4.9 Forms

One of the best features of PHP is possibility to respond to user queries or data submitted from HTML forms. You can process information gathered by an HTML form and use PHP code to make decisions based on this information to create dynamic web pages. There are two ways to get form data:

1. GET Method
2. POST Method

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? Character.

http://www.test.com/index.html?name1=value1&name2=value2

❖ The GET method is restricted to send upto 1024 characters only.
❖ Never use GET method if you have password or other sensitive information to be sent to the server.
❖ GET can't be used to send binary data, like images or word documents, to the server.
❖ The data sent by GET method can be accessed using QUERY_STRING environment variable.
❖ The PHP provides $_GET associative array to access all the sent information using GET method.

Following example show the usage GET method:"index.php"

```php
<?php
   if(isset($_GET['name']))
   {
       if( $_GET["name"] || $_GET["age"] )
       {
          echo "Welcome ". $_GET['name']. "<br />";
          echo "You are ". $_GET['age']. " years old.";
       }
   }
?>
<html>
  <body>
    <form action="index.php"  method="GET">
        Name: <input type="text" name="name" />
        Age: <input type="text" name="age" />
        <input type="submit" />
    </form>
  </body>
</html>
```

Output:

When you submit form:

The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

❖ The POST method does not have any restriction on data size to be sent.
❖ The POST method can be used to send ASCII as well as binary data.
❖ The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
❖ The PHP provides $_POST associative array to access all the sent information using POST method.

Following example show the usage GET method:"index.php"

```php
<?php
  if(isset($_POST['name']))
  {
   if( $_POST["name"] || $_POST["age"] )
   {
     echo "Welcome ". $_POST['name']. "<br />";
     echo "You are ". $_POST['age']. " years old.";
   }
  }
?>
<html>
  <body>
   <form action="index.php" method="POST">
        Name: <input type="text" name="name" />
        Age: <input type="text" name="age" />
        <input type="submit" />
   </form>
  </body>
</html>
```

Output:

When you submit the form:

4.10 Working with Databases:
   PHP communication with Mysql can be divided into 5 steps like
           1) Connecting to the Database
           2) Selecting to the Database
           3) Query the Database
           4) Fetching the Records or getting the records from Query or Query String.
           5) Close the Connection.

1) Connecting to the Database
We can connect in two ways:
           A) Normal Connection   B) Persistence Connection
           Normal Connection: Will be active for the Single Program, A Normal Connection can be
           Closed by Using mysql_close();
                   mysql_connect('hostname','username','password');
           Persistance Connection: A Persistance Connection is a Permanent Connection Once
           establish it cannot be closed.
                   mysql_pconnect('hostname','username','password');

2) **Selecting to the Database**
           mysql_select_db('dbname',[Optional Conn Handler]);
    Selects the Argumented data as current data...
     Error Handling in Database:
     a)mysql_error(): Returns the last Error Message Occurred with the data base server.
     b)mysql_errno(): Returns the last error code associated with the mysql database Server.

3) **Querying the Database :**
           a) mysql_query(SQL) : Executes the Query with the database server and returns the Query
           Handler.
           b) mysql_affected_rows : Returns the number of rows affected for the last insert, Update or
           delete Query.
           c) mysql_num_rows(Query handler) : Returns the number of records in the resultant for the
           select Query .

4) **Fetching the Records :**
           a) mysql_fetch_row(Query handler) : Returns a single record set from the database server, as
           a numerical array and moves the query handler to next record if record does not exist return
           false.

b) mysql_fetch_assoc(Query handler) : Returns the record Set as an associative array with field names as its index position - if record does not exist return false..

c) mysql_fetch_array(Query handler) : Returns the record set as an array with numerical or associative or both arrays . Returns false if no record found.

d) mysql_fetch_object(Query handler) : Returns the record set as an object with field names as its values .

5) Close the Connection :

a)mysql_close : Close the connection with the data base.

There are 2 other functions in php.Which gives a message. if not able to connect to database then we can read the message as

die('Message'); or die(mysql_errno().' - '.mysql_error()); or exit;

A Sample program on Database connectivity:

Create Student table in MySQL:

```
create table student
    (  sno integer(10),
        sname varchar(25),
        branch varchar(15));
```

conn.php

```php
<html>
<?php
   $host = "127.0.0.1";
   $user = "root";
   $pwd = "";
   $db = "test";
      $db_handle = mysql_connect($host,$user,$pwd);
      $db_found = mysql_select_db($db,$db_handle);
      if(isset($_POST['sno']))
      {
        if($db_found)
        {
          $in='INSERT INTO student VALUES(' . $_POST['sno'] .',"'. $_POST['sname'] . '","' .
$_POST['branch'] . ' ")';
          if(isset($in))
          {
        mysql_query($in);
          $sql = "select * from student order by sno";
          $res = mysql_query($sql); ?>
        <table border ="0" rules="all">
            <tr>
              <th> RNO </th>
                <th> SNAME </th>
                <th> BRANCH </th>
            </tr>
        <?php
            while($row = mysql_fetch_assoc($res))
            { ?>
            <tr>
             <td><?php echo $row['sno']; ?></td>
             <td><?php echo $row['sname']; ?></td>
             <td><?php echo $row['branch']; ?></td>
        </tr>
            <?php } ?>
            </table>
            <?php  mysql_close($db_handle);
            }
```

```
        }
        else
            echo "$db not found";
        }
    ?>
  <body>
      <form name="f1" method="POST" action="conn.php">
      <table border="1">
      <tr>    <th> Data Entry Form</th> </tr>
      <tr>
            <td> ROLL NO:<input type="text" name="sno"/></td>
      </tr>
      <tr>
            <td> SNAME  : <input type="text" name="sname"/></td>
      </tr>
      <tr>
            <td> BRANCH  :<input type="text" name="branch"/></td>
      </tr>
      <tr>
            <td align="center"><input type="Submit" name="Submit1" value="Insert"/></td>
      </tr>
</table>
</form>
</body></html>
```

Output:

When You Press Insert: