PHP Introduction

It is one of the most popular open source, server side scripting language especially meant for developing web applications.

HTML:-To create well formatted web documents in the year 1991 Tim Berners Lee developed HTML.HTML is a language for formatting web pages.

HTML has limited design capabilities. We can’t able to create most appealing and well designed web sites using only HTML.

CSS:-To create most appealing and well designed web sites, in the year 1996 Hakon Wium Lie developed CSS.CSS is a presentational language.

Java script:To add behaviour or interactivity to web elements or to create interactive web pages,in the year 1996 Brendan Each developed Javascript. Javascript is a behaviorl language for web pages.

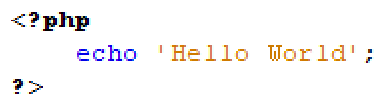
We can’t be able to create real time, dynamic web sites using only HTML,CSS,Javascript.Hence we use PHP.

PHP: To create real time dynamic web sites, in the year 1996 Rasmus Lerdorf developed PHP at Zend technology .PHP is implemented in C and C++.

Initially Rasmus Lerdorf wrote PHP to maintain his personal home page. Later extended it to communicate with databases, file systems and work with web forms and called this implementation as PHP/FI(personal Home Page/Forms Interpreter).Later the name changed to Hypertext Pre-processor.

PHP can be integrated with the number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.

PHP can be embedded within a normal HTML web pages. That means inside your HTML documents you'll have PHP statements like this:



```
<?php
    echo 'Hello World';
?>
```

A PHP file can also contain tags such as HTML and client side scripts such as JavaScript.

Advantages of PHP over Other Languages

→**Easy to learn:** PHP is often considered as the best and preferable choice of scripting language to learn.

→**Open source:** PHP is an open-source project — the language is developed and maintained by a worldwide community of developers who make its source code freely available to download and use. There are no costs associated with using PHP for individual or commercial projects, including future updates.

→**Fast Performance:** Scripts written in PHP usually execute faster than those written in other scripting languages like ASP.NET or JSP.

→**Vast Community:** Since PHP is supported by the worldwide community, finding help or documentation for PHP online is extremely easy.

→**Platform Independent**: PHP are available for WINDOWS, MAC, LINUX & UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.

→**Compatibility**: PHP is compatible with almost all local servers used today like Apache, IIS etc.

→**Embedded**: PHP code can be easily embedded within HTML tags and script.

What is the scope for a PHP developer

PHP is one of the oldest and most popular programming languages well suited for web application. More than 70+ of the websites and other web application are using PHP as their most preferred backend coding language. However, along with PHP, you also have good understanding of PHP based framework and CMS.

A list of PHP framework and CMS are given below:

PHP Web-Framework: -

1.Laravel Framework

2.Zend Framework

3.CakePHP Framework

4.Symfony Framework

PHP CMS (Content Management System): -

1.WordPress 2.Joomla

3.Drupal 4.Magento 5.Open Cart

Versions of php

Version	Release date	Supported until	Notes
1.0	8 June 1995		Officially called "Personal Home Page Tools (PHP Tools)". This is the first use of the name "PHP".
2.0	1 November 1997		Officially called "PHP/FI 2.0". This is the first release that could actually be characterised as PHP, being a standalone language with many features that have endured to the present day.
3.0	6 June 1998	20 October 2000	Development moves from one person to multiple developers. Zeev Suraski and Andi Gutmans rewrite the base for this version.
4.0	22 May 2000	23 June 2001	Added more advanced two-stage parse/execute tag-parsing system called the Zend engine.
4.1	10 December 2001	12 March 2002	Introduced "superglobals" ($_GET, $_POST, $_SESSION, etc.)
4.2	22 April 2002	6 September 2002	Disabled register_globals by default.
4.3	27 December 2002	31 March 2005	Introduced the command-line interface (CLI), to supplement the CGI.
4.4	11 July 2005	7 August 2008	Fixed a memory corruption bug, which required breaking binary compatibility with extensions compiled against PHP version 4.3.x.
5.0	13 July 2004	5 September 2005	Zend Engine II with a new object model.

5.1	24 November 2005	24 August 2006	Performance improvements with introduction of compiler variables in re-engineered PHP Engine.Added PHP Data Objects (PDO) as a consistent interface for accessing databases.
5.2	2 November 2006	6 January 2011	Enabled the filter extension by default. Native JSON support.
5.3	30 June 2009	14 August 2014	Namespace support; late static bindings, jump label (limited goto), closures, PHP archives (phar), garbage collectionfor circular references, improved Windows support, sqlite3, mysqlnd as a replacement for libmysql as underlying library for the extensions that work with MySQL.
5.4	1 March 2012	3 September 2015	Trait support, short array syntax support. Removed items: register_globals, safe_mode, allow_call_time_pass_reference, session_register(), session_unregister() and session_is_registered().
5.5	20 June 2013	10 July 2016	Support for generators, finally blocks for exceptions handling, OpCache (based on Zend Optimizer+) bundled in official distribution.
5.6	28 August 2014	31 Dec 2018	Constant scalar expressions, variadic functions, argument unpacking, new exponentiation operator, extensions of the use statement for functions and constants
6.x	Not released	N/A	Abandoned version of PHP that planned to include native Unicode support.
7.0	3 Dec 2015	3 Dec 2018	Zend Engine 3,bitwise shift consistency across platforms, ??(null coalesce) operator, Unicode codepoint escape syntax,return type declarations, calar type (integer, float, string and boolean) declarations,<=> "spaceship" three-way comparison operator, PHP "errors" with the more modern exceptions,[4 and shorthand syntax for importing multiple items from a namespace.

7.1	1 Dec 2016	1 Dec2019	void return type,class constant visibility modifiers
7.2	30 November 2017	30 November 2020	

PHP - Environment Setup

In order to develop and run PHP Web pages three vital components need to be installed on your computer system.

- **Web Server** − PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server.
- **Database** − PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.
- **PHP Parser** − In order to process PHP script instructions a parser must be installed to generate HTML output that can be sent to the Web Browser.

To install PHP, there are many AMP options available in the market that are given below:

- **WAMP** for Windows
- **LAMP** for Linux
- **MAMP** for Mac
- **SAMP** for Solaris
- **FAMP** for FreeBSD
- **XAMPP** (Cross, Apache, MySQL, PHP, Perl) for Cross Platform: It includes some other components too such as FileZilla, OpenSSL, Webalizer, Mercury Mail etc.

PHP Example

All PHP code goes between php tag. A syntax of PHP tag is given below:

```
<?php
//your code here
?>
```

Ex:

```
<!DOCTYPE>
<html>
<body>
<?php
echo "<h2>Hello First PHP</h2>";
?>
</body>
</html>
```

Output: **Hello First PHP**

PHP Variables

→Variables are used to store data, like string of text, numbers, etc. Variable values can change over the course of a script. Here're some important things to know about variables:

→In PHP, a variable does not need to be declared before adding a value to it. PHP automatically converts the variable to the correct data type, depending on its value.

→After declaring a variable it can be reused throughout the code.

→The assignment operator (=) used to assign value to a variable.

In PHP variable can be declared as: `$var_name = value;`

```
<?php
// Declaring variables
$txt = "Hello World!";
$number = 10;
 // Displaying variables value
echo $txt;  // Output: Hello
World!
echo $number; // Output: 10
?>
```

Naming Conventions for PHP Variables

These are the following rules for naming a PHP variable:

- All variables in PHP start with a $ sign, followed by the name of the variable.
- A variable name must start with a letter or the underscore character _.
- A variable name cannot start with a number.
- A variable name in PHP can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- A variable name cannot contain spaces.

PHP $ and $$ Variables

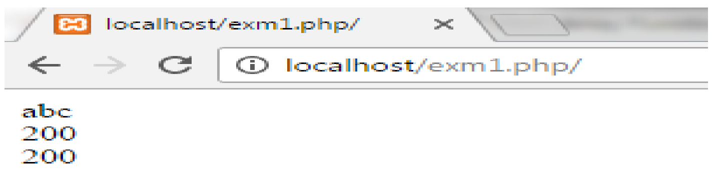
The **$var** (single dollar) is a normal variable with the name var that stores any value like string, integer, float, etc.

The **$$var** (double dollar) is a reference variable that stores the value of the $variable inside it.

Example 1

```
<?php
$x = "abc";
$$x = 200;
echo $x."<br/>";
echo $$x."<br/>";
echo $abc;
?>
```

Output:

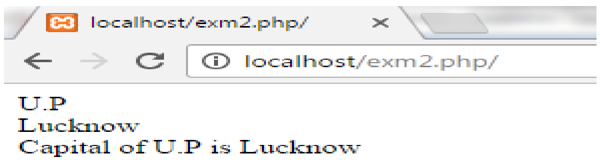


```
abc
200
200
```

Example2

```
<?php
 $x="U.P";
$$x="Lucknow";
echo $x. "<br>";
echo $$x. "<br>";
echo "Capital of $x is " . $$x;
?>
```

Output:

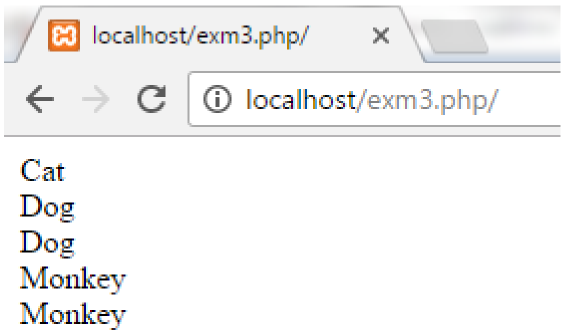


```
U.P
Lucknow
Capital of U.P is Lucknow
```

Example3

```
<?php
$name="Cat";
${$name}="Dog";
${${$name}}="Monkey";
echo $name. "<br>";
echo ${$name}. "<br>";
echo $Cat. "<br>";
echo ${${$name}}. "<br>";
echo $Dog. "<br>";
?>
```

Output:



```
Cat
Dog
Dog
Monkey
Monkey
```

PHP Constants

PHP constants are name or identifier that can't be changed during the execution of the script. PHP constants can be defined by 2 ways:

1. Using define() function
2. Using const keyword

PHP constants follow the same PHP variable rules. For example, it can be started with letter or underscore only.Conventionally, PHP constants should be defined in uppercase letters.

→define(name, value, case-insensitive)

1. name: specifies the constant name
2. value: specifies the constant value

3. case-insensitive: Default value is false. It means it is case sensitive by default.

Ex: 1-

```
<?php
define("MESSAGE", "Hello JavaTpoint PHP");
echo MESSAGE;
?>
```

```
Output:
Hello JavaTpoint PHP
```

Ex: 2-

```
<?php
define("MESSAGE","Hello JavaTpoint PHP",true);//not case sensitive
echo MESSAGE;
echo message;
?>
```

```
Output:
Hello JavaTpoint PHPHello JavaTpoint PHP
```

Ex:-3-

```
<?php
define("MESSAGE","Hello JavaTpoint PHP",false);//case sensitive
echo MESSAGE;
echo message;
?>
```

```
Output:
Hello JavaTpoint PHP
Notice: Use of undefined constant message - assumed 'message'
in C:\wamp\www\vconstant3.php on line 4
message
```

→PHP constant: const keyword

The const keyword defines constants at compile time. It is a language construct not a function.It is bit faster than define().It is always case sensitive.

```
<?php
const MESSAGE="Hello const by JavaTpoint PHP";
echo MESSAGE;
?>
```

```
Output:  Hello const by JavaTpoint PHP
```

Difference between echo and print in PHP

echo

→echo is a statement i.e used to display the output. it can be used with parentheses echo or without parentheses echo.

→echo can pass multiple string separated as (,)

→echo doesn’t return any value

→echo is faster than print

Example-1

```
<?php
$name="Ravi";
echo $name;
//or
echo ($name);
 ?>
```

Output

Ravi

Example-2

```
<?php
 $name = "Ravi ";
 $profile = "PHP Developer";
 $age = 25;
 echo $name , $profile , $age, " years old";
 ?>
```

Output

Ravi PHP Developer 25 years old

Example-3

```
<?php
 $name = "Ravi ";
 $ret =  echo $name;
 ?>
```

Output

```
Parse error: syntax error, unexpected T_ECHO
```

Print

1. Print is also a statement i.e used to display the output. it can be used with parentheses print( ) or without parentheses print.
2. using print can doesn’t pass multiple argument
3. print always return 1
4. it is slower than echo

Example- 1:

```
<?php
 $name="Ravi";
 print $name;
 //or
 print ($name);
 ?>
```

Output : Ravi

Example-2:-

```
<?php
$name = "Ravi ";
$profile = "PHP Developer"; $age = 25;
print $name , $profile , $age, " years old";
?>
```

Output: Parse error: syntax error

Example-3-

```
<?php
 $name = "Ravi "; $ret =  print $name;
 //To test it returns or not
echo $ret;
 ?>
```

Output

Ravi

PHP Data Types

PHP data types are used to hold different types of data or values. PHP supports 8 primitive data types that can be categorized further in 3 types:

1. Scalar Types
2. Compound Types
3. Special Types

PHP Data Types: Scalar Types

There are 4 scalar data types in PHP.

1. boolean
2. integer
3. float
4. string

PHP Data Types: Compound Types

There are 2 compound data types in PHP.

1. array
2. object

PHP Data Types: Special Types

There are 2 special data types in PHP.

1. resource
2. NULL

→ Integers

- Integers are whole numbers, like 1, 12, and 256. The range of acceptable values varies according to the details of your platform but typically extends from -2,147,483,648 to +2,147,483,647. Integer literals can be written in decimal, octal, or hexadecimal. Decimal values are represented by a sequence of digits, without leading zeros. The sequence may begin with a plus (+) or minus (-) sign. If there is no sign, positive is assumed. **Ex: 1998, -641, +33**
- ➢ Octal numbers consist of a leading 0 and a sequence of digits from 0 to 7. Like decimal numbers, octal numbers can be prefixed with a plus or minus.

  Ex:- 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. Like decimal and octal values, you can include a sign in hexadecimal numbers:

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

Use the is_int( ) function (or its is_integer( ) alias) to test whether a value is an integer:

```
if (is_int($x))
// $x is an integer
```

→ Floating-Point Numbers

Floating-point numbers (often referred to as real numbers) represent numeric values with decimal digits. Like integers, their limits depend on your machine's details. PHP floating-point numbers are equivalent to the range of the double data type of your C compiler. Usually, this allows numbers between 1.7E-308 and 1.7E+308 with 15 digits of accuracy.

3.14

-7.1

0.314E1 // $0.314*10^1$, or 3.14

17.0E-3 // $17.0*10^{-3}$, or 0.017

Use the is_float( ) function (or its is_real( ) alias) to test whether a value is a floating point number:

```
if (is_float($x))
 // $x is a floating-point number
```

→ Booleans

A boolean value represents a "truth value"—it says whether something is true or not. In PHP, the following values are false:

- The keyword false
- The integer 0
- The floating-point value 0.0
- The empty string ("") and the string "0"
- An object with no values or functions
- The NULL value

Use the is_bool( ) function to test whether a value is a boolean:

```
if (is_bool($x))
   // $x is a boolean_________________
```

→ 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 within double quotes, while within single quotes they are not:

```
$name = "Guido";
echo "Hi, $name\n";
echo 'Hi, $name';
```

Output:

```
Hi, Guido
Hi, $name
```

Double quotes also support a variety of string escapes.

Escape sequences in double-quoted strings

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

\}	Right brace
\[	Left bracket
\]	Right bracket
\0 through \777	ASCII character represented by octal value
\x0 through \xFF	ASCII character represented by hex value

A single-quoted string only recognizes \\ to get a literal backslash and \' to get a literal single quote:

```
$dos_path = 'C:\\WINDOWS\\SYSTEM';
$publisher = 'Tim O\'Reilly';
echo "$dos_path $publisher\n";
```

```
C:\WINDOWS\SYSTEM Tim O'Reilly
```

To test whether two strings are equal, use the == comparison operator:

```
if ($a == $b) { echo "a and b are equal" }
```

Use the is_string( ) function to test whether a value is a string:

```
if (is_string($x))
   // $x is a string
```

→ Objects

PHP supports object-oriented programming (OOP). OOP promotes clean modular design, simplifies debugging and maintenance, and assists with code reuse.Classes are the unit of object-oriented design. A class is a definition of a structure that contains properties (variables) and methods (functions). Classes are defined with the classkeyword. Once a class is defined, any number of objects can be made from it with the new keyword, and the properties and methods can be accessed with the -> construct:

```
class Person
{
  var $name = '';
  function name ($newname = NULL)
{
      if (! is_null($newname))
      $this->name = $newname;
       return $this->name;
  }
}
$ed = new Person;
$ed->name('Edison');
printf("Hello, %s\n", $ed->name);
```

```
Hello, Edison
```

Use the is_object( ) function to test whether a value is an object:

```
if (is_object($x)) {
   // $x is an object
}
```

→ Arrays

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):

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

Use the is_array( ) function to test whether a value is an array:

```
if (is_array($x)) {
   // $x is an array
}
```

→Resources

Resources are really integers under the surface. Their main benefit is that they're garbage collected when no longer in use. When the last reference to a resource value goes away, the extension that created the resource is called to free any memory, close any connection, etc. for that resource:

```
function search ( )
{
  $res = database_connect( );     // fictitious function
  $database_query($res);          // database connection automatically closed
}
```

Use the is_resource( ) function to test whether a value is a resource:

```
if (is_resource($x))
   // $x is a resource
```

→ NULL

The NULL value represents a variable that has no value.

```
$aleph = Null;         // same
$aleph = NULL;      // same
```

Use the is_null( ) function to test whether a value is NULL—for instance, to see whether a variable has a value:

```
if (is_null($x))
   // $x is NULL
```

Magic Constants

Magic constants are the predefined constants in PHP which get changed on the basis of their use. They start with double underscore (__) and ends with double underscore.

Here are some of the useful constants that are made available by PHP:

→ __LINE__ returns the line number in the source file.

```
<?php
echo "line number: " . __LINE__;
?>
```

→ __FILE__ represents the name of your file, including its full path.

```
<?php
echo "the name of this file is: " . __FILE__;
?>
```

→ __DIR__ represents only the path to the file:

```
<?php
echo "the directory of this file is: " . __DIR__;
?>
```

→ __CLASS__ returns the name of the current class:

```
<?php
class Sample
{
   public function __construct() {
      echo __CLASS__;
   }
}
$obj = new Sample(); // Sample
?>
```

→ __FUNCTION__ returns the name of the current function:

```
<?php
function mySampleFunc()
{
   echo  "the name the function is: " . __FUNCTION__;
}
mySampleFunc(); //the name of function is: mySampleFunc

?>
```

→ __METHOD__ represents the name of the current method:

```
<?php
class Sample
{
   public static function myMethod() {
      echo  "the name of method is: " . __METHOD__;
   }
}

Sample::myMethod();
?>
```

→ __NAMESPACE__ returns the name of the current namespace:

```
<?php
namespace MySampleNS;
echo "the namespace is: " . __NAMESPACE__;
// the name space is: MySampleNS
?>
```

Magic Methods

→__construct() is magic method which PHP invokes to create object instances of your class. It can accept any number of arguments.

```
<?php
class MySample
{
   public function __construct($foo) {
      echo __CLASS__ . " constructor called with $foo.";
   }
}
$obj = new MySample(42);
```

Output: MySample constructor called with 42

→ __destruct() method is called when the object is destroyed by PHP's garbage collector. It accepts no arguments.

```
<?php
class MySample
{
   public function __destruct() {
      echo__CLASS__ . " destructor called.";
   }
}
$obj = new MySample;                    // MySample destructor called
```

```
?>
```

PHP | Operators

Operators are used to perform operations on some values. In other words, we can describe operators as something that takes some values, performs some operation on them and gives a result. From example, “1 + 2 = 3” in this expression ‘+’ is an operator. It takes two values 1 and 2, performs addition operation on them to give 3.

Given below are the various groups of operators:

- Arithmetic Operators
- Logical or Relational Operators
- Comparison Operators
- Conditional or Ternary Operators
- Assignment Operators
- Spaceship Operators (Introduced in PHP 7)
- Array Operators
- Increment/Decrement Operators
- String Operators

→Arithmetic Operators

Operator	Name	Syntax	Operation
+	Addition	$x + $y	Sum the operands
–	Subtraction	$x – $y	Differences the operands
*	Multiplication	$x * $y	Product of the operands
/	Division	$x / $y	Quotient of the operands
**	Exponentiation	$x ** $y	$x raised to the power $y
%	Modulus	$x % $y	Remainder of the operands

```
<?php
$x=29;$y=4;
Echo $x+$y.”br/”;
Echo $x-$y.”br/”;
Echo $x*$y.”br/”;
Echo $x/$y.”br/”;
Echo $x%$y.”br/”;
Echo $x**$y;
?>
```

Output:

```
33
25
116
7.25
1
```

→Logical or Relational Operators

Operator	Name	Syntax	Operation
and	Logical AND	$x and $y	True if both the operands are true else false
or	Logical OR	$x or $y	True if either of the operand is true else false
xor	Logical XOR	$x xor $y	True if either of the operand is true and false if both are true
&&	Logical AND	$x && $y	True if both the operands are true else false
||	Logical OR	$x || $y	True if either of the operand is true else false
!	Logical NOT	!$x	True if $x is false

```
<?php

$x=50; $y = 30;
if ($x == 50 and $y == 30)

echo “and success”;

if ($x == 50 or $y == 20)

    echo "or Success \n";

if ($x == 50 xor $y == 20)

    echo "xor Success ";

if ($x == 50 && $y == 30)

    echo "&& Success \n";
```

```
 if ($x == 50 || $y == 20)
   echo "|| Success \n";
 if (!$z)
   echo "! Success \n";
 ?>
```

Output:

```
and Success
or Success
xor Success
&& Success
|| Success
! Success
```

→Comparison Operators

These operators are used to compare two elements and outputs the result in boolean form.

Operator	Name	Syntax	Operation
==	Equal To	$x == $y	Returns True if both the operands are equal
!=	Not Equal To	$x != $y	Returns True if both the operands are not equal
<>	Not Equal To	$x <> $y	Returns True if both the operands are not equal
===	Identical	$x === $y	Returns True if both the operands are equal and are of the same type
!==	Not Identical	$x !== $y	Returns True if both the operands are unequal and are of different types
<	Less Than	$x < $y	Returns True if $x is less than $y
>	Greater Than	$x > $y	Returns True if $x is greater than $y
<=	Less Than or Equal To	$x <= $y	Returns True if $x is less than or equal to $y

>=	Greater Than or Equal To	$x >= $y	Returns True if $x is greater than or equal to $y

```
<?php
 $a = 80;
$b = 50;
$c = "80";
var_dump($a == $c) + "\n";
var_dump($a != $b) + "\n";
var_dump($a <> $b) + "\n";
var_dump($a === $c) + "\n";
var_dump($a !== $c) + "\n";
var_dump($a < $b) + "\n";
var_dump($a > $b) + "\n";
var_dump($a <= $b) + "\n";
var_dump($a >= $b);
 ?>
```

Output:

```
bool(true)
bool(true)
bool(true)
bool(false)
bool(true)
bool(false)
bool(true)
bool(false)
bool(true)
```

→Conditional or Ternary Operators

These operators are used to compare two values and take either of the result

simultaneously, depending on whether the outcome is TRUE or FALSE.
Syntax:

$var = (condition)? value1 : value2;

Operator	**Name**	**Operation**
?:	Ternary	If condition is true ? then $x : or else $y. This means that if condition is true then left result of the colon is accepted otherwise the result on right.

```
<?php
 $x = -12;
 echo ($x > 0) ? 'The number is positive' : 'The number is negative';
 ?>
```

Output:

```
The number is negative
```

→Assignment Operators

Operator	**Name**	**Syntax**	**Operation**
=	Assign	$x = $y	Operand on the left obtains the value of the operand on right
+=	Add then Assign	$x += $y	Simple Addition same as $x = $x + $y
-=	Subtract then Assign	$x -= $y	Simple subtraction same as $x = $x – $y
*=	Multiply then Assign	$x *= $y	Simple product same as $x = $x * $y
/=	Divide then Assign (quotient)	$x /= $y	Simple division same as $x = $x / $y
%=	Divide then Assign (remainder)	$x %= $y	Simple division same as $x = $x % $y

Example:

```
<?php
 // simple assign operator
$y = 75;
echo $y, "\n";
 // add then assign operator
$y = 100;
$y += 200;
echo $y, "\n";
 // subtract then assign operator
$y = 70;
$y -= 10;
echo $y, "\n";
// multiply then assign operator
$y = 30;
$y *= 20;
echo $y, "\n";
// Divide then assign(quotient) operator
$y = 100;
$y /= 5;
echo $y, "\n";
// Divide then assign(remainder) operator
$y = 50;
$y %= 5;
echo $y;
?>
```

Output:

```
75
300
60
600
20
0
```

→String Operators

Operator	Name	Syntax	Operation
.	Concatenation	$x.$y	Concatenated $x and $y
.=	Concatenation and assignment	$x.=$y	First concatenates then assigns, same as $x = $x.$y

Example:

```
<?php
$x = "Geeks";
$y = "for";
$z = "Geeks!!!";
echo $x . $y . $z, "\n";
 $x .= $y . $z;
echo $x;
 ?>
```

Output:

```
GeeksforGeeks!!!
GeeksforGeeks!!!
```

→Array Operators in PHP

We can perform operations on arrays using **Arrays Operators** like Union, Equality, Identity,Inequality and Non-identity.

List of Array Operators

Name	Syntax	Result
Union	$array1 + $array2	Union of $array1 and $array2.
Equality	$array1 == $array2	TRUE if $array1 and $array2 have the same key/value pairs.
Identity	$array1 === $array2	TRUE if $array1 and $array2 have the same key/value pairs in the same order and of the same types.
Inequality	$array1 != $array2	TRUE if $array1 is not equal to $array2.
Inequality	$array1 <> $array2	TRUE if $array1 is not equal to $array2.
Non-identity	$array1 !== $array2	TRUE if $array1 is not identical to $array2.

➔Union Array Operator (+)

The Union operator (+) appends right-hand oparand (array) to the left-hand oparand (array). If there are keys that exist in both arrays, then the elements from the left-hand array will be used, and the matching elements from the right-hand array will be ignored.

Example:

```
<?php
$array1 = array("a"=>"bca","b"=>"mca");
 $array2 = array("a"=>"bca","d"=>"mca");
echo "Union of \$array1 and \$array2: <br>";
 print_r($array1 + $array2);
echo "<br>Union of \$array2 and \$array1: <br>";
print_r($array2 + $array1);
```

```
?>
```

Output:

```
Union of $array1 and $array2:
Array ( [a] => bca [b] => mca [d] => mca )
Union of $array2 and $array1:
Array ( [a] => bca [d] => mca [b] => mca )
```

➔Equality Array Operator (==)

Equality Array Operator is used to check if both arrays have the same elements or not.
If array1 and array2 have the same elements (key value pairs) then Boolean true is returned, else Boolean false is returned.

Example :

```
<?php
$array1 = array("Java","Smalltalk","Objective-C","Perl");
$array2 = array("Java","Smalltalk","Objective-C","Perl");
$array3 = array("Java","Smalltalk","Objective-C","Perl",".Net");
echo var_dump($array1 == $array2) ."<br>";
echo var_dump($array2 == $array3) ."<br>";
?>
```

Output

```
bool(true)
bool(false)
```

→Inequality Array Operator (!=)

Inequality Array Operator is used to check if both arrays have the same elements or not.
If array1 and array2 have do not have same elements (key value pairs) then boolean false is returned, else boolean true is returned.

Example :

```
<?php
$array1 = array("Java","Smalltalk","Objective-C","Perl");
$array2 = array("Java","Smalltalk","Objective-C","Perl");
$array3 = array("Java","Smalltalk","Objective-C","Perl",".Net");
echo var_dump($array1 != $array2) ."<br>";
echo var_dump($array2 != $array3) ."<br>";
```

```
?>
```

Output

```
bool(false)
bool(true)
```

→Identity Array Operator (===)

Two arrays are Identical Only when the Keys and Values both are equal.

Example :

```
<?php
$array1 = array("PHP", "Java");
$array2 = array("1" => "Java", "0" => "PHP");
// == returns true
echo var_dump($array1 == $array2) ."<br>";
// === returns false as keys are different
echo var_dump($array1 === $array2) ."<br>";
?>
```

Output

```
bool(true)
bool(false)
```

→Non-identity Array Operator (!==)

Two arrays are Non-identity when the Keys and Values both are not equal.

Example :

```
<?php
$array1 = array("PHP", "Java");
$array2 = array("1" => "Java", "0" => "PHP");
// == returns false
echo var_dump($array1 != $array2) ."<br>";
// === returns true as keys are different
echo var_dump($array1 !== $array2) ."<br>";
?>
```

Output

```
bool(false)
bool(true)
```

→Increment/Decrement Operators

These are called the unary operators as it work on single operands. These are used to increment or decrement values.

Operator	Name	Syntax	Operation
++	Pre-Increment	$x++	First returns $x, then increment it by one
—	Pre-Decrement	$x–	First returns $x, then decrement it by one
++	Post-Increment	++$x	First increments $x by one, then return $x
—	Post-Decrement	–$x	First decrements $x by one, then return $x

Example:

```
<?php
  $x = 2;
$y=++$x;
echo $x." ".$y;
 echo "<br/>";
  $x = 2;
$y=$x++;
echo $x." ".$y;
 ?>
```

Output:

```
3 3
3 2
```

→Spaceship Operators (Introduced in PHP 7)

Operator	Syntax	Operation
$x < $y	$x <=> $y	Identical to -1 (right is greater)
$x > $y	$x <=> $y	Identical to 1 (left is greater)
$x <= $y	$x <=> $y	Identical to -1 (right is greater) or identical to 0 (if both are equal)
$x >= $y	$x <=> $y	Identical to 1 (if left is greater) or identical to 0 (if both are equal)
$x == $y	$x <=> $y	Identical to 0 (both are equal)
$x != $y	$x <=> $y	Not Identical to 0

Example:

```
<?php
$x = 50;
$y = 50;
$z = 25;
echo $x <=> $y.”<br/>”;
echo $x <=> $z.”<br/>”;
echo $z <=> $y;
 // We can do the same for Strings
$x = "Ram";
$y = "Krishna";
echo $x <=> $y.”<br/>”;
echo $x <=> $y.<br/>;
echo $y <=> $x;
?>
```

Output:

```
0
1
-1
1
1
-1
```

Bitwise Operators

Bitwise operators allow evaluation and manipulation of specific bits within an integer.

Bitwise Operators

Example	Name	Result
$a & $b	And	Bits that are set in both *$a* and *$b* are set.
$a | $b	Or (inclusive or)	Bits that are set in either *$a* or *$b* are set.
$a ^ $b	Xor (exclusive or)	Bits that are set in *$a* or *$b* but not both are set.
~ $a	Not	Bits that are set in *$a* are not set, and vice versa.
$a << $b	Shift left	Shift the bits of *$a* *$b* steps to the left (each step means "multiply by two
$a >> $b	Shift right	Shift the bits of *$a* *$b* steps to the right (each step means "divide by two"

Decision Making Statements in PHP

Decision making statements in PHP are used to take some decision in the PHP Script. Decision making statements allows you to control execution of certain part of the code based on the outcome of the condition at run time.

PHP supports following Decision Making Statements:

(1) Simple If Statement

(2) If...else Statement

(3) Nested If Statement

(4) If...elseif Ladder

(5) Switch...Case Statement

→Simple IF Statement

Syntax
```
  if (condition )
  {
     True Statement;
  }
```
Example
```
    <?php
      $mark=23;
      if ($mark < 35)
          echo "Fail";
      ?>
```

→ If...else Statement

Syntax:
```
   if (condition )
{
True Statement;
   }
   else
   {
      False Statement;
   }
```

Example:
```
  <?php
        $mark=20;
        if ($mark < 35)
           echo "Fail";
        else
           echo "Pass";
   ?>
```

→Nested If Statement

Syntax
```
   if (OuterCondition)
   {
      if (InnerCondition)
      {
         True Statement Block for InnerCondition
      }
      else
      {
         Fasle Statement Block for InnerCondition
      }
   }
   else
   {
      False Statement Block for OuterCondition
   }
```

Example

```
 <?php
     $temp=38;
     if($temp>20)
     {
        if($temp>40)
        echo "Very Hot...";
        else
        echo "Hot...";
       }
     else
           echo "Cool....";
     ?>
```

→If...elseif Ladder

Syntax

```
  if (Condition1)
  {
    True Statement Block1;
  }
  else if (Condition2)
  {
    True Statement Block2;
  }
  .........................
    else if (Condition N)
  {
    True Statement BlockN;
  }
  else
  {
     Default Statement Block;
  }
```

Example

```
  <?php
    $per=45;
    if ($per >= 66)
              echo "Distinctionn";
    else if ($per <66 && $per >=56)
             echo "First Class";
     else if ($per <56 && $per >= 44)
              echo "Second Class";
     else if ($per <44 && $per >= 35)
               echo "Pass Class";
      else
           echo "Sorry Fail";
   ?>
```

→**Switch Case Statement**

Syntax

```
  switch(Variable or Value)
  {
    case Value1:
            Statement Block For Value1;
            break;
    case Value2:
            Statement Block For Value2;
            break;
    ............
      case ValueN:
            Statement Block For ValueN;
            break;
    default:
          Default Statement Block For No Match;
  }
```

Example

```
  <?php
     $ch = 1;
     $a = 2;
     $b = 3;
     switch ($ch)
     {
       case 1:
              $c=$a + $b;
              echo "Answer=".$c;
                break;
       case 2:
              $c=$a - $b
                echo "Answer=".$c;
               break;
       case 3:
              $c=$a * $b;
              echo "Answer=". $c;
                break;
       case 4:
              $c=$a / $b;
              echo "Answer=". $c;
              break;
       default:
         echo "Wrong Choice";
     }
   ?>
```

Looping Control Statement in PHP

Using looping control statement same task can be performed repeatadely for specific amount of time or until desired condition is not satisfied.

PHP supports following Looping Control Statements:

(1) While Loop

(2) Do...While Loop

(3) For Loop

(4) Foreach Loop

→While Loop

This process is repeated until condition specified within while loop becomes false. It is also known as entry controlled loop because the condition is checked first and then body of the loop is executed based on the condition. So if condition is false at the first trial the body of loop will never executes.

Syntax

```
   while (condition)
   {
       body of loop;
   }
```

Example

```
<?php
  $i=1;
 while($i<=10)
 {
       echo $i." ";
       $i++;
 }
?>
```

Output

```
1 2 3 4 5 6 7 8 9 10
```

→Do While Loop

It is also known as exit controlled loop because the condition is checked at last and then body of the loop is executed based on the condition. So even if condition is false at the first trial the body of loop executes at least once.

Syntax

```
    do
    {
         body of loop;
    } while (condition);
```

Example

```
<?php
   $i=1;
  do
  {
    echo $i." ";
     $i++;
  }while($i<=10);
```

Output

```
1 2 3 4 5 6 7 8 9 10
```

→For Loop

for loop is used to repeatedly perform some task for specific number of times. It is also known as iterative statement. It is useful when you know in advance how many times you want to repeat the task.

Syntax

```
   for(initialization;condition;iteration)
    {
         body of loop;
    }
```

Here,

Initialization: It indicates the starting point for the loop.

Condition: It indicates control condition for the loop. Control condition determines when loop will stop.

Iteration: You can either increment or decrement the value of the counter variable based on your requirement.

Example

```
<?php
  for ($i=1;$i<=10;$i++)
       echo $i. " ";
 ?>
```

Output

```
1 2 3 4 5 6 7 8 9 10
```

→foreach loop

foreach looping structure is used to iterate throgh each elements of an array starting from first element.In foreach loop how many times the loop will iterate depands on the number of elements in an array.foreach looping structure is used specially with Associative array.

Syntax

```
foreach ($ArrayName as $Value)
{
  Body of loop
}
```

Example

```
<?php
$MyArray = array (Yesha, Vidhi, Jiwansh);
foreach ($MyArray as $Value)
{
  echo $value. " ";
}
?>
```

Output

```
Yesha  Vidhi  Jiwansh
```

ARRAY

PHP - Arrays

An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- **Numeric array** – An array with a numeric index. Values are stored and accessed in linear fashion.
- **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.
- **Multidimensional array** – An array containing one or more arrays and values are accessed using multiple indices

Numeric Array(Indexed 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:**Here we have used** array() **function to create array.**

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

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.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE – Don't keep associative array inside double quote while printing otherwise it would not return any value.

Example

```
<?php

$dept=array("bba" =>50,"bca"=>60,"mba"=>20,"mca"=>70);

echo "values"."<br/>";

foreach($dept as $value)

        echo $value."<br/>";

foreach($dept as $x=>$y)

        echo $x." ".$y."<br/>";

?>
```

Output:

```
Values
50
60
20
70
Keys=>Values
bba 50
bca 60
mba 20
mca 70
```

Multidimensional Arrays

PHP multidimensional array is also known as array of arrays. It allows you to store tabular data in an array. PHP multidimensional array can be represented in the form of matrix which is represented by row * column.

Example

```
<?php
$student=array(                    // A two-dimensional array
array(1,"p1",40),
array(2,"p2",60),
array(3,"p3",70)
);
//One way
foreach($student as $x)
{
        foreach($x as $val)
                echo $val." ";
      echo "<br/>";
}
//Another way

for($r=0;$r<3;$r++)
{
        for($c=0;$c<3;$c++)
                echo $student[$r][$c]."<br/>";
}
?>
```

Example:

```
<?php
$dept=array(
"BCA"=>array("progin C"=>40,"DS"=>55),
"BBA"=>array("HRM"=>70,"FINANCE"=>85));
foreach($dept as $val=>$x)
{
        echo $val."<br/>";
        foreach($x as $da=>$y)
        echo $da." ".$y."<br/>";
        echo "<br/>";
}
?>
```

Output:

```
BCA
progin C 40
DS 55
BBA
HRM 70
FINANCE 85
```

The Dynamic Variable (Arrays)

Example

```
$numberList3 = array();
for($i = 0; $i < 10; $i++)
{

$numberList3[$i] = $i;

}
print_r($numberList3);
```

Result

```
Array ( [0] => 0 [1] => 1 [2] => 2 [3] => 3 [4] => 4 [5] => 5 [6] => 6 [7] => 7 [8] => 8
[9] => 9 )
```

Here dynamically create an array with values of 0-9 in that order. We created a for loop and each time we iterated, or went through the loop, we added a new key and value pair to our new array.

Common PHP Array Functions

```
<?php
```

//array_pop():-This function removes an element from the end of an array.

```
$aArray = ["a","b","c"];
array_pop($aArray);
print_r($aArray);
echo "</br>";
```

//array_push():-This function adds an element to the end of an array.

```
$bArray = array("a","b","c");
array_push($bArray,"d");
print_r($bArray);
echo "</br>";
```

//array_shift():-This function removes an element from the beginning of an array.

```
$dArray = array("a","b","c");
array_shift($dArray);
print_r($dArray);;
echo "</br>";
```

//array_unshift():-This function adds an element to the beginning of an array.

```
$dArray = array("a","b","c");
array_unshift($dArray,"d");
print_r($dArray);;
echo "</br>";
```

//array_reverse():-The function reverses the order of elements in an array.

```
$data = array(10, 20, 25, 60);
print_r(array_reverse($data));
echo "</br>";
```

//array_merge():-This function merges two or more arrays to create a single composite array.

```
$data1 = array("cat", "goat");

$data2 = array("goat", "cow");

print_r(array_merge($data1, $data2));

echo "</br>";
```

//array_unique():-This function strips an array of duplicate values.

```
$data = array(1,1,4,6,7,4);

print_r(array_unique($data));

echo "</br>";
```

//array_values():-This function accepts a PHP array and returns a new array containing only its values

```
$data = array("MCA" => 50, "MBA" =>80);

print_r(array_values($data));

echo "</br>";
```

//array_keys():-This function accepts a PHP array and returns a new array containing only its keys

```
$data = array("MCA" => 50, "MBA" =>80);

print_r(array_keys($data));

echo "</br>";
```

//sizeof():-This function returns the number of elements in an array.

```
$data = array("red", "green", "blue");

echo "Array has " . sizeof($data) . " elements";

echo "</br>";
```

//count():-This function returns the number of elements in an array.

```
$data = array("red", "green", "blue");

echo "Array has " . count($data) . " elements";

echo "</br>";
```

//max():-This function returns the greatest number in an array.

```
$data = array(10,20,30);

echo "The greater number is " . max($data);

echo "</br>";
```

//min():-This function returns the smallest number in an array.

```
$data = array(10,20,30);

echo "The greater number is " . min($data);

echo "</br>";
```

//array_flip():-The function exchanges the keys and values of a PHP associative array.

```
$data = array("a" => "apple", "b" => "ball");

print_r(array_flip($data));

echo "</br>";
```

//sort():-This function sorts the elements of an array in ascending order. String values will be arranged in ascending alphabetical order.

```
$data = array("g", "t", "a", "s");

sort($data);

print_r($data);

echo "</br>";
```

//array_search():-This function searches the values in an array for a match to the search term, and returns the corresponding key if found.

```
$cArray = array("a","b","c");

echo array_search("a",$cArray);

echo "</br>";
```

//array_rand():-This function selects one or more random elements from an array.

```
$data = array("white", "black", "red");

echo "Today's color is " . $data[array_rand($data)];

echo "<br/>";
```

//array_slice():-This function is useful to extract a subset of the elements of an array, as another array.

```
$data = array("vanilla", "strawberry", "mango", "peaches");

print_r(array_slice($data, 1, 2));

echo "<br/>";
```

//each():- Each time each() is called, it returns the current key-value pair and moves the array cursor forward one element.

```
$data = array("bca" => "pm", "mca" => "mp");

while (list($key, $value) = each($data))

echo "$key: $value \n";echo "<br/>";
```

//array_change_key_case():-This function changes the case of all key of an array.

```
$salary=array("Sonoo","Vimal"=>"250000","Ratan"=>"200000");

print_r(array_change_key_case($salary,CASE_UPPER));

echo "<br/>";
```

//array_chunk() :-By using array_chunk() method, you can divide array into many parts.

```
$salary=array(10,20,30,40);

print_r(array_chunk($salary,2));

?>
```

Array Operators in PHP : Tutorial

We can perform operations on arrays using **Arrays Operators** like Union, Equality, Identity,Inequality and Non-identity.

List of Array Operators

Name	**Syntax**	**Result**
Union	$array1 + $array2	Union of $array1 and $array2.
Equality	$array1 == $array2	TRUE if $array1 and $array2 have the same key/value pairs.
Identity	$array1 === $array2	TRUE if $array1 and $array2 have the same key/value pairs in the same order and of the same types.
Inequality	$array1 != $array2	TRUE if $a is not equal to $b.
Inequality	$array1 <> $array2	TRUE if $a is not equal to $b.
Non-identity	$array1 !== $array2	TRUE if $a is not identical to $b.

➔Union Array Operator (+)

The Union operator (+) appends right-hand oparand (array) to the left-hand oparand (array).
If there are keys that exist in both arrays, then the elements from the left-hand array will be used, and the matching elements from the right-hand array will be ignored.

Example:

```
<?php
$array1 = array("a"=>"bca","b"=>"mca");
 $array2 = array("a"=>"bca","d"=>"mca");
echo "Union of \$array1 and \$array2: <br>";
 print_r($array1 + $array2);
echo "<br>Union of \$array2 and \$array1: <br>";
print_r($array2 + $array1);
```

```
?>
```

Output:

```
Union of $array1 and $array2:
Array ( [a] => bca [b] => mca [d] => mca )
Union of $array2 and $array1:
Array ( [a] => bca [d] => mca [b] => mca )
```

➔Equality Array Operator (==)

Equality Array Operator is used to check if both arrays have the same elements or not.
If array1 and array2 have the same elements (key value pairs) then Boolean true is returned, else Boolean false is returned.

Example :

```
<?php
$array1 = array("Java","Smalltalk","Objective-C","Perl");
$array2 = array("Java","Smalltalk","Objective-C","Perl");
$array3 = array("Java","Smalltalk","Objective-C","Perl",".Net");
echo var_dump($array1 == $array2) ."<br>";
echo var_dump($array2 == $array3) ."<br>";
?>
```

Output

```
bool(true)
bool(false)
```

→Inequality Array Operator (!=)

Inequality Array Operator is used to check if both arrays have the same elements or not.
If array1 and array2 have do not have same elements (key value pairs) then boolean false is returned, else boolean true is returned.

Example :

```
<?php
$array1 = array("Java","Smalltalk","Objective-C","Perl");
$array2 = array("Java","Smalltalk","Objective-C","Perl");
$array3 = array("Java","Smalltalk","Objective-C","Perl",".Net");
```

```
echo var_dump($array1 != $array2) ."<br>";
echo var_dump($array2 != $array3) ."<br>";
?>
```

Output

```
bool(false)
bool(true)
```

→Identity Array Operator (===)

Two arrays are Identical Only when the Keys and Values both are equal.

Example :

```
<?php
$array1 = array("PHP", "Java");
$array2 = array("1" => "Java", "0" => "PHP");
// == returns true
echo var_dump($array1 == $array2) ."<br>";
// === returns false as keys are different
echo var_dump($array1 === $array2) ."<br>";
?>
```

Output

```
bool(true)
bool(false)
```

→Non-identity Array Operator (!==)

Two arrays are Non-identity when the Keys and Values both are not equal.

Example :

```
<?php
$array1 = array("PHP", "Java");
$array2 = array("1" => "Java", "0" => "PHP");
// == returns false
echo var_dump($array1 != $array2) ."<br>";
// === returns true as keys are different
echo var_dump($array1 !== $array2) ."<br>";
?>
```

Output

```
bool(false)
bool(true)
```

→Write a Program for finding the biggest & smallest number in an array without using any array functions.

```
<?php
$numbers = array(12,23,45,20,5,6,34,17,9,56,999);
$length = count($numbers);
$max = $numbers[0];
$min=$numbers[0];

for($i=1;$i<$length;$i++)
 {
        if($max<$n[$i])

                $max=$n[$i];

        if($min>$n[$i])

                $min=$n[$i];

  }

echo "The biggest number is ".$max."<br/>";
echo "The smallest number is ".$min;

?>
```

→Write a Program for sorting in PHP

```
<?php
$n=array(10,2,30,4,70,7);
for($i=0;$i<5;$i++)
{
        for($j=$i+1;$j<6;$j++)
        {
                if($n[$i]>$n[$j])
                {
                        $temp=$n[$i];
                        $n[$i]=$n[$j];
                        $n[$j]=$temp;
                }
        }
}
echo "Sorted list in ascending order <br />";
print_r($n);
```

//→Write a Program for searching an element in PHP

```
$n=array(10,2,30,4,70,7);
$key=30;
for($i=0;$i<6;$i++)
{
        if($n[$i]==$key)
                echo "$key is present in".($i+1)."location ";
}  ?>
```

FUNCTION

PHP Functions

A function is a self-contained block of code that performs a specific task.PHP has a huge collection of internal or built-in functions that you can call directly within your PHP scripts to perform a specific task, like `gettype()`, `print_r()`, `var_dump`, etc.In addition to the built-in functions, PHP also allows you to define your own functions.

Here are some advantages of using functions:

- **Functions reduces the repetition of code within a program** — Function allows you to extract commonly used block of code into a single component. Now can you can perform the same task by calling this function wherever you want without having to copy and paste the same block of code again and again.
- **Functions makes the code much easier to maintain** — Since a function created once can be used many times, so any changes made inside a function automatically implemented at all the places without touching the several files.
- **Functions makes it easier to eliminate the errors** — When the program is subdivided into functions, if any error occur you know exactly what function causing the error and where to find it. Therefore, fixing errors becomes much easier.
- **Functions can be reused in other application** — because a function is separated from the rest of the script, it's easy to reuse the same function in other applications just by including the php file containing those functions.

There are two types of functions as:

1. Internal (built-in) Functions
2. User Defined Functions

Rules for naming user defined functions:

- A valid function name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.
- Function names are case-insensitive.
- The function body enclosed within a pair of braces . The opening curly brace **({)** indicates the beginning of the function code and the closing curly **(})** brace indicates the termination of the function.
- All functions in PHP start with function().

- Functions can also accept parameters.
- A function can also be used to return values.

Syntax:

function functionName()

{ statement 1 : statement 2 : statement 3 : ...... }

Example:

```
<?php
function myFunction(){
  echo "Example PHP function";
}
myFunction();
?>
```

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.

Example:

```
<?php
function addFunction($number, $number)
{
  $number = $number + $number;
  echo "The sum of two numbers is : $number";
}
addFunction(10, 20);
?>
```

Output:- The sum of two numbers is:30

PHP Default Argument Value

```
<?php
function sum($a=10,$b=20,$c=30)
{
        echo "sum=".($a+$b+$c)."<br/>";
}
sum(10,20,30);
sum(2,3);
sum(4);
sum();
?>
```

PHP Functions - Returning values

All functions in PHP return a value, even if you don't explicitly cause them to. Thus, the concept of "void" functions does not really apply to PHP. You can specify the return value of your function by using the return keyword:

Example1:
```
<?php
function hello(){
  return "Hello World"; // No output is shown
}
$txt = hello(); // Assigns the return value "Hello World" to $txt
echo hello(); // Direct Displays "Hello World"
?>
```

Example2:
```
<?php
function sum($x, $y) {
   $z = $x + $y;
   return $z;
}
echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

Output

```
5 + 10 = 15
7 + 13 =20
2 + 4 =6
```

Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. Following example depicts this behaviour.

```
<html>
     <head>
     <title>Dynamic Function Calls</title>
   </head>
     <body>
       <?php
        function sayHello() {
           echo "Hello<br />";
        }
         $function_holder = "sayHello";
        $function_holder();
```

```
    ?>
      </body>
</html>
```

This will display following result –

Hello

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.

```
<?php
function cube(&$x)
{
$x = $x * $x * $x;
}
$result = 5;
cube($result);
echo $result;
?>
```

Output: 125

Scope

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

- Local variables
- Function parameters
- Global variables
- Static variables.

Local Variables

A variable declared in a function is considered local; that is, it can be referenced solely in that function. Any assignment outside of that function will be considered to be an entirely different variable from the one contained in the function –

```
<?php
  $x = 4;
    function assignx () {
    $x = 0;
    print "\$x inside function is $x. <br />";
  }
```

```
    assignx();
  print "\$x outside of function is $x. <br />";
?>
```

```
This will produce the following result –
$x inside function is 0.
$x outside of function is 4.
```

Global Variables

In contrast to local variables, a global variable can be accessed in any part of the program. However, in order to be modified, a global variable must be explicitly declared to be global in the function in which it is to be modified. This is accomplished, conveniently enough, by placing the keyword **GLOBAL**in front of the variable that should be recognized as global.

```
<?php
  $somevar = 15;
    function addit()
 {
    GLOBAL $somevar;
    $somevar++;
      print "Somevar is $somevar";
  }
  addit();
?>
```

```
This will produce the following result –
Somevar is 16
```

Static Variables

In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.You can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

```
<?php
  function keep_track() {
    STATIC $count = 0;
    $count++;
    print $count;
    print "<br />";
  }
   keep_track();
  keep_track();
  keep_track();
```

```
?>
```

This will produce the following result –

```
1
2
3
```

Function Parameters

Function parameters are declared after the function name and inside parentheses. They are declared much like a typical variable would be –

```
<?php
  // multiply a value by 10 and return it to the caller
  function multiply ($value) {
    $value = $value * 10;
    return $value;
  }
    $retval = multiply (10);
  Print "Return value is $retval\n";
?>
```

This will produce the following result –

```
Return value is 100
```

Recursion

When a function calls itself it is called as recursion.

Ex-1 (Find the factorial of a number using recursion)

```
<div align="center">
<?php
function fact($n)
{
if($n==0)
        return 1;
else
        return $n*fact($n-1);
}
```

```
if(isset($_POST['btn']))
{
$n=$_POST['p1'];
echo "Factoial=".fact($n);
}
?>
<form method="post">
<br>
Enter a number:<input type="text" name="p1">
<input type="submit" name="btn">
</div>
</form>
```

Ex-2(Find the sum of two numbers using recursion)

```
<div align="center">
<?php
function sum($n1,$n2)
{
if($n2==0)
        return $n1;
else
        return sum(++$n1,–$n2);
}
if(isset($_POST['btn']))
{
        $n1=$_POST['p1'];
        $n2=$_POST['p2'];
        echo "sum=".sum($n1,$n2);
}
?>
<form method="post">
<br>
        Enter 1st number:<input type="text" name="p1">
        Enter 2nd number:<input type="text" name="p2">
        <input type="submit" name="btn">
</div>
</form>
```

Ex-3(Find the product of 2 nos using recursion)

```
<div align="center">
<?php
function product($n1,$n2)
{
if($n2==0)
        return 0;
else if($n2==1)
        return $n1;
else
        return  $n1+product($n1,–$n2);
}
if(isset($_POST['btn']))
{
        $n1=$_POST['p1'];
        $n2=$_POST['p2'];
        echo "Multiplication=".product($n1,$n2);
}
?>
<form method="post">
<br>
Enter 1st number:<input type="text" name="p1">
Enter 2nd number:<input type="text" name="p2">
<input type="submit" name="btn">
</div>
</form>
```

Ex-4(Find the power of 2 nos using recursion)

```
<div align="center">
<?php

function power($n1,$n2)
{
if($n2==0)
            return 0;
else if($n2==1)
            return $n1;
else
            return  $n1*power($n1,–$n2);
}
if(isset($_POST['btn']))
{
$n1=$_POST['p1'];
$n2=$_POST['p2'];
echo "Power value=".power($n1,$n2);
}
?>
<form method="post">
<br>
Enter 1st number:<input type="text" name="p1">
Enter 2nd number:<input type="text" name="p2">
<input type="submit" name="btn">
</div>
</form>
```

Ex-5(LCM of 2 nos using recursion)

```
<div align="center">
<?php
function gcd($n1,$n2)
{
        $r=$n1%$n2;
        if($r==0)
                return $n2;
        else
                return  gcd($n2,$r);
}
if(isset($_POST['btn']))
{
        $n1=$_POST['p1'];
        $n2=$_POST['p2'];
        echo "gcd value=".gcd($n1,$n2);
}
?>
<form method="post">
<br>
        Enter 1st number:<input type="text" name="p1">
        Enter 2nd number:<input type="text" name="p2">
        <input type="submit" name="btn">
</div>
</form>
```

Ex-6(Using recursion print the series f(x)=x-x^3/3!+x^5/5!——)

```
<div align="center">

<?php

function series($x,$n)

{

        static $count=1;
        static $sum1=0;
        static $sum=0;
        static $i=1;

                $fact=1;
                $p=$i;
                while($p>0)
                {
                        $fact=$fact*$p;
                        $p–;
                }
                if($count%2!=0)
                        $sum1+=pow($x,$i)/$fact;
                else
                        $sum+=pow($x,$i)/$fact;
                $i+=2;
                $count++;
                if($count<=$n)
                {
                series($x,$n);
                }
                return $sum1-$sum;
}
if(isset($_POST['btn']))
{
$x=$_POST['p1'];
$n=$_POST['p2'];
echo "sum of series=".series($x,$n);

}
?>

<form method="post">
<br>
Enter the number:<input type="text" name="p1">
Enter no of terms:<input type="text" name="p2">
<input type="submit" name="btn">
</div>
</form>
```

STRINGS

Strings

A string is series of characters, where a character is the same as a byte. This means that PHP only supports a 256-character set, and hence does not offer native Unicode support.

Note: As of PHP 7.0.0, there are no particular restrictions regarding the length of a string on 64-bit builds. On 32-bit builds and in earlier versions, a string can be as large as up to 2GB (2147483647 bytes maximum)

Syntax

A string literal can be specified in four different ways:

- single quoted
- double quoted
- heredoc syntax
- nowdoc syntax (since PHP 5.3.0)

→Single quoted

The simplest way to specify a string is to enclose it in single quotes (the character ').

Note: Unlike the double-quoted and heredoc syntaxes, variables and escape sequences for special characters will not be expanded when they occur in single quoted strings.

```
<?php
echo 'Arnold once said: "I\'ll be back"';          // Outputs: Arnold once said: "I'll be
back"
echo 'You deleted C:\\*.*?';                        // Outputs: You deleted C:\*.*?
echo 'This will not expand: \n a newline';         // Outputs: This will not expand: \n a
newline
echo 'Variables do not $expand $either';           // Outputs: Variables do not $expand
$either
?>
```

→Double quoted

If the string is enclosed in double-quotes ("), PHP will interpret the following escape sequences for special characters:

Escaped characters

Sequence	Meaning
\n	linefeed (LF or 0x0A (10) in ASCII)

\r	carriage return (CR or 0x0D (13) in ASCII)
\t	horizontal tab (HT or 0x09 (9) in ASCII)
\v	vertical tab (VT or 0x0B (11) in ASCII) (since PHP 5.2.5)
\e	escape (ESC or 0x1B (27) in ASCII) (since PHP 5.4.4)
\f	form feed (FF or 0x0C (12) in ASCII) (since PHP 5.2.5)
\\	backslash
\$	dollar sign
\"	double-quote

\[0-7]{1,3}   the sequence of characters matching the regular expression is a character in octal notation, which silently overflows to fit in a byte (e.g. "\400" === "\000")

\x[0-9A-Fa-f]{1,2}   the sequence of characters matching the regular expression is a character in hexadecimal notation

\u{[0-9A-Fa-f]+}   the sequence of characters matching the regular expression is a Unicode codepoint, which will be output to the string as that codepoint's UTF-8 representation (added in PHP 7.0.0)

The most important feature of double-quoted strings is the fact that variable names will be expanded.

→Heredoc

A third way to delimit strings is the heredoc syntax: <<<. After this operator, an identifier is provided, then a newline. The string itself follows, and then the same identifier again to close the quotation.

Heredoc string quoting example

```
<?php
$str = <<<EOD
Example of string
spanning multiple lines
using heredoc syntax.
EOD;
echo $str;
```

```
?>
```

➔Nowdoc

Nowdocs are to single-quoted strings what heredocs are to double-quoted strings. A nowdoc is specified similarly to a heredoc, but no parsing is done inside a nowdoc..

A nowdoc is identified with the same <<< sequence used for heredocs, but the identifier which follows is enclosed in single quotes, e.g. <<<'EOT'. All the rules for heredoc identifiers also apply to nowdoc identifiers, especially those regarding the appearance of the closing identifier.

Example #7 Nowdoc string quoting example

```
<?php

$str = <<<'EOD'

Example of string

spanning multiple lines

using nowdoc syntax.

EOD;

echo $str;

?>
```

PHP String Functions

PHP provides various string functions to access and manipulate strings.

→strtolower() function

The strtolower() function returns string in lowercase letter.

Syntax
string strtolower (string $string)

Example

```
<?php
$str="My country is India";
$str=strtolower($str);
echo $str;
?>
```

Output:

```
my country is india
```

→strtoupper() function

The strtoupper() function returns string in uppercase letter.
Syntax
string strtoupper (string $string)

Example

```
<?php
$str="My country is India";
$str=strtoupper($str);
echo $str;
?>
```

Output:

```
MY COUNTRY IS INDIA
```

→ ucfirst() function

The ucfirst() function returns string converting first character into uppercase. It doesn't change the case of other characters.
Syntax
string ucfirst (string $str)

Example

```
<?php
$str=” my country is India”;
$str=ucfirst($str);
echo $str;
?>
```

```
Output:
My country is India
```

→ lcfirst() function

The lcfirst() function returns string converting first character into lowercase. It doesn't change the case of other characters.

Syntax

string lcfirst (string $str)

Example

```
<?php
$str=" My country is India";
$str=lcfirst($str);
echo $str;
?>
```

```
Output:
my country is India
```

→ucwords() function

The ucwords() function returns string converting first character of each word into uppercase.

Syntax

string ucwords (string $str)

Example

```
<?php
$str=" My country is India ";
$str=ucwords($str);
echo $str;
?>
```

```
Output:
My Country Is India
```

→strrev() function

The strrev() function returns reversed string.

Syntax

string strrev (string $string)

Example

```
<?php
$str=" My country is India ";
$str=strrev($str);
echo $str;
?>
```

Output:

```
aidnI si yrtnuoc yM
```

→ strlen() function

The strlen() function returns length of the string.

Syntax

int strlen (string $string)

Example

```
<?php
$str=" My country is India ";
$str=strlen($str);
echo $str;
?>
```

Output:

```
19
```

→strpos() function

The strpos() function is used to search for a string or character within a string.

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Syntax:

String Strpos(string $string1,string $string2)

Example:

```
<?php

Echo strops(“Hello world!”,”world”);

?>
```

Output:

6

→ str_word_count() function

The PHP str_word_count() function counts the number of words in a string:

Example:

```
<?php
echo str_word_count("Hello world!");
?>
```

Output:

2

→str_replace() function

The PHP str_replace() function replaces some characters with some other characters in a string.

The example below replaces the text "world" with "Dolly":

```
<?php
echo str_replace("world", "Dolly", "Hello world!");

?>
```

Output:

Hello Dolly!

→str_split() Function

The str_split() function splits a string into an array.

Syntax

str_split(*string,length*)

Example:

```
<?php

$string_name='Welcome to tact';

print_r(str_split($string_name));

echo '<br>';

print_r(str_split($string_name,4));

echo '<br>';

?>
```

Output

```
Array (    [0] => W    [1] => e     [2] => l     [3] => c     [4] => o     [5] => m     [6] => e
[7] =>      [8] => t     [9] => o     [10] =>      [11] => t     [12] =>a     [13] =>c     [14] =>t

Array (    [0] => Welc     [1] => ome      [2] => to t     [3] => act
```

→implode() function

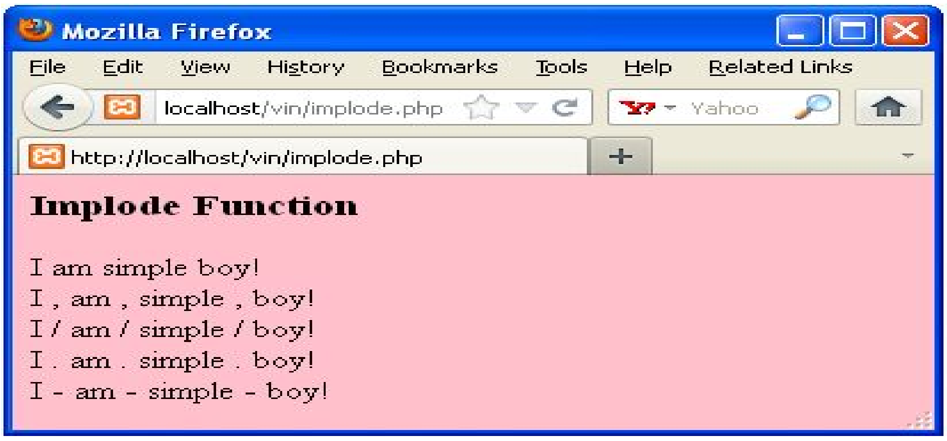
It is used to join array elements with a string

Syntax

implode(*separator,array*)

```
<?php
$arr = array ('I','am','simple','boy!');
$space_separated = implode(" ", $arr);
$comma_separated = implode(" , ", $arr);
$slash_separated = implode(" / ", $arr);
$dot_separated = implode(" . ", $arr);
$hyphen_separated = implode(" - ", $arr);
echo $space_separated.'<br>';
echo $comma_separated.'<br>';
echo $slash_separated.'<br>';
echo $dot_separated.'<br>';
echo $hyphen_separated;
?>
```

Output


Mozilla Firefox
File Edit View History Bookmarks Tools Help Related Links
localhost/vin/implode.php
Yahoo
http://localhost/vin/implode.php

```
Implode Function

I am simple boy!
I , am , simple , boy!
I / am / simple / boy!
I . am . simple . boy!
I - am - simple - boy!
```

→join() function

Join is the alias of implode().

Example:

```
<?php
   $array = array('1', '2', '3', '4', '5', '6', '7', '8', '9', '10');
   echo join('<br>', $array);
 ?>
```

Output

```
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10
```

-->explode() Function

The explode() function breaks a string into an array.

Note: The "separator" parameter cannot be an empty string.

Note: This function is binary-safe.

Syntax

explode(*separator,string,limit*)

Parameter	Description
separator	Required. Specifies where to break the string
string	Required. The string to split
limit	Optional. Specifies the number of array elements to return. Possible values: →Greater than 0 - Returns an array with a maximum of limit element(s) →Less than 0 - Returns an array except for the last -limit elements() →0 - Returns an array with one element

Example:

```
<?php
$str = 'one,two,three,four';
// zero limit
print_r(explode(',',$str,0));
// positive limit
print_r(explode(',',$str,2));
// negative limit
print_r(explode(',',$str,-1));
?>
```

Output

```
Array ( [0] => one,two,three,four )
Array ( [0] => one [1] => two,three,four )
Array ( [0] => one [1] => two [2] => three )
```

→htmlentities() Function

Convert some characters to HTML entities:

```
<?php
$str = '<a href="https://www.w3schools.com">Go to w3schools.com</a>';
echo htmlentities($str);
?>
```

The HTML output of the code above will be (View Source):

```
&lt;a href=&quot;https://www.w3schools.com&quot;&gt;Go to
w3schools.com&lt;/a&gt;
```

→substr() function

The substr() function returns a part of a string.

<u>Syntax:</u>

substr(*string,start,length*)

String	**Required.Specifies the string to return apart of**
Start	**Required.Specifies where to start in the string.**
	→A positive number-Start at a specified position in the string.
	→ A negative number - Start at a specified position from the end of the string
	→0 - Start at the first character in string
Length	**Optional.**Specifies the length of the returned string.Default string.
	A positive number- The length to be returned from the start parameter.

```
<?php
echo substr("Hello world",10)."<br>";
echo substr("Hello world",1)."<br>";
echo substr("Hello world",3)."<br>";
echo substr("Hello world",7)."<br>";
echo substr("Hello world",-1)."<br>";
echo substr("Hello world",-10)."<br>";
echo substr("Hello world",-8)."<br>";
echo substr("Hello world",-4)."<br>";
?>
```

Output

```
d
ello world
lo world
orld
d
ello world
lo world
orld
```

Example-2

```
<?php
// Positive numbers:
echo substr("Hello world",0,10)."<br>";
echo substr("Hello world",1,8)."<br>";
echo substr("Hello world",0,5)."<br>";
echo substr("Hello world",6,6)."<br>";

// Negative numbers:
echo substr("Hello world",0,-1)."<br>";
echo substr("Hello world",-10,-2)."<br>";
echo substr("Hello world",0,-6)."<br>";
?>
```

Output

```
Hello worl
ello wor
Hello
world
Hello worl
ello wor
Hello
```

→ trim() Function

Remove characters from both sides of a string.

Example:

```
<?php
$str = "Hello World!";
echo $str . "<br>";
echo trim($str,"Hed!");
?>
```

Output

```
Hello World!
llo Worl
```

→ strstr() Function

Find the first occurrence of a word in the string and return the rest of the string.

Example1:

```
<?php
echo strstr("Hello world!","world");
?>
```

Output

```
world
```

Example2:

```
<?php
echo strstr("Hello world!",111);
?>
```

Output

```
o world!
```

Example3:

Return the part of the string before the first occurence of "world":

```
<?php
echo strstr("Hello world!","world",true);
?>
```

Output

```
Hello
```

→str_repeat() Function

Repeat the string for multiple times.

Ex:

```
<?php
echo str_repeat("@",13);
?>
```

Output: @@@@@@@@@@@@@

→strcmp() Function

This function compares two strings.

Syntax:

Strcmp(string1,string2)

Parameter	**Description**
String1	**Required.Specifies the first string to compare.**
String2	**Required.Specifies the first string to compare.**
Return value	**This function returns**

→ 0 - if the two strings are equal

→ <0 - if string1 is less than string2

→ >0 - if string1 is greater than string2

Example:

```
<?php
echo strcmp("Hello","Hello");
echo "<br>";
echo strcmp("Hello","hELLo"
);
?>
```

Output

```
0
-1
```

--> strrchr() Function

Search a string for "What", and return all characters from this position to the end of the string:

```
<?php
echo strrchr("Hello world! What a beautiful day!",What);
?>
```

Output

What a beautiful day!

→ PHP filter_var() Function

The filter_var() function filters a variable with the specified filter.Returns the filtered data on success or FALSE on failure.

Syntax: Filter_var(variable,filter,options)

Parameter	Description
variable	Required. Specifies the variable to filter
filter	Optional. Specifies the ID of the filter to use. Default is FILTER_SANITIZE_STRING. Check the Complete PHP Filter Reference for possible filters A filter ID can be an ID name (like FILTER_VALIDATE_EMAIL) or an ID number (like 274)
options	Optional. Specifies an associative array of flags/options or a single flag/option. Check each filter for possible options and flags

Example

```
<?php
if(!filter_var("someone@example....com", FILTER_VALIDATE_EMAIL))
 {
 echo("E-mail is not valid");
 }
else
 {
 echo("E-mail is valid");
 }
?>
```

Output: E-mail is not valid

PHP Filter Functions

Function	Description
filter_id()	Returns the ID number of a specified filter
filter_input()	Get input from outside the script and filter it
filter_input_array()	Get multiple inputs from outside the script and filters them
filter_list()	Returns an array of all supported filters
filter_var_array()	Get multiple variables and filter them
filter_var()	Get a variable and filter it

PHP Filters

ID Name	Description
FILTER_CALLBACK	Call a user-defined function to filter data
FILTER_SANITIZE_STRING	Strip tags, optionally strip or encode special characters
FILTER_SANITIZE_STRIPPED	Alias of "string" filter
FILTER_SANITIZE_ENCODED	URL-encode string, optionally strip or encode special characters
FILTER_SANITIZE_SPECIAL_CHARS	HTML-escape '"<>& and characters with ASCII value less than 32
FILTER_SANITIZE_EMAIL	Remove all characters, except letters, digits and !#$%&'*+-/=?^_`{|}~@.[]
FILTER_SANITIZE_URL	Remove all characters, except letters, digits and $-_.+!*'(),{}|\\^~[]`<>#%";/?:@&=
FILTER_SANITIZE_NUMBER_INT	Remove all characters, except digits and +-
FILTER_SANITIZE_NUMBER_FLOAT	Remove all characters, except digits, +- and optionally .,eE
FILTER_SANITIZE_MAGIC_QUOTES	Apply addslashes()
FILTER_UNSAFE_RAW	Do nothing, optionally strip or encode special characters
FILTER_VALIDATE_INT	Validate value as integer, optionally from the specified range
FILTER_VALIDATE_BOOLEAN	Return TRUE for "1", "true", "on" and "yes", FALSE for "0", "false", "off", "no", and "", NULL otherwise
FILTER_VALIDATE_FLOAT	Validate value as float
FILTER_VALIDATE_REGEXP	Validate value against regexp, a Perl-compatible regular expression
FILTER_VALIDATE_URL	Validate value as URL, optionally with required components
FILTER_VALIDATE_EMAIL	Validate value as e-mail
FILTER_VALIDATE_IP	Validate value as IP address, optionally only IPv4 or IPv6 or not from private or reserved ranges

Format Strings

Every time an input is expected to be used and evaluated as part of the formatted string, it is preceded by a percent sign (%), followed by the specifiers/rules:

Note: *All specifiers, excluding the* ***type*** *specifier, are optional.*

- A **sign** specifier. Placing a plus sign (+) forces negative AND positive signs to be visible (only negative values are specified by default).
- A **padding** specifier. The default is a space, and does not need to be specified. A zero (0) can be used as well without any secondary notation. If any other character is to be used, it should be preceded with a single quote (‘).
- An **alignment** specifier. The default is right-justified (thus padding is placed on the left of the string). Placing a dash/subtract (–) will set it to left-justified.
- A **width** specifier. This integer determines the minimum length in characters the output should be. When combined with padding, the specified width minus the input’s length determines the number of padded characters that will be added.
- A **precision** specifier. A period (.) followed by an integer, sets the number of decimal places that should be output for a float. If used on a string, it sets a maximum character limit for the output.

A **type** specifier:

- *%* – a literal percent sign, thus would be written %% to display a percent sign in the formatting string
- *b* – the input should be an integer, a binary number is the output.
- *c* – the input should be an integer between 0-255, representing the ASCII byte-value. The character represented is output.
- *d* – the input should be an integer.
- *e* – the input is scientific notation.
- *u* – the input is an unsigned decimal number.
- *f* – the input is a float (locale aware).
- *F* – the input is a float (not locale aware).
- *o* – the input is an integer, an octal number is the output.
- *s* – the input is a string.
- *x* – the input is an integer, a hexadecimal number is the output (with lowercase letters).
- *X* – the input is an integer, a hexadecimal number is the output (with uppercase

letters).

Examples:

```
<?php
printf( "Australia comprises %d states and %d territories", 6, 10 );
printf( "Here's the number %s as a float (%f), a binary integer (%b), an octal integer
(%o), and a hex integer (%x).", 543.21, 543.21, 543.21, 543.21, 543.21 );
printf( "%d", 36 );                             // Displays "36"
printf( "%d", -36 );                            // Displays "-36"
printf( "%+d", 36 );                            / Displays "+36"
echo "<br/>";
printf( "%+d", -36 );                           // Displays "-36"
echo "<br/>";
printf( "%04d", 12 );                           // Displays "0012"
echo "<br/>";
printf( "%04d", 1234 );                         // Displays "1234"
echo "<br/>";
printf( "%05d", 12345 );                        // Displays "12345"
echo "<br/>";
printf( "%  10s", "Hello" );                    // Displays "     Hello"
echo "<br/>";
printf( "%'*10s", "Hello" );                    // Displays "*****Hello"
echo "<br/>";
printf( "%'*-10s", "Hello" );                   // Displays "Hello*****"
echo "<br/>";
printf( "%f", 123.456 );                        // Displays "123.456000"
echo "<br/>";
printf( "%.2f", 123.456 );                      // Displays "123.46"
printf( "%.10f", 123.456 );                     // Displays "123.4560000000"
echo "<br/>";
printf( "%.0f", 123.456 );                      // Displays "123"
printf( "%08.2f", 123.456 );
printf( "%.3s", "Hello" );                      // Displays "He"
?>
```

→Sprintf() Function

The sprintf() function writes a formatted string to a variable.The arg1, arg2, ++ parameters will be inserted at percent (%) signs in the main string. This function works "step-by-step". At the first % sign, arg1 is inserted, at the second % sign, arg2 is inserted, etc.

Example:

```
<?php
$number = 9;
$str = "Beijing";
$txt = sprintf("There are %u million bicycles in %s.",$number,$str);
echo $txt;
?>
```

→vsprintf() Function

The vsprintf() function writes a formatted string to a variable.Unlike sprintf(), the arguments in vsprintf(), are placed in an array. The array elements will be inserted at the percent (%) signs in the main string.

```
<?php
$number = 9;
$str = "Beijing";
$txt = vsprintf("There are %u million bicycles in %s.",array($number,$str));
echo $txt;
?>
```

→ vprintf Function

Operates as printf() but accepts an array of arguments, rather than a variable number of arguments. Display array values as a formatted string according to format (which is described in the documentation for sprintf()).

int vprintf (string $format , array $args)

Example

```
<?php
vprintf("%04d-%02d-%02d", explode('-', '1988-8-1')); // 1988-08-01
?>
```

→fprintf() Function

Write the output to a file.

```
<?php
$number = 9;
$str = "Beijing";
$file = fopen("test.txt","w");
echo vfprintf($file,"There are %u million bicycles in %s.",$number,$str);
?>
```

The following text will be written to the file "test.txt":

```
There are 9 million bicycles in Beijing.
```

→vfprintf() Function

Unlike fprintf(), the arguments in vfprintf(), are placed in an array. The array elements will be inserted at the percent (%) signs in the main string.

```
<?php
$number = 9;
$str = "Beijing";
$file = fopen("test.txt","w");
echo vfprintf($file,"There are %u million bicycles in %s.",array($number,$str));
?>
```

The following text will be written to the file "test.txt":

```
There are 9 million bicycles in Beijing.
```

Find the position of a substring

```
<?php
$str='i love ind india';
$s='india';
$i = $m = $c =$flag=0;
        while ( !empty($str[$c]))
        {
                if ( $str[$m] == $s[$i] )
                {

                                $i++;
                                $m++;

                        if ( empty($s[$i]))               //Find occurance
                        {
                                echo $s.' is present in '.($c+1).' location<br>';
                                $flag=1;
                                $i=0;
                                $c=$m;
                        }
                }
                else              //... mismatch
                {

                        $c++;
                        $m = $c;
                        $i=0;
                }
        }
        if($flag==0)
                echo $s.' is not present';
?>
```

Join the strings

```
<?php
$str1="Welcome";
$str2="TACT";
for($i=0;$i<strlen($str1);$i++);
for($j=0;$j<strlen($str2);$j++)
{
$str1[$i]=$str2[$j];
$i++;
}
echo $str1;
?>
```

Split the string

```
<?php
$str = "this is a test string ";
$ar =array();
$tmp = "";
$j=0;
for($i=0; $i<strlen($str); $i++)
{
            if ($str[$i] != ' ')
             {
                        $tmp[$j]=$str[$i];
                        $j++;
                        continue;
              }
    $ar[]=$tmp;
    $tmp = "";
}
echo "Words are<br>";
 foreach ($ar as $val)
        echo $val."<br/>";
?>
```

Replacing the string with another string

```
<?php
$str='i love ind indian';

$s='ind';

$r='odisha';

$ans="";

        $i = $m = $c = $j = 0;

        while ( !empty($str[$c]))

        {

                if ( $str[$m] == $s[$i] ) // ...... matching

                {

                                $i++;
                                $m++;
                        if ( empty($s[$i])) //.....found occ
                        {
                                echo $s.' is present in '.($c+1).' location<br>';
                                $flag=1;                //.... copy replace string in ans string
 .....
                                for($k=0; !empty($r[$k]);$k++,$j++)
                                $ans[$j] = $r[$k];
                                $i=0;
                                $c=$m;
                        }
                }
                else //... mismatch

                {

                        $ans[$j] = $str[$c];

                        $j++;

                        $c++;

                        $m = $c;

                        $i=0;

                }

        }

echo $ans;

if($flag==0)

                echo $s.' is not present';

?>
```

PHP Regular Expressions: Preg_match, Preg_split, Preg_replace

What is a Regular Expressions?

Regular expressions are powerful pattern matching algorithm that can be performed in a single expression.Regular expressions use arithmetic operators such as (+,-,^) to create complex expressions. Regular expressions help you accomplish tasks such as validating email addresses, IP address etc.

Why to use regular expressions

- Regular expressions simplify identifying patterns in string data by calling a single function. This saves us coding time.
- When validating user input such as email address, domain names, telephone numbers, IP addresses,
- Highlighting keywords in search results
- When creating a custom HTML template. Regular expressions can be used to identify the template.

Regular expressions in PHP

PHP has built in functions that allow us to work with regular functions.

- preg_match – this function is used to perform a pattern match on a string. It returns true if a match is found and false if a match is not found.
- preg_split – this function is used to perform a pattern match on a string and then split the results into a numeric array
- preg_replace – this function is used to perform a pattern match on a string and then replace the match with the specified text.

Syntax:

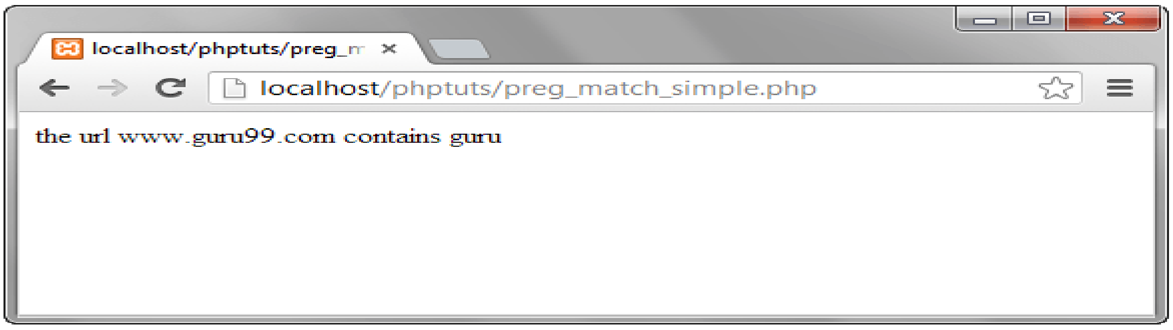
```
<?php
function_name('/pattern/',subject);
?>
```

- "function_name(...)" is either preg_match, preg_split or preg_replace.
- "/.../" The forward slashes denote the beginning and end of our regular expression
- "'/pattern/'" is the pattern that we need to matched
- "subject" is the text string to be matched against

→PHP Preg_match

The preg_match function to perform a simple pattern match for the word guru in a given URL.

```
<?php
$my_url = "www.guru99.com";
if (preg_match("/guru/", $my_url))
        echo "the url $my_url contains guru";
else
        echo "the url $my_url does not contain guru";
?>
```



```
the url www.guru99.com contains guru
```

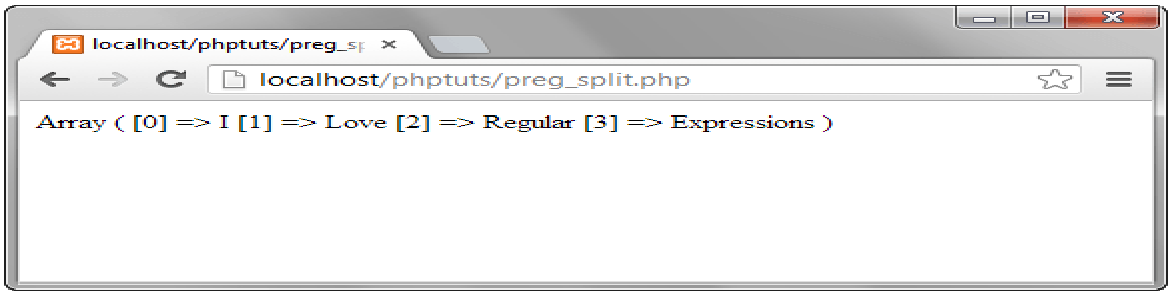
HERE,

- "preg_match(...)" is the PHP regular expression function
- "'/guru/'" is the regular expression pattern to be matched
- "$my_url" is the variable containing the text to be matched against.

→PHP Preg_split

The code below illustrates the implementation of preg_split().

```
<?php
$my_text="I Love Regular Expressions";
$my_array  = preg_split("/ /", $my_text);
print_r($my_array );
?>
```

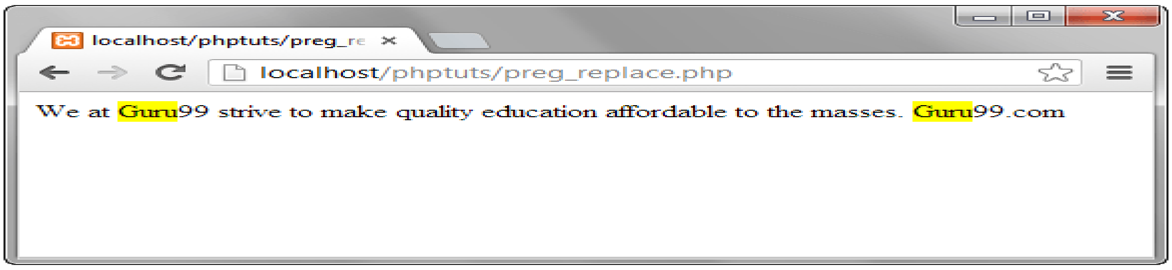


```
Array ( [0] => I [1] => Love [2] => Regular [3] => Expressions )
```

➔PHP Preg_replace

preg_replace function performs a pattern match and then replaces the pattern with something else.The code below searches for the word guru in a string.It replaces the word guru with the word guru surrounded by css code that highlights the background colour.

```
<?php
$text = "We at Guru99 strive to make quality education affordable to the masses. Guru99.com";
$text = preg_replace("/Guru/", '<span style="background:yellow">Guru</span>', $text);
echo $text;
?>
```



```
We at Guru99 strive to make quality education affordable to the masses. Guru99.com
```

Meta characters

Metacharacter	Description	Example
.	Matches any single character except a new line	/./ matches anything that has a single character
^	Matches the beginning of a string / excludes characters	/^PH/ matches any string that starts with PH
$	Matches pattern at the end of the string	/com$/ matches guru99.com,yahoo.com Etc
*	Matches any zero (0) or	/com*/ matches computer,

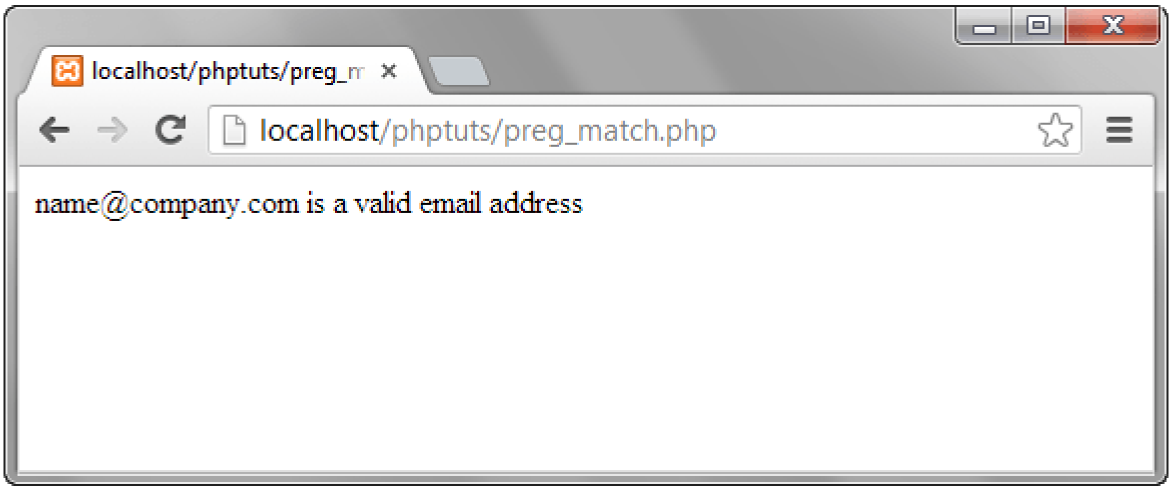
	more characters	communication etc.
+	Requires preceding character(s) appear at least once	/yah+oo/ matches yahoo
\	Used to escape meta characters	/yahoo+\.com/ treats the dot as a literal value
[...]	Character class	/[abc]/ matches abc
a-z	Matches lower case letters	/a-z/ matches cool, happy etc.
A-Z	Matches upper case letters	/A-Z/ matches WHAT, HOW, WHY etc.
0-9	Matches any number between 0 and 9	/0-4/ matches 0,1,2,3,4

```
<?php
$my_email = "name@company.com";
if (preg_match("/^[a-zA-Z0-9._-]+@[a-zA-Z0-9-]+.[a-zA-Z.]{2,5}$/", $my_email))
        echo "$my_email is a valid email address";
else
        echo "$my_email is NOT a valid email address";
?>
```

HERE,

- "'/.../'" starts and ends the regular expression
- "^[a-zA-Z0-9._-]" matches any lower or upper case letters, numbers between 0 and 9 and dots, underscores or dashes.
- "+@[a-zA-Z0-9-]" matches the @ symbol followed by lower or upper case letters, numbers between 0 and 9 or dashes.
- "+\.[a-zA-Z.]{2,5}$/" escapes the dot using the backslash then matches any lower or upper case letters with a character length between 2 and 5 at the end of the string.

Browse to the URL **http://localhost/phptuts/preg_match.php**



```
name@company.com is a valid email address
```

HTML WITH PHP

HTML Form with PHP

- Forms are used to get input from the user and submit it to the web server for processing.A form is an HTML tag that contains graphical user interface items such as input box, check boxes radio buttons etc.
- The form is defined using the <form>...</form> tags and GUI items are defined using form elements such as <input>.The <input> element can be displayed in several ways, depending on the type attribute.

  Syntax:

  HTML Form Tags

Tag	**Description**
<form>	It defines an HTML form to enter inputs by the user side.
<input>	It defines an input control.
<textarea>	It defines a multi-line input control.
<label>	It defines a label for an input element.
<fieldset>	It groups the related element in a form.
<legend>	It defines a caption for a <fieldset> element.
<select>	It defines a drop-down list.
<optgroup>	It defines a group of related options in a drop-down list.
<option>	It defines an option in a drop-down list.
<button>	It defines a clickable button.

HTML Input Tags

Type	**Description**
<input type=”text”>	Defines a one line text input field
<input type=”radio”>	Defines a radio button(For selecting one from multiple options)
<input type=”checkbox”>	Defines a checkbox(For selecting multiple options)
<input type=”submit”>	Defines a submit button.(For submitting the form)

→The Action Attribute

The **action** attribute defines the action to be performed when the form is submitted. Normally, the form data is sent to a web page on the server when the user clicks on the submit button. If the action attribute is omitted, the action is set to the current page.
<form **action="/action_page.php**">

→The Target Attribute

The **target** attribute specifies if the submitted result will open in a new browser tab, a frame, or in the current window.The default value is "**_self**" which means the form will be submitted in the current window.To make the form result open in a new browser tab, use the value "**_blank**": **<form action="/action_page.php"** target="_blank"**>**

→The Method Attribute

The **method** attribute specifies the HTTP method (**GET** or **POST**) to be used when submitting the form data:

Ex: <form action="/action_page.php" **method="get"**> or
<form action="/action_page.php" **method="post"**>

Here is the list of <form> attributes:

Attribute	**Description**
accept-charset	Specifies the charset used in the submitted form (default: the page charset)
action	Specifies an address (url) where to submit the form (default: the submitting page).
autocomplete	Specifies if the browser should autocomplete the form (default: on).
enctype	Specifies the encoding of the submitted data (default: is url-encoded).
method	Specifies the HTTP method used when submitting the form (default: GET).
name	Specifies a name used to identify the form .
novalidate	Specifies that the browser should not validate the form.

POST	GET
Values not visible in the URL	Values visible in the URL
Has not limitation of the length of the values since they are submitted via the body of HTTP	Has limitation on the length of the values usually 255 characters. This is because the values are displayed in the URL. Note the upper limit of the characters is dependent on the browser.
Has lower performance compared to Php_GET method due to time spent encapsulation the Php_POST values in the HTTP body	Has high performance compared to POST method dues to the simple nature of appending the values in the URL.
Supports many different data types such as string, numeric, binary etc.	Supports only string data types because the values are displayed in the URL
Results cannot be book marked	Results can be book marked due to the visibility of the values in the URL

target	Specifies the target of the address in the action attribute (default: _self).

Capturing Form Data with PHP

To access the value of a particular form field, you can use the following superglobal variables. These variables are available in all scopes throughout a script.

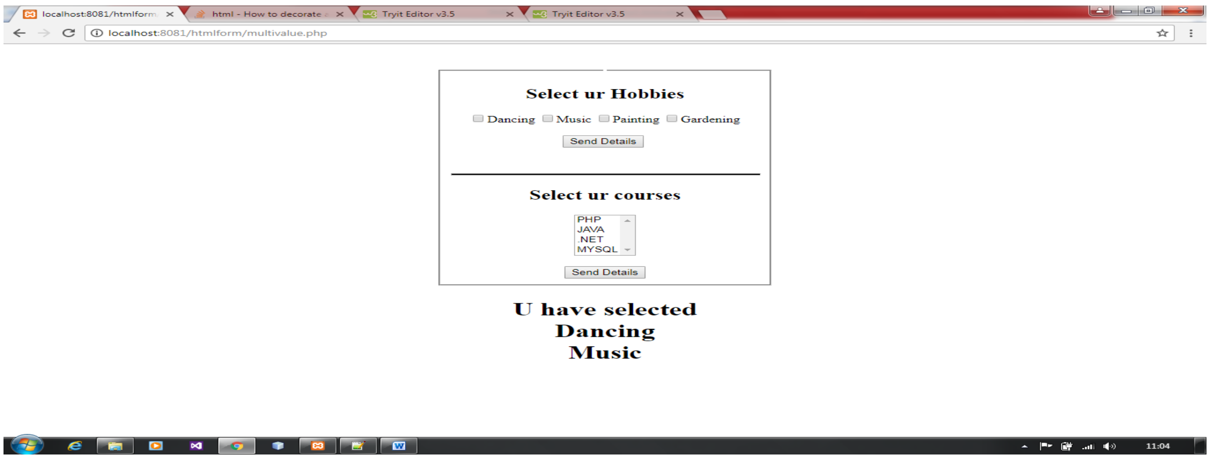
Superglobal	**Description**
$_GET	Contains a list of all the field names and values sent by a form using the get method (i.e. via the URL parameters).
$_POST	Contains a list of all the field names and values sent by a form using the post method (data will not visible in the URL).
$_REQUEST	Contains the values of both the $_GET and $_POST variables as well as the values of the $_COOKIE superglobal variable.

PHP Multi-Values Form Fields

When you submit those fields to the web server, those form fields send multiple values, rather than a single value. So how do you handle these multi-values fields in PHP? It’s kind of tricky. You need to add square brackets ([]) after the field name of the multi-valued form field. When PHP sees this sign ([]), it creates a nested array of values within the $_GET or $_POST array based on the method you use in the form.

```
<html>
<head>
</head>
<body>
<center>
<br>
<br>
<form method="post">
<center>
<fieldset style="width:350px" style="height:300px">
<legend></legend>
<label><h2>Select ur Hobbies</h2></label>
<input type="checkbox" name="chk[]" value="Dancing">Dancing
<input type="checkbox" name="chk[]" value="Music">Music
<input type="checkbox" name="chk[]" value="Painting">Painting
<input type="checkbox" name="chk[]" value="Gardening">Gardening<br><br>
<input type="submit" name="submit" id="submitButton" value="Send Details" /> <br/><br/><br/>
<hr style = 'background-color:#000000; height:2px;width:100%;' />
<label><h2>Select ur courses</h2></label>
<select name="courses[]" multiple="multiple" >
<option value="PHP">PHP</OPTION>
<option value="JAVA">JAVA</option>
<option value=".NET">.NET</option>
<option value="MYSQL">MYSQL</option>
</select><br/><br/>
<input type="submit" name="submit1"  value="Send Details" />
```

```
</fieldset>
</center>
</form>
</center>
<?php
if(isset($_POST["submit"])) //Checks if the send button is pressed
{
                    if(isset($_POST["chk"])) //checks if any interest is checked
               {
                         echo "<center>";
                         echo "<h1>U have selected<br>";
                foreach($_POST["chk"] as $value) //Iterate the interest array and get the values
                         echo  $value."</br>";  //print the values
                         echo "</h1></center>";
              }
}
if(isset($_POST["submit1"])) //Checks if the send button is pressed
{
          if(isset($_POST["courses"])) //checks if any interest is checked
    {
                         echo "<center>";
                         echo "<h1>U have selected<br>";
                            foreach($_POST["courses"] as $value) //Iterate the interest array and get the
values
        {
            echo  $value."</br>";  //print the values
        }
                         echo "</h1></center>";
    }
}
?>
</body>
</html>
```



localhost:8081/htmlform/multivalue.php

Select ur Hobbies

Dancing Music Painting Gardening

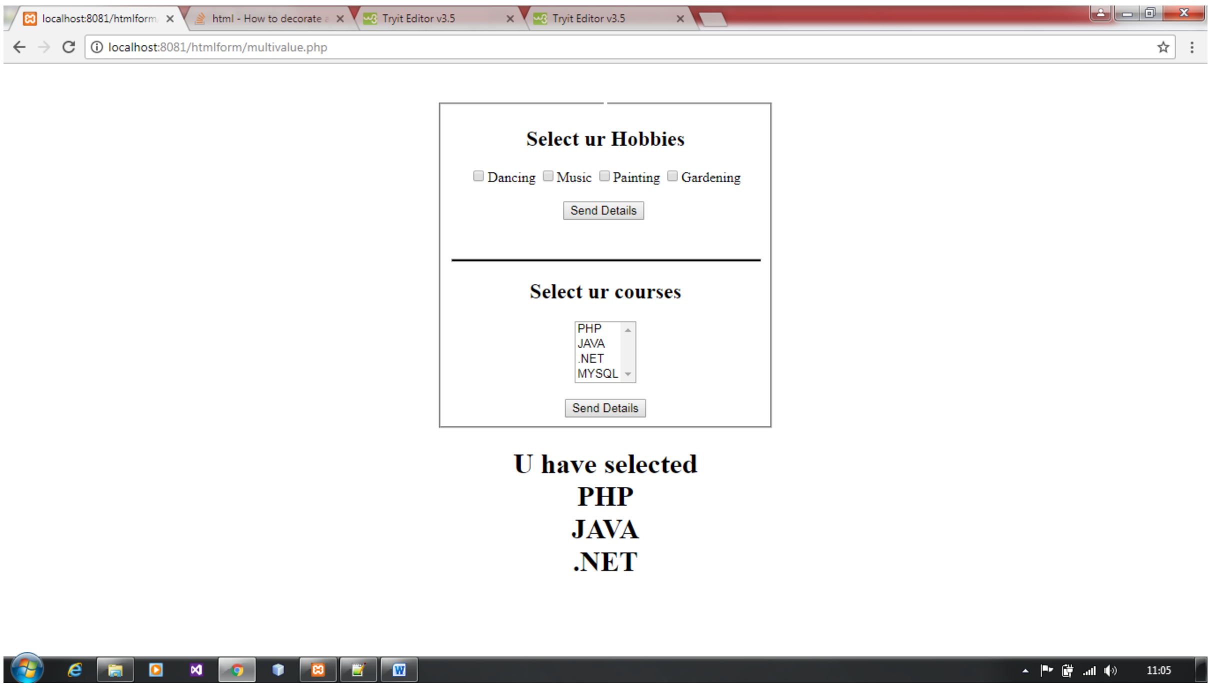
Send Details

Select ur courses

PHP
JAVA
.NET
MYSQL

Send Details

U have selected
Dancing
Music



localhost:8081/htmlform/multivalue.php

Select ur Hobbies

Dancing Music Painting Gardening

Send Details

Select ur courses

PHP
JAVA
.NET
MYSQL

Send Details

U have selected
PHP
JAVA
.NET

PHP: Redirect To a form After Form Submission

In PHP, redirection is done by using header() function as it is considered to be the fastest method to redirect traffic from one web page to another. The main advantage of this method is that it can navigate from one location toanother without the user having to click on a link or button.

Example

Userdata.php

```
<html>
<body>
<form action="phptext.php" method="GET">
Name:<input type="text" name="n1"><br>
E-mail:<input type="text" name="e1"><br>
<input type="submit" name="sub">
</form>
</body>
</html>
```

Phptext.php

```
<?php
if(isset($_POST["sub"]))
{
echo "Name :".$_POST["n1"]."<br/>";
echo "Email:".$_POST["e1"]."<br/>";
}
?>
```