# Unit – 5
# PHP

Prepared by:
Dr. Susheela Dahiya

# Content

- Introduction to PHP
- Features of PHP
- Syntax of PHP
- Comments in PHP
- Case Sensitivity in PHP
- PHP Variables
- Variables Scope
  - *Local Scope*
  - *Global Scope*
  - *Static Scope*
- PHP Echo and Print Statement
- PHP Data Types
  - Scalar
  - Compound
  - Special
- PHP Constants
- PHP Operators
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Increment/Decrement operators
  - Logical operators
  - String operators
  - Array operators
  - Bitwise Operators

# PHP

- PHP stands for HyperText Preprocessor.
- PHP is an interpreted language, i.e. there is no need for compilation.
- PHP is a server side scripting language.
- PHP is faster than other scripting language e.g. asp and jsp.

# PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

# PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data
- With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

# PHP Features

There are given many features of PHP.

- **Performance**: Script written in PHP executes much faster then those scripts written in other languages such as JSP & ASP.
- **Open Source Software**: PHP source code is free available on the web, you can developed all the version of PHP according to your requirement without paying any cost.
- **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.

# Install PHP

To install PHP, we will suggest you to install AMP (Apache, MySQL, PHP) software stack. It is available for all operating systems. 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, OpenSSL, Mercury Mail etc.
- If you are on Windows and don't want Perl and other features of XAMPP, you should go for WAMP. In a similar way, you may use LAMP for Linux and MAMP for Macintosh.

- **XAMPP Download Link: https://www.apachefriends.org/download.html**

# Difference between Client and Server side Scripting

| Client-side scripting | Server-side scripting |
|---|---|
| Source code is visible to the user. | Source code is not visible to the user because its output of server-side is an HTML page. |
| Its main function is to provide the requested output to the end user. | Its primary function is to manipulate and provide access to the respective database as per the request. |
| It usually depends on the browser and its version. | In this any server-side technology can be used and it does not depend on the client. |
| It runs on the user's computer. | It runs on the webserver. |
| There are many advantages linked with this like faster. response times, a more interactive application. | The primary advantage is its ability to highly customize, response requirements, access rights based on user. |
| It does not provide security for data. | It provides more security for data. |
| HTML, CSS, and javascript are used. | PHP, Python, Java, Ruby are used. |
| No need of interaction with the server. | It surge the processing load on the server. |

# PHP Syntax

- A PHP script can be placed anywhere in the document.
- A PHP script starts with **<?php** and ends with **?>**:
- The default file extension for PHP files is ".php".
- A PHP file normally contains HTML tags, and some PHP scripting code.

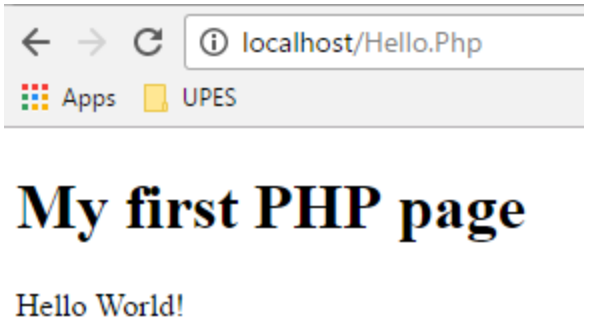
# A Simple Program

- <!DOCTYPE html>
  <html>
  <body>

  <h1>My first PHP page</h1>

  **<?php**
  **echo "Hello World!";**
  **?>**

  </body>
  </html>



# Comments in PHP

- A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.
- <!DOCTYPE html>
  <html>
  <body>

  <?php
  **// This is a single-line comment**

  **# This is also a single-line comment**

  **/***
  **This is a multiple-lines comment block**
  **that spans over multiple**
  **lines**
  ***/**

  // You can also use comments to leave out parts of a code line
  $x = 5 **/* + 15 */** + 5;
  echo $x;
  ?>

  </body>
  </html>

10

# Case Sensitivity

- <!DOCTYPE html>
  <html>
  <body>

  <?php
  **ECHO** "Hello World!<br>";
  **echo** "Hello World!<br>";
  **EcHo** "Hello World!<br>";
  ?>

  </body>
  </html>

Hello World!
Hello World!
Hello World!

- In PHP, **all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions** are NOT case-sensitive.

# PHP Variables

- A variable in PHP is a name of memory location that holds data.
- A variable is a temporary storage that is used to store data temporarily.
- In PHP, a variable is declared using $ sign followed by variable name.

$variablename=value;

```php
<?php
$str="hello string";
$x=200;
$y=44.6;
echo "string is: $str <br/>";
echo "integer is: $x <br/>";
echo "float is: $y <br/>";
?>
```

```
string is: hello string
integer is: 200
float is: 44.6
```

All variable names are case-sensitive.

- <!DOCTYPE html>
  <html>
  <body>

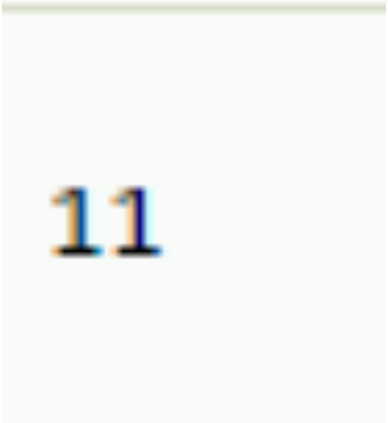
  <?php
  $color = "red";
  echo "My car is " . $color . "<br>";
  echo "My house is " . $COLOR . "<br>";
  echo "My boat is " . $coLOR . "<br>";
  ?>

  </body>
  </html>

My car is red
My house is
My boat is

# Sum of Two Variables

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



- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

**Rules for PHP variables:**

- A variable starts with the $ 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 can only contain alpha-numeric characters and underscores (A-z, 0-9, and _ )
- Variable names are case-sensitive ($age and $AGE are two different variables)

Note: PHP is a **loosely typed language**, it means PHP automatically converts the variable to its correct data type.

```
<?php
$a="hello";//letter (valid)
$_b="hello";//underscore (valid)
$4c="hello";//number (invalid)
$*d="hello";//special symbol (invalid)

echo "$a <br/> $_b";
echo "$4c <br/> $*d";

?>
```

# PHP Variables Scope

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- ❑local
- ❑global
- ❑static

# Global Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

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

<?php

$x = 5; // global scope

function myTest()
{
   // using x inside this function will generate an error
   echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>
```

Variable x inside function is:

Variable x outside function is: 5

# Local Scope

- A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

Variable x inside function is: 5

Variable x outside function is:

- <!DOCTYPE html>
  <html>
  <body>

  <?php
  function myTest() {
  **$x = 5; // local scope**
  echo " Variable x inside function is: $x ";
  }
  myTest();

  // using x outside the function will generate an error
  **echo " Variable x outside function is: $x ";**
  ?>
  </body>
  </html>

# The Global Keyword

- The global keyword is used to access a global variable from within a function
- <!DOCTYPE html>
  <html>
  <body>

  <?php
  $x = 5;
  $y = 10;

  function myTest()
 {
  **global $x, $y;**
  $y = $x + $y;
}

myTest();  // run function
**echo $y;** // output the new value for variable $y
?>

</body>
</html> within a function

15

# The static Keyword

- Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted.
- Each time the function is called, that variable will still have the information it contained from the last time the function was called.
- **Note:** The variable is still local to the function.

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

<?php
function myTest()
{
    static $x = 0;
    echo $x;
    $x++;
}

myTest();
echo "<br>";
myTest();
echo "<br>";
myTest();
?>

</body>
</html>
```

0
1
2

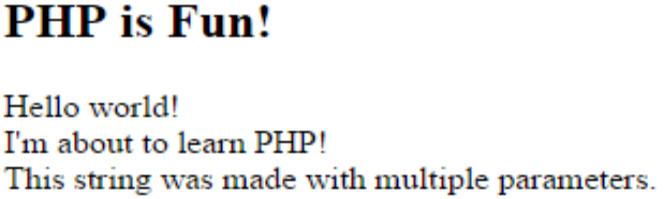
# PHP Echo and Print

In PHP there are two basic ways to get output: **echo and print.**

- echo and print are more or less the same. They are both used to output data to the screen.
- The differences are small:
  - echo has no return value while print has a return value of 1 so it can be used in expressions.
  - echo can take multiple parameters (although such usage is rare) while print can take one argument.
  - echo is marginally faster than print.

# PHP echo Statement

- The echo statement can be used with or without parentheses: echo or echo() [in case when passing multiple parameters]
- <!DOCTYPE html>
  <html>
  <body>
  <?php
  echo "<h2>PHP is Fun!</h2>";
  echo "Hello world!<br>";
  echo "I'm about to learn PHP!<br>";
  echo "This ", "string ", "was ", "made ", "with multiple parameters.";
  ?>
  </body>
  </html>

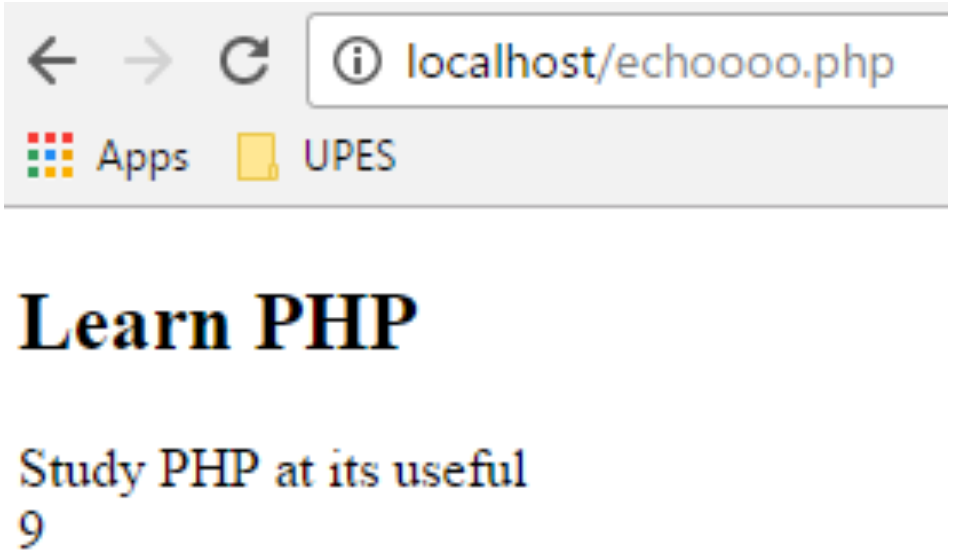


```
<!DOCTYPE html>
<html>
<body>

<?php
$txt1 = "Learn PHP";
$txt2 = "its useful";
$x = 5;
$y = 4;

echo "<h2>" . $txt1 . "</h2>";
echo "Study PHP at " . $txt2 . "<br>";
echo $x + $y;
?>

</body>
</html>
```



# PHP Print Statement

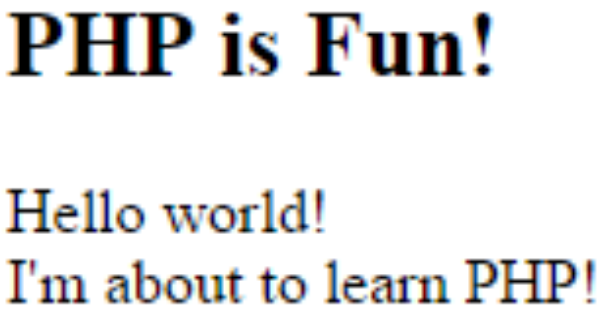
- The print statement can be used with or without parentheses: print or print().

**Output String**

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

<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>

</body>
</html>
```



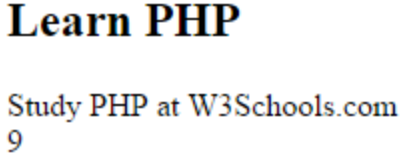
- **Output Variables and text**

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

<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

print "<h2>" . $txt1 . "</h2>";
print "Study PHP at " . $txt2 . "<br>";
print $x + $y;
?>

</body>
</html>
```



# Difference

echo can take multiple parameters (although such usage is rare) while print can take one argument

```
<?php

 $name = "Ravi ";
 $profile = "PHP Developer";
 $age = 25;

echo $name , $profile , $age, " years old";

?>
```

**Output:** Ravi PHP Developer25 years old

```
<?php

$name = "Ravi ";
 $profile = "PHP Developer";
 $age = 25;

print $name , $profile , $age, " years old";

?>
```

**Output** : Parse error: syntax error

# Difference

- echo has no return value while print has a return value of 1 so it can be used in expressions.

```
<?php

$name = "Ravi ";
$ret =  echo $name;

?>
```

**Output**

Parse error: syntax error, unexpected

```
<?php

$name = "Ravi ";
$ret =  print $name;

//To test it returns or not
echo $ret;

?>
```

**Output**

Ravi

# 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:
  - **Scalar Types:** Boolean, integer, float, string
  - **Compound Types:** array, object
  - **Special Types:** resource, NULL
- The PHP var_dump() function returns the data type and value

# PHP String

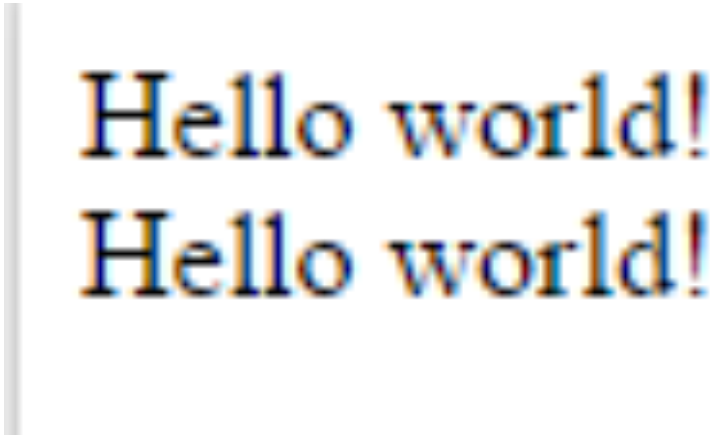
- A string is a sequence of characters, like "Hello world!". A string can be any text inside quotes. You can use single or double quotes:

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

  <?php
  $x = "Hello world!";
  $y = 'Hello world!';

  echo $x;
  echo "<br>";
  echo $y;
  ?>

  </body>
  </html>
  ```


  ```

# PHP Integer

- An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

**Rules for integers:**

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

- In the following example $x is an integer.

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

<?php
$x = 528;
$y= 0XFFFF;
$z= 0777;
var_dump($x);
 var_dump($y);
 var_dump($z);
?>

</body>
</html>
```

**Output:**

```
int(528)
int(4095)
int(511)
```

# PHP Float

- A float (floating point number) is a number with a decimal point or a number in exponential form.
- In the following example $x is a float.

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 528.20;
$y= 0.000000000000002;
echo var_dump($x);
echo "<br>";
echo var_dump($y);
?>
</body>
</html>
```

**Output:**

**float(528.2)**
**float(2.0E-14)**

# PHP Boolean

- A Boolean represents two possible states: TRUE or FALSE.
- $x = true;
  $y = false;
- Booleans are often used in conditional testing.

# PHP Array

- An array stores multiple values in one single variable.
- In the following example $cars is an array. The PHP var_dump() function returns the data type and value:
- <!DOCTYPE html>
  <html>
  <body>

  <?php
  **$cars = array("Volvo","BMW","Toyota");**
  var_dump($cars);
  ?>

  </body>
  </html>

array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }

- Arrays can be categorized as:
  - **Indexed arrays** - Arrays with a numeric index
  - **Associative arrays** - Arrays with named keys
  - **Multidimensional arrays** - Arrays containing one or more arrays

# Indexed Array

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

<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>

</body>
</html>
```

**Output:**

I like Volvo, BMW and Toyota.

# Associative Array

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

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>

</body>
</html>
```

**Output:**

Peter is 35 years old.

# Multidimensional Array

- 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.
- PHP understands multidimensional arrays that are two, three, four, five, or more levels deep.
- **The dimension of an array indicates the number of indices you need to select an element.**
- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

# Two-dimensional Arrays

- A two-dimensional array is an array of arrays.

| Name | Stock | Sold |
|---|---|---|
| Volvo | 22 | 18 |
| BMW | 15 | 13 |
| Saab | 5 | 2 |
| Land Rover | 17 | 15 |

- $cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );

```php
<?php
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );

echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
```

**Output:**

Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15.

# Program

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

  <?php
  $cars = array
    (
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Saab",5,2),
    array("Land Rover",17,15)
    );

  for ($row = 0; $row < 4; $row++) {
    echo " Row number $row ";
    echo "<ul>";
    for ($col = 0; $col < 3; $col++) {
      echo "<li>".$cars[$row][$col]."</li>";
    }
    echo "</ul>";
  }
  ?>

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

# Output

**Row number 0**
- Volvo
- 22
- 18

**Row number 1**
- BMW
- 15
- 13

**Row number 2**
- Saab
- 5
- 2

**Row number 3**
- Land Rover
- 17
- 15

# Accessing Multidimensional Array

```
<?php
// Multidimensional array
$superheroes = array(
    "First-man" => array(
        "name" => "Peter",
        "email" => "peter@mail.com",
    ),
    "second-man" => array(
        "name" => "Clark",
        "email" => "clark@mail.com",
    ),
    "third-man" => array(
        "name" => "Harry",
        "email" => "harry@mail.com",
    )
);
```

```
// Printing all the keys and values one by one

$keys = array_keys($superheroes);
for($i = 0; $i < count($superheroes); $i++) {
    echo $keys[$i] . "{<br>";
    foreach($superheroes[$keys[$i]] as $key => $value) {
        echo $key . " : " . $value . "<br>";
    }
    echo "}<br>";
}
?>
```

- **Output:**

First-man{
name : Peter
email : peter@mail.com
}
second-man{
name : Clark
email : clark@mail.com
}
third-man{
name : HarryS
email : harry@mail.com
}

# PHP Object

- An object is a data type **which stores data and information on how to process that data**.
- In PHP, an object must be explicitly declared.
- First we must declare a class of object. For this, we use the **class keyword**. A class is a structure that can contain properties and methods
- In a class, variables are called properties and functions are called methods

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

<?php

class Car
 {
   function Car()
   {
       $this->model = "VW";

      /*The $this keyword refers to the current object, and is only available inside
methods */


    }
}
// create an object
$Neeraj = new Car();

// show object properties
echo $Neeraj->model;
?>

</body>
</html>
```

VW

# PHP NULL Value

- Null is a special data type which can have only one value: NULL.
- A variable of data type NULL is a variable that has no value assigned to it.
- If a variable is created without a value, it is automatically assigned a value of NULL.
- Variables can also be emptied by setting the value to NULL

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

<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>

</body>
</html>
```

NULL

# PHP Resource

- The special resource type is not an actual data type. It is the **storing of a reference to functions and resources external to PHP.**
- A common example of using the resource data type is a database call.

# PHP Constants

- Constants are like variables except that once they are defined **they cannot be changed or undefined.**
- A PHP constant is an identifier (name) for a simple value. The value cannot be changed during the script.
- A valid constant name starts with a **letter or underscore** (**no $ sign** before the constant name).
- Unlike variables, **constants are automatically global across the entire script.**

# Create a PHP Constant

- To create a constant, **use the define() function.**
- Syntax

  **define(*name, value, case-insensitive*)**
- Parameters:
  - *name*: Specifies the name of the constant
  - *value*: Specifies the value of the constant
  - *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

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

  <?php
  // case-insensitive constant name
  define("GREETING", "Welcome to Google.com!",
  true);
  echo greeting;
  ?>

  </body>
  </html>
  ```

**Output:**

Welcome to Google.com!
  ```

# Constants are Global

- They can be used across the whole script.
- <!DOCTYPE html>
  <html>
  <body>

  <?php
  define("GREETING", "Welcome to google.com!");

  function myTest()
  {
  **echo GREETING**;
  }

  myTest();
?>

</body>
</html>

**Output:**
Welcome to google.com!

# PHP Operators

- Operators are used to perform operations on variables and values.

- PHP divides the operators in the following groups:
  - Arithmetic operators
  - Assignment operators
  - Comparison operators
  - Increment/Decrement operators
  - Logical operators
  - String operators
  - Array operators
  - Bitwise Operators

# Arithmetic Operators

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

```
<!DOCTYPE html>
<html>
<body>
<?php
$x=10;
$y=5;
echo $x + $y; //15
echo"<br>";
echo $x - $y; //5
echo"<br>";
echo $x * $y;//50
echo"<br>";
echo $x / $y;//2
echo"<br>";
echo $x %$y;//0
echo"<br>";
echo $x**$y;//100000
echo"<br>";
?>
</body>
</html>
```

# Assignment Operators

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

# PHP Comparison Operators

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

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 100;
$y = "100";

var_dump($x == $y); // returns true because values are equal    bool(true)
var_dump($x === $y); // returns false because types are not equal    bool(false)
var_dump($x != $y); // returns false because values are equal    bool(false)
var_dump($x <> $y); // returns false because values are equal    bool(false)
var_dump($x !== $y); // returns true because types are not equal  bool(true)
var_dump($x > $y); // returns true because $x is greater than $y  bool(false)
var_dump($x < $y); // returns true because $x is less than $y    bool(false)
var_dump($x >= $y); // returns true because $x is greater than or equal to $y  bool(true)
var_dump($x <= $y); // returns true because $x is less than or equal to $y     bool(true)
?>
</body>
</html>
```

# PHP Increment / Decrement Operators

| Operator | Name | Description |
|---|---|---|
| ++$x | Pre-increment | Increments $x by one, then returns $x |
| $x++ | Post-increment | Returns $x, then increments $x by one |
| --$x | Pre-decrement | Decrements $x by one, then returns $x |
| $x-- | Post-decrement | Returns $x, then decrements $x by 1 |

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

<?php
$x = 10;
echo ++$x;   //11
echo"<br>";
echo $x++;   //11
echo"<br>";
echo $x;   //12
echo"<br>";
echo --$x;  //11
echo"<br>";
echo $x--;  //11
echo"<br>";
echo $x;  //10
echo"<br>";
?>

</body>
</html>
```

# PHP Logical Operators

| Operator | Name | Example | Result |
|---|---|---|---|
| and | And | $x and $y | True if both $x and $y are true |
| or | Or | $x or $y | True if either $x or $y is true |
| xor | Xor | $x xor $y | True if either $x or $y is true, but not both |
| && | And | $x && $y | True if both $x and $y are true |
| \|\| | Or | $x \|\| $y | True if either $x or $y is true |
| ! | Not | !$x | True if $x is not true |

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

<?php
$x = 100;
$y = 50;

if ($x == 100 and $y == 50)
{
   echo "Hello world!";
   echo"<br>";
}
if ($x == 100 or $y == 80)
{
echo "Hello world!";
echo"<br>";
}
if ($x == 100  xor $y == 80) {
echo "Hello world!";
echo"<br>";
}
if ($x == 100 && $y == 50) {
echo "Hello world!";
echo"<br>";
}
```

```
if ($x == 100 || $y == 80) {
echo "Hello world!";
echo"<br>";
}
if ($x !== 90) {
echo "Hello world!";
echo"<br>";
}
?>

</body>
</html>
```

# PHP String Operators

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

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


<?php
$txt1 = "Hello";
$txt2 = " world!";
$txt1 .= $txt2;
echo $txt1;  // Hello world
echo"<br>";
$txt3 = "Hi";
$txt4 = "  Rahul!";
echo $txt3 . $txt4;// Hi Rahul
?>
</body>
</html>
```

# PHP Array Operators

| Operator | Name | Example | Result |
|---|---|---|---|
| + | Union | $x + $y | Union of $x and $y |
| == | Equality | $x == $y | Returns true if $x and $y have the same key/value pairs |
| === | Identity | $x === $y | Returns true if $x and $y have the same key/value pairs **in the same order and of the same types** |
| != | Inequality | $x != $y | Returns true if $x is not equal to $y |
| <> | Inequality | $x <> $y | Returns true if $x is not equal to $y |
| !== | Non-identity | $x !== $y | Returns true if $x is not identical to $y |

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = array("a" => "red", "b" => "green");
$y = array("b" => "green", "a" => "red");
print_r($x + $y); // union of $x and $y    Array ( [a] => red [b] => green )
echo"<br>";
var_dump($x == $y);//bool(true)
echo "<br>";
var_dump($x === $y);//bool(false)
echo "<br>";
var_dump($x != $y);//bool(false)
echo "<br>";
var_dump($x <> $y);//bool(false)
echo "<br>";
var_dump($x !== $y);//bool(true)
?>
</body>
</html>
```

# Bitwise Operators

- Bitwise operators allow operating on the bitwise representation of their arguments.

| Operator | Name | Example | Result |
|---|---|---|---|
| & | And | $x & $y | Bits that are set in both $x and $y are set. |
| \| | Or | $x \| $y | Bits that are set in either $x or $y are set. |
| ^ | Xor | $x ^ $y | Bits that are set in $x or $y but not both are set. |
| ~ | Not | ~$x | Bits that are set in $x are not set, and vice versa. |
| << | Shift left | $x << $y | Shift the bits of $x $y steps to the left.# |
| >> | Shift right | $x >> $y | Shift the bits of $x $y steps to the right.* |

```php
<?php
$x=5;
$y=1;
echo $x & $y;      //1
echo"<br>";
echo $x | $y;     //5
echo"<br>";
echo $x ^ $y;      //4
echo"<br>";
echo $x & ~ $y;    //4
echo"<br>";
echo $x >> $y;     //2
echo"<br>";
echo $x << $y;     //10
?>
```

| Operators | Additional Information | Associativity |
|---|---|---|
| clone new | clone and new | non-associative |
| [ | array() | left |
| ** | arithmetic | right |
| ++ -- ~ (int) (float) (string) (array) (object) (bool) @ | increment/decrement and types | right |
| instanceof | types | non-associative |
| ! | logical (negation) | right |
| * / % | arithmetic | left |
| + - . | arithmetic and string concatenation | left |
| << >> | bitwise (shift) | left |
| < <= > >= | comparison | non-associative |
| == != === !== <> | comparison | non-associative |
| & | bitwise AND | left |
| ^ | bitwise XOR | left |
| \| | bitwise OR | left |
| && | logical AND | left |
| \|\| | logical OR | left |
| ?: | ternary | left |
| = += -= *= **= /= .= %= &= \|= ^= <<= >>= => | assignment | right |
| and | logical | left |
| xor | logical | left |
| or | logical | left |

# Single Quoted PHP String

- We can create a string in PHP by enclosing text in a single quote. It is the easiest way to specify string in PHP.

```
<?php
$str='Hello text within single quote';
echo $str;
?>
```

- **Output:**

Hello text within single quote

# Example 1

```
<?php
$str1='Hello text
multiple line
text within single quoted string';
$str2='Using double "quote" directly inside single quoted string'
;
$str3='Using escape sequences \n in single quoted string';
echo "$str1 <br/> $str2 <br/> $str3";
?>
```

- **Output:**
- Hello text multiple line text within single quoted string
  Using double "quote" directly inside single quoted string
  Using escape sequences \n in single quoted string

# Example 2

- **In single quoted PHP strings, most escape sequences and variables will not be interpreted. But, we can use single quote through \ and backslash through \\ inside single quoted PHP strings.**

```
<?php
$num1=10;
$str1='trying variable $num1';
$str2='trying backslash n and backslash t inside single quoted string \n \t';
$str3='Using single quote \'my quote\' and \\backslash';
echo "$str1 <br/> $str2 <br/> $str3";
?>
```

**Output:**

- trying variable $num1
  trying backslash n and backslash t inside single quoted string \n \t
  Using single quote 'my quote' and \backslash

# Double Quoted PHP String

- In PHP, we can specify string through enclosing text within double quote also. But escape sequences and variables will be interpreted using double quote PHP strings.

```
<?php
$str="Hello text within double quote";
echo $str;
?>
```

- Output:

```
Hello text within double quote
```

# Example 1

```
<?php
$str1="Using double "quote" directly inside double quoted string";
echo $str1;
?>
```

**Output:**

**Parse error**: syntax error, unexpected 'quote' (T_STRING) in C:\xampp\htdocs\Class\Randooom.php on line 2

***you **can't use double quote directly** inside double quoted string.

# Example 2

```
<?php
$str1="Hello text
multiple line
text within double quoted string";
$str2="Using double \"quote\" with backslash inside double q
uoted string";
$str3="Using escape sequences \n in double quoted string";
echo "$str1 <br/> $str2 <br/> $str3";
```

**Output:**

Hello text multiple line text within double quoted string

Using double "quote" with backslash inside double quoted string

Using escape sequences in double quoted string

# Example 3

In double quoted strings, **variable will be interpreted**.

```
<?php
$num1=10;
echo "Number is: $num1";
?>
```

**Output:**

- Number is: 10

# PHP CONDITIONAL STATEMENTS

In PHP we have the following conditional statements:

- **if statement** - executes some code if one condition is true
- **if...else statement** - executes some code if a condition is true and another code if that condition is false
- **if...elseif....else statement** - executes different codes for more than two conditions
- **switch statement** - selects one of many blocks of code to be executed

# If Statement

**Syntax**

- if (*condition*) {
  *code to be executed if condition is true*;
  }

```
<?php
$num=12;
if($num<100)
{
echo "$num is less than 100";
}
?>
```

**Output:**

12 is less than 100

# The if...else Statement

- **Syntax**

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

```
<?php
$str1="Hi";
$str2 ="Hi";
if($str1==$str2)
  echo "Both the strings are equal";
else
  echo "Both the strings are not equal";
?>
```

**Output:** Both the strings are equal

```
<?php
$month =date("M");
if($month == "OCT")
{
echo "The month is October<br>";
echo "Its Diwali";

}
else
{
echo "The month is not October<br>";
echo "It is  ". $month;
}
?>
```

**Output:**

The month is not October
It is Feb

# The if...elseif....else Statement

**Syntax**

- if (*condition*) {
  *code to be executed if this condition is true;*
  } elseif (*condition*) {
  *code to be executed if this condition is true;*
  } else {
  *code to be executed if all conditions are false;*
  }

```
<?php
$m=date("M");
echo "Hi everyone<br>";

if ($m == "Jan") {
   echo "Its January!";
}
elseif ($m == "Feb")
{
   echo "Its February!";
}
elseif ($m== "Mar")
{
   echo "Its March!";
}
else
{ echo "Its some other month";
}
?>
```

**Output:**

Hi everyone
Its  February!

# The switch Statement

**Syntax**

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    case label3:
        code to be executed if n=label3;
        break;
    ...
    default:
        code to be executed if n is different from all labels;
}
```

****break** to prevent the code from running into the next case automatically. The **default** statement is used if no match is found.

```php
<?php
$favcolor = "red";

switch ($favcolor) {
  case "red":
    echo "Your favorite color is red!";
    break;
  case "blue":
    echo "Your favorite color is blue!";
    break;
  case "green":
    echo "Your favorite color is green!";
    break;
  default:
    echo "Your favorite color is neither red, blue, nor green!";
}
?>
```

**Output:**

Your favorite color is red!

# Looping

In PHP, we have the following looping statements:

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

# While Loop

- he while loop executes a block of code as long as the specified condition is true.
- Syntax
- while (*condition is true*) {
  *code to be executed*;
  }

```
<?php
$ctr=1;
echo"The multiples of the number 25 upto 10 times are:<br>";
while($ctr<11)
{
echo"25 X $ctr=".($ctr*25)."<br>";
$ctr++;
}
?>
```

**Output:**

The multiples of the number 25 upto 10 times are:
25 X 1=25
25 X 2=50
25 X 3=75
25 X 4=100
25 X 5=125
25 X 6=150
25 X 7=175
25 X 8=200
25 X 9=225
25 X 10=250

# do...while Loop

- The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

**Syntax**

- do {
  *code to be executed;*
  } while (*condition is true*);

**In a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition is false the first time.

```php
<?php
$ctr=0;
echo"The multiples of the number 40 up to 10 times are:<br>";
do {
$ctr++;
echo "40  X  $ctr =".($ctr*40). "<br>";
}while($ctr<10);
?>
```

**Output:**

The multiples of the number 40 up to 10 times are:
40 X 1 =40
40 X 2 =80
40 X 3 =120
40 X 4 =160
40 X 5 =200
40 X 6 =240
40 X 7 =280
40 X 8 =320
40 X 9 =360
40 X 10 =400

# For Loop

- The for Loop statement is a repetition statement designed to be used when executing a code block a specific number of times.
  - The for loop is a special kind of loop covering the following tasks:
  - Set a counter variable to some initial value.
  - Check to see if the conditional statement is true.
  - Execute the code spec held within the loop
  - Increment a counter at the end of each iteration through the loop.

- **Syntax**

for (*init counter; test counter; increment counter*)
{
   *code to be executed;*
}

- **Parameters:**

***init counter*:** Initialize the loop counter value

***test counter*:** Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.

***increment counter***: Increases the loop counter value

```
<?php
for($ctr =1;$ctr<=5;$ctr++)
{
echo"The for loop has executed $ctr time<br>";
}
?>
```

- **Output:**

The for loop has executed 1 time
The for loop has executed 2 time
The for loop has executed 3 time
The for loop has executed 4 time
The for loop has executed 5 time

```
<?php
for($a =1,$b=10;$a<5;$a++,$b++)
{
$c=$a+$b;
echo"a:".$a."<br> b:". $b."<br> c:".$c."<br> <br>";
}
?>
```

- **Output:**

a:1
b:10
c:11

a:2
b:11
c:13

a:3
b:12
c:15

a:4
b:13
c:17

# For Each Loop

- The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.
- **Syntax for Indexed array**

foreach ($*array* as $*value*)

{

*code to be executed;*

}

- For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

- **Syntax for Associative array:**

foreach ($*array* as *$key* => $*value*)

{

*code to be executed;*

}

- This does the same thing, except that the current element's key will be assigned to the variable $key on each pass of the loop.

```
<?php
$name= array(“Dr","Susheela");
$num=1;
foreach($name as $value)
{
echo"Value held in position $num is: $value.<br>";
$num++;
}
?>
```

- **Output:**

Value held in position 1 is: Dr.
Value held in position 2 is: Susheela.

```
<?php
$name=
array("FirstName"=>“Dr","LastName"=>“Susheela");
$num=1;
foreach($name as $name => $value)
{
echo"The name value pair on position $num
is:\$name[$name]=>$value.<br>";
$num++;
}
?>
```

**Output:**

The name value pair on position 1 is:$name[FirstName]=>Dr.
The name value pair on position 2 is:$name[LastName]=>Susheela.

- When foreach() loop first starts executing, the internal array pointer is automatically reset to the first element of the array.
- The foreach() loop operates on a copy of a specified array and not the array itself.
- Changes to the array element returned are not reflected in the original array. However, the internal pointer of the original array is advanced when the copy of the array is being processed.

# Infinite Loops

- Loops can be very handy but also very dangerous. Infinite Loops are loops that fail to exit, causing them to execute until power of the computer is stopped.
- This occurs no matter what occurs during the loop, the condition will never become false and therefore for the loop never exits.
- Relying on user input to terminate a loop is possible (ctrl+C).

```
<?php
while(1)
{
print"In loop!";
}
?>
```

```
<?php
for(;;)
{
print “In loop”;
}
?>
```

# Special Loop Keywords

- **break**
  - It accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.
- **Continue**
  - It accepts an optimal numeric argument which tells it how many levels of enclosing loops it should skip to the end of.

** When used inside loops in order to manipulate the loop behaviour, break causes PHP to exit the loop and continue causes PHP to skip the remaining code block in the current loop iteration and go onto the next iteration.

```php
<?php
for($i=0;$i<12;$i++)
{
    if($i ==5)
    {
        echo("Loop Continued at $i<br>");
        continue;
    }
    if($i==10)
    {
        echo("Loop terminated at $i<br>");
        break;
    }
    echo("Number $i <br>");
}
?>
```

**Output:**

Number 0
Number 1
Number 2
Number 3
Number 4
Loop Continued at 5
Number 6
Number 7
Number 8
Number 9
Loop terminated at 10

# Loops within Loops(Nested Loops)

```
<?php
for($a=1;$n<3;$a++)
{
      for($b=0;$b<7;$b=$b+2)
      {
            for($c=10;$c>8;$c--)
            {
                 echo("A holds:$a, B holds: $b, C holds:$c <br>");
                break 2;
            }
      }
}
?>
```

**Output:**

A holds:1, B holds: 0, C holds:10

A holds:2, B holds: 0, C holds:10

** Break 2 means that the control will break out from the inner loop and one above it

# Functions

# Content

- Introduction to Functions
- Advantage of Functions
- Creation of User Defined Functions
- Function Arguments
  - Call by Value (default)
  - Call by Reference
  - Default argument values
  - Variable-length argument list.
- Returning Value
- Recursive Function

# Introduction

- PHP function is a piece of code that can be reused many times.
- It can take input as argument list and return value.
- There are thousands of built-in functions in PHP.

**PHP User Defined Functions**

- Besides the built-in PHP functions, we can create our own functions.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.
- In PHP, we can define **Conditional function**, **Function within Function** and **Recursive function** also.

# Advantage of PHP Functions

- **Code Reusability**: PHP functions are defined only once and can be invoked many times, like in other programming languages.

- **Less Code**: It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.

- **Easy to understand**: PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

# Creation of User Defined function in PHP

- A user defined function declaration starts with the word "function":

**Syntax**

function *functionName*()

{

*code to be executed*;

}

**Note: Function name must be start with letter and underscore only like other labels in PHP. It can't be start with numbers or special symbols.**

# Example

```
<?php
function sayHello()
{
echo "Hello PHP Function";
}
sayHello();//calling function
?>
```

Output: Hello PHP Function

# PHP Function Arguments

- We can pass the information in PHP function through arguments which is separated by comma.
- PHP supports
  - Call by Value (default)
  - Call by Reference
  - Default argument values
  - Variable-length argument list.

# Default Argument Value

- We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument
- If we call the function setHeight() without arguments it takes the default value as argument:

**Example1**

```
<?php
function setHeight($minheight = 50)
{
   echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight();
setHeight(135);
setHeight(80);
?>
```

**Output**:

```
The height is : 350
The height is : 50
The height is : 135
The height is : 80
```

# Example 2

```
<?php
function greeting($first="Sono",$last="Jaiswal")
{
echo "Greeting: $first $last<br/>";
}
greeting();
greeting("Rahul");
greeting("Michael","Clark");
?>
```

- Output:

Greeting: Sono Jaiswal
Greeting: Rahul Jaiswal
Greeting: Michael Clark

# Call by Value

PHP allows you to call function by value and reference both. In case of PHP call by value, actual value is not modified if it is modified inside the function.

```
<?php
function adder($str2)
{
   $str2 .= 'Call By Value';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
```

Output: Hello

In this example, variable $str is passed to the adder function where it is concatenated with 'Call By Value' string. But, printing $str variable results 'Hello' only. It is because changes are done in the local variable $str2 only. It doesn't reflect to $str variable.

# Call By Reference

- Value passed to the function doesn't modify the actual value by default (call by value). But we can do so by passing value as a reference.
- By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

# Example

```
<?php
function adder(&$str2)
{
   $str2 .= 'Call By Reference';
}
$str = 'Hello ';
adder( &$str);
echo $str;
?>
```

Output:

Hello Call By Reference

# Variable Length Argument Function

- PHP supports variable length argument function. It means you can pass 0, 1 or n number of arguments in function. To do so, you need to use 3 ellipses (dots) before the argument name.
- The 3 dot concept is implemented for variable length argument since PHP 5.6.

# Example

```
<?php
function add(...$numbers)
{
    $sum = 0;
    foreach ($numbers as $n) {
        $sum += $n;
    }
    return $sum;
}

echo add(1, 2, 3, 4);
?>
```

**Output:**

- 10

# Returning Value

- To let a function return a value, use the return statement:

```
<?php
function cube($n)
{
return $n*$n*$n;
}
echo "Cube of 3 is: ".cube(3);
?>
```

- Output: Cube of 3 is: 27

# Recursive Function

- PHP also supports recursive function call like C/C++. In such case, we call current function within function. It is also known as recursion.
- It is recommended to avoid recursive function call over 200 recursion level because it may smash the stack and may cause the termination of script.

# Example

```
<?php
function factorial($n)
{
    if ($n < 0)
        return -1; /*Wrong value*/
    if ($n == 0)
        return 1; /*Terminating condition*/
    return ($n * factorial ($n -1));
}

echo factorial(5);
?>
```

- Output:120

# Functions

# Introduction

- PHP function is a piece of code that can be reused many times.
- It can take input as argument list and return value.
- There are thousands of built-in functions in PHP.

**PHP User Defined Functions**

- Besides the built-in PHP functions, we can create our own functions.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.
- In PHP, we can define **Conditional function**, **Function within Function** and **Recursive function** also.

# Advantage of PHP Functions

- **Code Reusability**: PHP functions are defined only once and can be invoked many times, like in other programming languages.

- **Less Code**: It saves a lot of code because you don't need to write the logic many times. By the use of function, you can write the logic only once and reuse it.

- **Easy to understand**: PHP functions separate the programming logic. So it is easier to understand the flow of the application because every logic is divided in the form of functions.

# Creation of User Defined function in PHP

- A user defined function declaration starts with the word "function":

**Syntax**

function *functionName*()

{

*code to be executed*;

}

**Note: Function name must be start with letter and underscore only like other labels in PHP. It can't be start with numbers or special symbols.**

# Example

```
<?php
function sayHello()
{
echo "Hello PHP Function";
}
sayHello();//calling function
?>
```

Output: Hello PHP Function

# PHP Function Arguments

- We can pass the information in PHP function through arguments which is separated by comma.
- PHP supports
  - Call by Value (default)
  - Call by Reference
  - Default argument values
  - Variable-length argument list.

# Default Argument Value

- We can specify a default argument value in function. While calling PHP function if you don't specify any argument, it will take the default argument
- If we call the function setHeight() without arguments it takes the default value as argument:

**Example1**

```
<?php
function setHeight($minheight = 50)
{
   echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight();
setHeight(135);
setHeight(80);
?>
```

**Output**:

```
The height is : 350
The height is : 50
The height is : 135
The height is : 80
```

# Example 2

```
<?php
function greeting($first="Sono",$last="Jaiswal")
{
echo "Greeting: $first $last<br/>";
}
greeting();
greeting("Rahul");
greeting("Michael","Clark");
?>
```

- Output:

Greeting: Sono Jaiswal
Greeting: Rahul Jaiswal
Greeting: Michael Clark

# Call by Value

PHP allows you to call function by value and reference both. In case of PHP call by value, actual value is not modified if it is modified inside the function.

```
<?php
function adder($str2)
{
   $str2 .= 'Call By Value';
}
$str = 'Hello ';
adder($str);
echo $str;
?>
```

Output: Hello

In this example, variable $str is passed to the adder function where it is concatenated with 'Call By Value' string. But, printing $str variable results 'Hello' only. It is because changes are done in the local variable $str2 only. It doesn't reflect to $str variable.

# Call By Reference

- Value passed to the function doesn't modify the actual value by default (call by value). But we can do so by passing value as a reference.
- By default, value passed to the function is call by value. To pass value as a reference, you need to use ampersand (&) symbol before the argument name.

# Example

```
<?php
function adder(&$str2)
{
   $str2 .= 'Call By Reference';
}
$str = 'Hello ';
adder( &$str);
echo $str;
?>
```

Output:

Hello Call By Reference

# Variable Length Argument Function

- PHP supports variable length argument function. It means you can pass 0, 1 or n number of arguments in function. To do so, you need to use 3 ellipses (dots) before the argument name.
- The 3 dot concept is implemented for variable length argument since PHP 5.6.

# Example

```
<?php
function add(...$numbers)
{
    $sum = 0;
    foreach ($numbers as $n) {
        $sum += $n;
    }
    return $sum;
}

echo add(1, 2, 3, 4);
?>
```

**Output:**

- 10

# Returning Value

- To let a function return a value, use the return statement:

```
<?php
function cube($n)
{
return $n*$n*$n;
}
echo "Cube of 3 is: ".cube(3);
?>
```

- Output: Cube of 3 is: 27

# Recursive Function

- PHP also supports recursive function call like C/C++. In such case, we call current function within function. It is also known as recursion.
- It is recommended to avoid recursive function call over 200 recursion level because it may smash the stack and may cause the termination of script.

# Example

```
<?php
function factorial($n)
{
    if ($n < 0)
        return -1; /*Wrong value*/
    if ($n == 0)
        return 1; /*Terminating condition*/
    return ($n * factorial ($n -1));
}

echo factorial(5);
?>
```

- Output:120

# PHP Include Files

- The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.
- Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

# PHP include and require Statements

- It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

# Syntax

include 'filename';

or

require 'filename';

- *Use **require** when the file is 'strictly' required by the application.*

- *Use **include** when the file is not required and application should continue when file is not found.*

# PHP include vs. require

- The require statement is also used to include a file into the PHP code.

- However, there is one big difference between include and require; when a file is included with the **include** statement and PHP cannot find it, the script will continue to execute:

```
<html>
<body>
<h1>Welcome to my home page!</h1>
<?php include 'noFileExists.php';
echo "I have a $color $car.";
?>
</body>
</html>
```

include will only produce a warning (E_WARNING) and the script will continue.

- If we do the same example using the **require** statement, the echo statement will not be executed because the script execution dies after the require statement returned a fatal error:

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<?php require 'noFileExists.php';
echo "I have a $color $car.";
?>

</body>
</html>
```

**Output:** require will produce a fatal error (E_COMPILE_ERROR) and stop the script

# The include and require statements are identical, except upon *failure*

- require will produce a fatal error (E_COMPILE_ERROR) and stop the script

- include will only produce a warning (E_WARNING) and the script will continue

- Including files saves a lot of work.
- This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

# Browser Control

- PHP can control various features of a browser.
- This is important as often there is a need to reload the same page or redirecting the user to another page.
- Some of these features are accessed by controlling the information sent out in the HTTP header to the browser, this uses the header() command such as :

```
header("Location: index.php");
```

- We can also control the caching using same header() command

```
header("Cache-Control: no-cache");
```

**Or** can specify the content type like,

```
header("Content-Type: application/pdf");
```

# Example

```
<html>
<head>
<title> Browser Control</title>
</head>
<body>
    <?php
        header("Location: http://www.google.co.in");

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

- Redirect the page to Google

# Browser Detection

- The range of devices with browsers is increasing so it is becoming more important to know which browser and other details you are dealing with.
- The browser that the server is dealing can be identified using:

```
$browser_ID = $_SERVER['HTTP_USER_AGENT'];
```

- Typical response of the above code is follows:

  Mozilla/5.0 (**Windows NT 10.0**; Win64; **x64**) AppleWebKit/537.36 (KHTML, like Gecko) **Chrome**/56.0.2924.87

  - which specifies that user is using **Chrome** browser and **windows 10** OS with **64 bit** architecture

# Example

```
<html>
<head>
<title> Browser Detection</title>
</head>
<body>
    <?php
        $b = $_SERVER[‘HTTP_USER_AGENT’];
        echo($b);
    ?>
 </body>
</html>
```

Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36

# PHP Form Processing

# Form Processing

- For handling the form basically the PHP code is directly embedded within the HTML document.
- The values of the form elements are can be obtained by the PHP script using $_GET and $_POST variable
- The HTML makes use of GET or POST method when it invoke the PHP script.
- If we collect the values using the GET method then $_GET variable is used in the PHP script. Note that if the GET method is used then all variable names and values will be displayed in the URL as a Query String.
- But if we used POST method then these values will not be displayed the URL.

# Form Processing - Example

```
<html>
    <body>
        <form action="welcome_get.php" method="get">
            Name: <input type="text" name="name"><br />
            E-mail: <input type="text" name="email"><br />
            <input type="submit">
        </form>
    </body>
</html>
```

# Form Processing - Example

welcome_get.php

```
<html>
    <body>

        Welcome <?php echo $_GET["name"]; ?><br>
        Your email address is: <?php echo $_GET["email"]; ?>

    </body>
</html>
```