What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

What is a PHP File?

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

What Can PHP Do?

- PHP can generate dynamic web 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

PHP Structure

```
<?php  echo "Hello Everyone"; ?>
```

with HTML

```
<html>

<head>

    <title>PHP</title>

</head>

<body>

    <?php echo "Hello Everyone"; ?>

</body>

</html>
```

PHP Echo/Print

PHP echo and print Statements

`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`.

The PHP echo Statement

The `echo` statement can be used with or without parentheses: `echo` or `echo()`.

Display Text

The following example shows how to output text with the `echo` command (notice that the text can contain HTML markup):

```
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.";
```

Display Variables

The following example shows how to output text and variables with the `echo` statement:

Example

```
$txt1 = "Learn PHP";

$txt2 = "W3Schools.com";

$x = 5;

$y = 4;

echo "<h2>" . $txt1 . "</h2>";

echo "Study PHP at " . $txt2 . "<br>";

echo $x + $y;
```

The PHP print Statement

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

Display Text

The following example shows how to output text with the `print` command (notice that the text can contain HTML markup):

Example

```
print "<h2>PHP is Fun!</h2>";

print "Hello world!<br>";

print "I'm about to learn PHP!";
```

Display Variables

The following example shows how to output text and variables with the `print` statement:

Example

```
$txt1 = "Learn PHP";

$txt2 = "W3Schools.com";

$x = 5;

$y = 4;

print "<h2>" . $txt1 . "</h2>";

print "Study PHP at " . $txt2 . "<br>";

print $x + $y;

<?php

    echo "LBSTI";

   echo 'LBSTI';

   echo ('LBSTI');

    echo "Yahoo","Baba";

    echo "Yahoo"."Baba";

    echo "<h1><i>Yahoo"."Baba</i></h1>";

    echo 23.58;
```

```
    print "<h1><i>Yahoo"."Baba</i></h1>";

    print 23.58;

?>
```

PHP Variables

Variables are "containers" for storing information.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the `$` sign, followed by the name of the variable:

Example

```
$x = 5;

$y = "John"
```

In the example above, the variable `$x` will hold the value `5`, and the variable `$y` will hold the value `"John"`.

PHP Variables

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)

Output Variables

The PHP `echo` statement is often used to output data to the screen.

The following example will show how to output text and a variable:

Example

```
$txt = "W3Schools.com";

echo "I love $txt!";
```

The following example will output the sum of two variables:

Example

```
$x = 5;

$y = 4;

echo $x + $y;
```

Assign String to a Variable

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
$x = "John";

echo $x;
```

Assign Multiple Values

You can assign the same value to multiple variables in one line:

Example

All three variables get the value "Fruit":

```
$x = $y = $z = "Fruit";
```

```
<?php
    $name = "LBSTI<br>";
    $num = 2358;
    echo $name;
    echo "<h1>" . $name . "</h1>";


    echo "Hello how are you : " . $name ;
    echo $num;
?>
```

PHP Data Types

PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

Getting the Data Type

You can get the data type of any object by using the `var_dump()` function.

Example

The `var_dump()` function returns the data type and the value:

```
$x = 5;

var_dump($x);
```

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:

Example

```
$x = "Hello world!";

$y = 'Hello world!';



var_dump($x);

echo "<br>";

var_dump($y);
```

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: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example `$x` is an integer. The PHP `var_dump()` function returns the data type and value:

Example

```
$x = 5985;

var_dump($x);
```

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. The PHP `var_dump()` function returns the data type and value:

Example

```
$x = 10.365;

var_dump($x);
```

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

Example

```
$x = true;

var_dump($x);
```

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:

Example

```
$cars = array("Volvo","BMW","Toyota");

var_dump($cars);
```

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.

Tip: 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:

Example

```
$x = "Hello world!";

$x = null;

var_dump($x);

<?php

    $x = "LBSTI";

    $x = 2500;

    $x = 2500.50;

    $x = true;

    $x = array("html","css","js");

    $x = null;

    echo  $x . "<br>";

    echo  $x[0] . "<br>";
```

```
    var_dump($x);

?>
```

PHP Comment
Comments in PHP

A comment in PHP code is a line that is not executed as a part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

- Let others understand your code
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code
- Leave out some parts of your code

PHP supports several ways of commenting:

Example

Syntax for comments in PHP code:

```
// This is a single-line comment


# This is also a single-line comment


/* This is a

multi-line comment */
```

Single Line Comments

Single line comments start with `//`.

Any text between `//` and the end of the line will be ignored (will not be executed).

You can also use `#` for single line comments, but in this tutorial we will use `//`.

The following examples uses a single-line comment as an explanation:

Example

A comment before the code:

```
// Outputs a welcome message:

echo "Welcome Home!";
```

Multi-line Comments

Multi-line comments start with `/*` and end with `*/`.

Any text between `/*` and `*/` will be ignored.

The following example uses a multi-line comment as an explanation:

Example

Multi-line comment as an explanation:

```
/*

The next statement will

print a welcome message

*/

echo "Welcome Home!";
```

Comments in the Middle of the Code

The multi-line comment syntax can also be used to prevent execution of parts inside a code-line:

Example

The `+ 15` part will be ignored in the calculation:

```
$x = 5 /* + 15 */ + 5;

echo $x;

<?php

   /*  Single Line Comment */

    $x = "LBSTI"; //this  is first comment

    echo  $x;
```

```
    /*  Multiple Line Comment */

   /*  $x = "LBSTI";

    echo  $x; */

?>
```

PHP Constants

PHP Constants

A 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).

Note: 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);
```

`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. **Note:** Defining case-insensitive constants was deprecated in PHP 7.3. PHP 8.0 accepts only false, the value true will produce a warning.

Example

Create a constant with a **case-sensitive** name:

```
define("GREETING", "Welcome to W3Schools.com!");

echo GREETING;
```

Example

Create a constant with a **case-insensitive** name:

```
define("GREETING", "Welcome to W3Schools.com!", true);

echo greeting;
```

```
<?php
    define("test",50);
    echo test;
    define("test1",50,true);
    echo TEST1;
   $sum = test + 20;
   echo $sum;
?>
```

PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

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

```
<?php
    $a = 10;
    $b = 3;
```

```
    $c =  $a + $b;

    $c =  $a - $b;

    $c =  $a * $b;

    $c =  $a / $b;

    $c =  $a % $b;

    $c =  $a ** $b;

    $c =  $a % $b;

    $a++;

    ++$a;

    $a--;

    $a;

    $c = ($a + $b) * 2;


    echo $c;
?>
```

PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

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
    $a = 10;
    $b = 3;

    $a =  $a + $b;
    $a += $b;
    $a -=  $b;
    $a *=  $b;
    $a /=  $b;
    $a %=  $b;
    $a **=  $b;

    echo $a;
?>
```

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

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
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7.

```
<?php

    $a = 10;

    $b = 10;



    echo  $a == $b;

    echo  $a === $b;

    echo  $a != $b;

    echo  $a <> $b;

    echo  $a !== $b;

    echo  $a > $b;

    echo  $a < $b;

    echo  $a >= $b;

    echo  $a <= $b;

    echo  $a <=> $b; //Spaceship (-1,0,1)

?>
```

PHP If

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

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

---

PHP - The if Statement

The `if` statement executes some code if one condition is true.

Syntax

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

Example

Output "Have a good day!" if 5 is larger than 3:

```
if (5 > 3) {
  echo "Have a good day!";
}
```

```
<?php
    $a = 3;
    $b = 10;

    if($a < $b){
        echo "A is Smaller" ;
    }
    echo "Here is other statement";
```

```
    if($a == $b){
        echo "A is Smaller" ;
    }
    echo "Here is other statement";



    if($a === $b){
        echo "A is Smaller" ;
    }
    echo "Here is other statement";



    if($a == $b):
        echo "A is Smaller<br>" ;
        echo "A is Smaller<br>" ;
    endif;
    echo "Here is other statement";
?>
```

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

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

```
<?php
    $age = 20;

    /* Logical And Operator */
    if($age >= 18 && $age <= 21){
    echo "You are eligible.<br>";
    }
    echo "Here is other statement";

    /* Logical And Operator*/
    if($age >= 18 and $age <= 21){
    echo "You are eligible.<br>";
    }
    echo "Here is other statement";

    /* Logical Or Operator*/
    if($age >= 18 || $age <= 21){
    echo "You are eligible.<br>";
    }

    /* Logical Not Operator*/
    if(!($age >= 18)){
```

```
    echo "You are eligible.<br>";
    }

    /* Logical xor Operator*/
    if($age >= 18 xor $age <= 21){
    echo "You are eligible.<br>";
    }
?>
```

PHP - The if...else Statement

The `if...else` statement executes some code if a condition is true and another code if that condition is false.

Syntax

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

Example

Output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

```
$t = date("H");

if ($t < "20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
```

```
}
<?php
    $x = 15;

    if($x > 30){
      echo "X is Greater.";
    }else{
      echo "X is Smaller.";
    }



    $x = 100;

    if($x == 100){
      echo "X is Same.";
    }else{
      echo "X is not Same.";
    }



    $name = "LBSTI";
    $gender = "male";

    if($gender == "male"){
      echo "Hello Mr.". $name;
```

```
    }else{
      echo "Hello Miss.". $name;
    }
?>
```

PHP - The if...elseif...else Statement

The if...elseif...else statement executes different codes for more than two conditions.

Syntax

```
if (condition) {
  code to be executed if this condition is true;
} elseif (condition) {
  // code to be executed if first condition is false and this condition is true;
} else {
  // code to be executed if all conditions are false;
}
```

```
<?php
 $per = 47;

if($per >= 80 && $per <= 100){
  echo "You are in Merit.";
} elseif($per >= 60 && $per < 80){
  echo "You are in Ist Division.";
} elseif($per >= 45 && $per < 60){
  echo "You are in IInd Division.";
} elseif($per >= 33 && $per < 45){
  echo "You are in IIIrd Division.";
```

```
} elseif($per < 33){
  echo "You are Fail.";
} else{
  echo "Please Enter Valid Percentage.";
}
```

```
if($per >= 80 && $per <= 100):
  echo "You are in Merit.";
elseif($per >= 60 && $per < 80):
  echo "You are in Ist Division.";
elseif($per >= 45 && $per < 60):
  echo "You are in IInd Division.";
elseif($per >= 33 && $per < 45):
  echo "You are in IIIrd Division.";
elseif($per < 33):
  echo "You are Fail.";
else:
  echo "Please Enter Valid Percentage.";
endif;
?>
```

The PHP switch Statement

Use the `switch` statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (expression) {
```

```
  case label1:
    //code block
    break;
  case label2:
    //code block;
    break;
  case label3:
    //code block
    break;
  default:
    //code block
}
```

This is how it works:

- The *expression* is evaluated once
- The value of the expression is compared with the values of each case
- If there is a match, the associated block of code is executed
- The `break` keyword breaks out of the switch block
- The `default` code block is executed if there is no match

```
<?php
$weekday = 7;


switch($weekday){
  case 1:
      echo "Today is Monday";
  break;
  case 2:
      echo "Today is Tuesday";
```

```
  break;
  case 3:
      echo "Today is Wednesday";
  break;
  case 4:
      echo "Today is Thursday";
  break;
  case 5:
      echo "Today is Friday";
  break;
  case 6:
      echo "Today is Saturday";
  break;
  case 7:
      echo "Today is Sunday";
  break;
  default:
      echo "Enter the correct weekday.";
}
$weekday = 3;
switch($weekday){
  case 1 : case 2 : case 3 :
      echo "Today is Monday";
      echo "<br>This is just test.";
      break;
```

```
  case 4 :
      echo "Today is Thursday";
      break;
  case 5 :
      echo "Today is Friday";
      break;
  case 6 :
      echo "Today is Saturday";
      break;
  case 7 :
      echo "Today is Sunday";
      break;
  default :
      echo "Enter the correct weekday.";
}
$age = 18;
switch($age){
  case ($age >= 15 && $age <=20) :
      echo "You are eligible.";
  break;
  case ($age >= 20 && $age <= 30) :
      echo "You are not eligible.";
  break;
  default :
      echo "Enter the valid age.";
```

```
}

?>
```

Short Hand If...Else (Ternary Operator)

There is also a short-hand if else, which is known as the **ternary operator** because it consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements:

Syntax

```
variable = (condition) ? expressionTrue : expressionFalse;

<?php

$x = 10;



($x > 20) ? $z = "Greater" : $z = "Smaller";



$z = ($x > 20) ?  "Greater" : "Smaller";



$z = "Value is " . ($x > 20 ?  "Greater" : "Smaller");



echo $z;

?>
```

Strings

A string is a sequence of characters, like "Hello world!".
Strings in PHP are surrounded by either double quotation marks, or single quotation marks.

```
echo "Hello";

echo 'Hello';
```

Double or Single Quotes?

You can use double or single quotes, but you should be aware of the differences between the two.

Double quoted strings perform action on special characters.

E.g. when there is a variable in the string, it returns the *value* of the variable:

Example

Double quoted string literals perform operations for special characters:

```
$x = "John";

echo "Hello $x";

<?php

$a = "Hello" ;

$s = $a . " World ";

$a = 200;

$s = $a . " World " . 500;

$s = "Hello ";

$s .= " this is";

$s .= " our world";

$s .= 555;

echo $s;

?>
```

PHP while Loop

PHP while Loop

The `while` loop executes a block of code as long as the specified condition is true.

Example

Print `$i` as long as `$i` is less than 6:

```
$i = 1;

while ($i < 6) {

  echo $i;
```

```
  $i++;

}
```

The `while` loop does not run a specific number of times, but checks after each iteration if the condition is still true.

The condition does not have to be a counter, it could be the status of an operation or any condition that evaluates to either true or false.

```
<?php

$a = 1;


 while($a <= 10){

  echo "Hello LBSTI<br>";

  $a = $a + 1; //Increment Loop

}


while($a <= 20){

  echo $a . ") Hello LBSTI<br>";

  $a = $a++; //counting

}




$a = 10;

while($a >= 1){

  echo $a . ") Hello LBSTI<br>";

    $a--; //Decrement Loop

}
```

```
echo "<ul>";

while($a >= 1){

  echo "<li>".$a . ") Hello LBSTI</li>";

  $a = $a - 1;

}

echo "</ul>";




while($a <= 10){

  echo "Hello LBSTI<br>";

  $a = $a + 2; //increament of 2 or 3

}

?>
```

The PHP do...while Loop

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

Example

Print `$i` as long as `$i` is less than 6:

```
$i = 1;


do {

  echo $i;

  $i++;

} while ($i < 6);

<?php
```

```
$a = 1;


do{
   echo $a .") Hello LBSTI<br>";
   $a++;  //Increment Loop
}while($a <= 10)


$a = 10;


do{
   echo $a .") Hello LBSTI<br>";
   $a--;  //Decrement Loop
}while($a <= 1)
?>
```

PHP for Loop

The `for` loop is used when you know how many times the script should run.

Syntax

```
for (expression1, expression2, expression3) {
  // code block
}
```

This is how it works:

- *expression1* is evaluated once
- *expression2* is evaluated before each iterarion
- *expression3* is evaluated after each iterarion

```
<?php
```

```
//Increment

for($a = 1; $a <= 10; $a= $a++ ){

   echo $a .") Hello LBSTI<br>";

}


 //Decrement

for($a = 10; $a >= 1; $a= $a-- ){

   echo $a .") Hello LBSTI<br>";

}

?>
```

Nested Loop in PHP

The nested loop is the method of using the for loop inside the another for loop. It first performs a single iteration for the parent for loop and executes all the iterations of the inner loop. After that, it again checks the parent iteration and again performs all the iteration of the inner loop. The same process continuously executed until the parent test condition is TRUE.

```
<?php

for($a = 1; $a <= 100; $a = $a + 10){

  for($b = $a; $b < $a + 10; $b++){

    echo $b . " ";

  }

  echo "<br>";

}

?>
```

PHP Continue & Break

Continue in For Loops

The `continue` statement stops the current iteration in the `for` loop and continue with the next.

Example

Move to next iteration if `$x` = 4:

```
for ($x = 0; $x < 10; $x++) {
  if ($x == 4) {
    continue;
  }
  echo "The number is: $x <br>";
}
```

Continue in While Loop

The `continue` statement stops the current iteration in the `while` loop and continue with the next.

Example

Move to next iteration if `$x` = 4:

```
$x = 0;


while($x < 10) {
  if ($x == 4) {
    continue;
  }
  echo "The number is: $x <br>";
  $x++;
}
```

Break in For loop

The `break` statement can be used to jump out of a `for` loop.

Example

Jump out of the loop when `$x` is `4`:

```
for ($x = 0; $x < 10; $x++) {
  if ($x == 4) {
    break;
  }
  echo "The number is: $x <br>";
}
```

Break in While Loop

The `break` statement can be used to jump out of a `while` loop.

Example

```
$x = 0;

while($x < 10) {
  if ($x == 4) {
    break;
  }
  echo "The number is: $x <br>";
  $x++;
}
<?php
for ($a = 1; $a < 10; $a++) {
 if ($a == 3){
    //echo "No. : " . $a . "<br>";
    continue;
  }
```

```
  echo "Number : " . $a . "<br>";
}

for ($a = 1; $a < 10; $a++) {
 if ($a == 3){
    //echo "No. : " . $a . "<br>";
    break;
  }

  echo "Number : " . $a . "<br>";
}
?>
```

The goto statement

The **goto** statement is used to jump to another section of a program. It is sometimes referred to as an unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

Syntax:

```
statement_1;

if (expr)
       goto label;
statement_2;
statement_3;

label: statement_4;
```

```php
<?php

for($a = 1; $a <= 10; $a++){
  if($a == 3){
    goto a;
  }

  echo "Number : " . $a . "<br>";
}

echo "Hello";
echo " World";

a:
echo "Here is label A.";
?>
```
