Unit-1: PHP fundamentals

# Concepts of Php and introduction

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

* PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
* PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
* It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
* PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
* PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
* PHP is forgiving: PHP language tries to be as forgiving as possible.
* PHP Syntax is C-Like.

# Common uses of PHP

* PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
* PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
* You add, delete, modify elements within your database through PHP.
* Access cookies variables and set cookies.
* Using PHP, you can restrict users to access some pages of your website.
* It can encrypt data.

# Characteristics of PHP

Five important characteristics make PHP's practical nature possible −

* Simplicity
* Efficiency
* Security
* Flexibility
* Familiarity

"Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this −

[Live Demo](http://tpcg.io/cJf0To)

<html>

<head>

<title>Hello World</title>

</head>

<body>

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

</body>

</html>

Output:

Hello, World!

# Variables

variable starts with the $ sign, followed by the name of the variable:

<?php  
$txt = "Hello world!";  
$x = 5;  
$y = 10.5;  
?>

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: Remember that PHP variable names are case-sensitive!

# Variable Scope

PHP has three types of variable scopes:

1. Local variable
2. Global variable
3. Static variable

### Global

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

### Example

Variable with global scope:

<?php

    $name = "Sanaya Sharma";        //Global Variable

**function** global\_var()

    {

**global** $name;

        echo "Variable inside the function: ". $name;

        echo "</br>";

    }

    global\_var();

    echo "Variable outside the function: ". $name;

?>

Another way to use the global variable  inside the function is predefined $GLOBALS array.

<?php

    $num1 = 5;      //global variable

    $num2 = 13;     //global variable

**function** global\_var()

    {

            $sum = $GLOBALS['num1'] + $GLOBALS['num2'];

            echo "Sum of global variables is: " .$sum;

    }

    global\_var();

?>

### Local

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

### Example

Variable with local scope:

<?php  
function myTest() {  
  $x = 5; // local scope  
  echo "<p>Variable x inside function is: $x</p>";  
}  
myTest();  
  
// using x outside the function will generate an error  
echo "<p>Variable x outside function is: $x</p>";  
?>

### Static

It is a feature of PHP to delete the variable, once it completes its execution and memory is freed. Sometimes we need to store a variable even after completion of function execution. Therefore, another important feature of variable scoping is static variable. We use the static keyword before the variable to define a variable, and this variable is called as **static variable**.

Static variables exist only in a local function, but it does not free its memory after the program execution leaves the scope. Understand it with the help of an example:

### Example:

<?php

**function** static\_var()

    {

**static** $num1 = 3;       //static variable

        $num2 = 6;          //Non-static variable

        //increment in non-static variable

        $num1++;

        //increment in static variable

        $num2++;

        echo "Static: " .$num1 ."</br>";

        echo "Non-static: " .$num2 ."</br>";

    }

//first function call

    static\_var();

    //second function call

    static\_var();

?>

Output

Static: 4

Non-static: 7

Static: 5

Non-static: 7

# 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.

To understand the difference better, let's see some examples.

### Example

<?php

$x = "abc";

$$x = 200;

echo $x."<br/>";

echo $$x."<br/>";

echo $abc;

?>

Output

abc

200

200

# Constants

PHP constants are name or identifier that can't be changed during the execution of the script except .

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

Syntax

define(name, value)

Example

  <?php

define("MSG1","Hello JavaTpoint PHP");

echo MSG1;

const MSG2="Hello const by JavaTpoint PHP";

echo MSG2;

?>

# echo

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

Example:

<?php  
echo "<h2>PHP is Fun!</h2>";  
echo "Hello world!<br>";

$txt1 = "Learn PHP";  
$txt2 = "W3Schools.com";  
$x = 5;  
$y = 4;  
  
echo "<h2>" . $txt1 . "</h2>";  
echo "Study PHP at " . $txt2 . "<br>";  
echo $x + $y;

echo "Hello by PHP echo

this is multi line

text printed by

PHP echo statement

";

echo "Hello escape \"sequence\" characters";

?>

# Print

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

<?php  
print "<h2>PHP is Fun!</h2>";  
print "Hello world!<br>";  
print ("Hello by PHP print()");

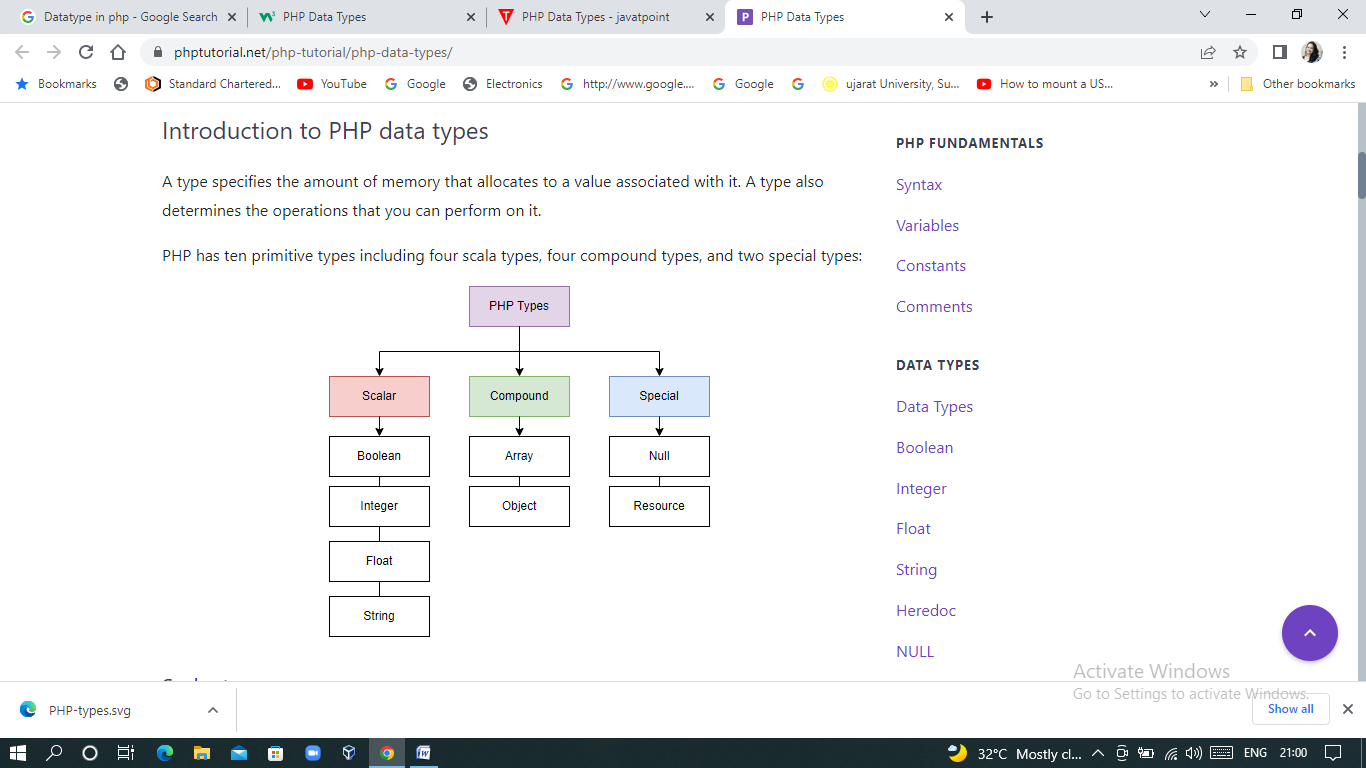
?>

**Echo VS Print**

|  |  |
| --- | --- |
| Echo | Print |
| 1. echo does not return any value. 2. We can pass multiple strings separated by comma (,) in echo. 3. echo is faster than print statement. | 1. print always returns an integer value, which is 1. 2. Using print, we cannot pass multiple arguments. 3. print is slower than echo statement. |

# Data types

A type specifies the amount of memory that allocates to a value associated with it.



**Scalar type:**

It hold single value only:

* 1. Boolean: Booleans are the simplest data type works like switch. It holds only two values: **TRUE (1)** or **FALSE (0)**
  2. Integer:Integer means numeric data with a negative or positive sign. It holds only whole numbers, i.e., numbers without fractional part or decimal points. The range of an integer must be lie between 2,147,483,648 and 2,147,483,647 i.e., -2^31 to 2^31.
  3. Float: A floating-point number is a number with a decimal point.
  4. String: A string is a non-numeric data type. It holds letters or any alphabets, numbers, and even special characters.String values must be enclosed either within single quotes or in double quotes. But both are treated differently.

Example:  $name = "Raman";

    //both single and double quote statements will treat different

    echo "Hello $name";

    echo "</br>";

    echo 'Hello $name';

Output:

Hello Javatpoint

Hello $company

**Compound Type:**

It hold multiple values

1. Array: An array is a compound data type. It can store multiple values of same data type in a single variable.

**Example:**

$scores = [1, 2, 3];

2. Objects are the instances of user-defined classes that can store both values and functions.

**Special type**

1.Resource: Resources are not the exact data type in PHP. Basically, these are used to store some function calls or references to external PHP resources. For example - a database call.

2.  Null: Null is a special data type that has only one value: NULL

# Operators

PHP divides the operators in the following groups:

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Explanation** |
| + | Addition | $a + $b | Sum of operands |
| - | Subtraction | $a - $b | Difference of operands |
| \* | Multiplication | $a \* $b | Product of operands |
| / | Division | $a / $b | Quotient of operands |
| % | Modulus | $a % $b | Remainder of operands |
| \*\* | Exponentiation | $a \*\* $b | $a raised to the power $b |

* Assignment operators: The assignment operators are used to assign value to different variables. The basic assignment operator is "=".

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Explanation** |
| = | Assign | $a = $b | The value of right operand is assigned to the left operand. |
| += | Add then Assign | $a += $b | Addition same as $a = $a + $b |
| -= | Subtract then Assign | $a -= $b | Subtraction same as $a = $a - $b |
| \*= | Multiply then Assign | $a \*= $b | Multiplication same as $a = $a \* $b |
| /= | Divide then Assign (quotient) | $a /= $b | Find quotient same as $a = $a / $b |
| %= | Divide then Assign (remainder) | $a %= $b | Find remainder same as $a = $a % $b |

* Conditional assignment operators

The bitwise operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Explanation** |
| & | And | $a & $b | Bits that are 1 in both $a and $b are set to 1, otherwise 0. |
| | | Or (Inclusive or) | $a | $b | Bits that are 1 in either $a or $b are set to 1 |
| ^ | Xor (Exclusive or) | $a ^ $b | Bits that are 1 in either $a or $b are set to 0. |
| ~ | Not | ~$a | Bits that are 1 set to 0 and bits that are 0 are set to 1 |
| << | Shift left | $a << $b | Left shift the bits of operand $a $b steps |
| >> | Shift right | $a >> $b | Right shift the bits of $a operand by $b number of places |

* Comparison Operators

Comparison operators allow comparing two values, such as number or string. Below the list of comparison operators are given:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Explanation** |
| == | Equal | $a == $b | Return TRUE if $a is equal to $b |
| === | Identical | $a === $b | Return TRUE if $a is equal to $b, and they are of same data type |
| !== | Not identical | $a !== $b | Return TRUE if $a is not equal to $b, and they are not of same data type |
| != | Not equal | $a != $b | Return TRUE if $a is not equal to $b |
| <> | Not equal | $a <> $b | Return TRUE if $a is not equal to $b |
| < | Less than | $a < $b | Return TRUE if $a is less than $b |
| > | Greater than | $a > $b | Return TRUE if $a is greater than $b |
| <= | Less than or equal to | $a <= $b | Return TRUE if $a is less than or equal $b |
| >= | Greater than or equal to | $a >= $b | Return TRUE if $a is greater than or equal $b |
| <=> | Spaceship | $a <=>$b | Return -1 if $a is less than $b Return 0 if $a is equal $b Return 1 if $a is greater than $b |

* Incrementing/Decrementing Operators

The increment and decrement operators are used to increase and decrease the value of a variable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Explanation** |
| ++ | Increment | ++$a | Increment the value of $a by one, then return $a |
| $a++ | Return $a, then increment the value of $a by one |
| -- | decrement | --$a | Decrement the value of $a by one, then return $a |
| $a-- | Return $a, then decrement the value of $a by one |

* Logical Operators

The logical operators are used to perform bit-level operations on operands. These operators allow the evaluation and manipulation of specific bits within the integer.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Explanation** |
| And | And | $a and $b | Return TRUE if both $a and $b are true |
| Or | Or | $a or $b | Return TRUE if either $a or $b is true |
| Xor | Xor | $a xor $b | Return TRUE if either $ or $b is true but not both |
| ! | Not | ! $a | Return TRUE if $a is not true |
| && | And | $a && $b | Return TRUE if either $a and $b are true |
| || | Or | $a || $b | Return TRUE if either $a or $b is true |

* String Operators

The string operators are used to perform the operation on strings. There are two string operators in PHP, which are given below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Operator** | | **Name** | | **Example** | | **Explanation** |
| . | Concatenation | | $a . $b | | Concatenate both $a and $b | |
| .= | Concatenation and Assignment | | $a .= $b | | First concatenate $a and $b, then assign the concatenated string to $a, e.g. $a = $a . $b | |

* Array Operators

The array operators are used in case of array. Basically, these operators are used to compare the values of arrays.

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Explanation** |
| + | Union | $a + $y | Union of $a and $b |
| == | Equality | $a == $b | Return TRUE if $a and $b have same key/value pair |
| != | Inequality | $a != $b | Return TRUE if $a is not equal to $b |
| === | Identity | $a === $b | Return TRUE if $a and $b have same key/value pair of same type in same order |
| !== | Non-Identity | $a !== $b | Return TRUE if $a is not identical to $b |
| <> | Inequality | $a <> $b | Return TRUE if $a is not equal to $b |

# Conditional Statements

to write code that perform different actions based on the results of a logical or comparative test conditions at run time.

* The **if** statement
* The **if...else** statement
* The **if...elseif....else** statement
* The **switch...case** statement

## The if Statement

The if statement is used to execute a block of code only if the specified condition evaluates to true.

Syntax:

if(condition){  
    // Code to be executed  
}

Example:

output "Have a nice weekend!" if the current day is Friday:

<?php

$d = date("D");

if($d == "Fri"){

echo "Have a nice weekend!";

}

?>

## The if...else Statement

You can enhance the decision making process by providing an alternative choice through adding an else statement to the if statement. The if...else statement allows you to execute one block of code if the specified condition is evaluates to true and another block of code if it is evaluates to false. It can be written, like this:

Syntax:

if(condition){  
    // Code to be executed if condition is true  
} else{  
    // Code to be executed if condition is false  
}

Exaple:Output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!"

<?php

$d = date("D");

if($d == "Fri"){

echo "Have a nice weekend!";

} else{

echo "Have a nice day!";

}

?>

## The if...elseif...else Statement

The if...elseif...else a special statement that is used to combine multiple if...else statements.

Syntax:

if(condition1){  
    // Code to be executed if condition1 is true  
} elseif(condition2){  
    // Code to be executed if the condition1 is false and condition2 is true  
} else{  
    // Code to be executed if both condition1 and condition2 are false  
}

Example:

output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday, otherwise it will output "Have a nice day!"

<?php

$d = date("D");

if($d == "Fri"){

echo "Have a nice weekend!";

} elseif($d == "Sun"){

echo "Have a nice Sunday!";

} else{

echo "Have a nice day!";

}

?>

The switch-case statement is an alternative to the if-elseif-else statement, which does almost the same thing. The switch-case statement tests a variable against a series of values until it finds a match, and then executes the block of code corresponding to that match.

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

Consider the following example, which display a different message for each day.

#### Example

<?php

$today = date("D");

switch($today){

case "Mon":

echo "Today is Monday. Clean your house.";

break;

case "Tue":

echo "Today is Tuesday. Buy some food.";

break;

case "Wed":

echo "Today is Wednesday. Visit a doctor.";

break;

case "Thu":

echo "Today is Thursday. Repair your car.";

break;

case "Fri":

echo "Today is Friday. Party tonight.";

break;

case "Sat":

echo "Today is Saturday. Its movie time.";

break;

case "Sun":

echo "Today is Sunday. Do some rest.";

break;

default:

echo "No information available for that day.";

break;

}

?>

# Arrays

It is used to hold multiple values of similar type in a single variable.

Advantage of PHP Array

**Less Code**: We don't need to define multiple variables.

**Easy to traverse**: By the help of single loop, we can traverse all the elements of an array.

**Sorting**: We can sort the elements of array.

There are 3 types of array in PHP.

1. Indexed Array
2. Associative Array
3. Multidimensional Array

## Indexed Array

PHP index is represented by number which starts from 0. We can store number, string and object in the PHP array. All PHP array elements are assigned to an index number by default.

There are two ways to define indexed array:

$season=**array**("summer","winter","spring","autumn");

echo "Season are: $season[0], $season[1], $season[2] and $season[3]";

**OR**

$season[0]="summer";

$season[1]="winter";

$season[2]="spring";

$season[3]="autumn";

echo "Season are: $season[0], $season[1], $season[2] and $season[3]";

## Associative Array

We can associate name with each array elements in PHP using => symbol.

There are two ways to define associative array:

$salary=**array**("Sonoo"=>"350000","John"=>"450000","Kartik"=>"200000");

echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";

echo "John salary: ".$salary["John"]."<br/>";

echo "Kartik salary: ".$salary["Kartik"]."<br/>";

**OR**

$salary["Sonoo"]="350000";

$salary["John"]="450000";

$salary["Kartik"]="200000";

echo "Sonoo salary: ".$salary["Sonoo"]."<br/>";

echo "John salary: ".$salary["John"]."<br/>";

echo "Kartik salary: ".$salary["Kartik"]."<br/>";

# 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.

## Definition

$emp = **array**

  (

**array**(1,"sonoo",400000),

**array**(2,"john",500000),

**array**(3,"rahul",300000)

  );

**for** ($row = 0; $row < 3; $row++) {

**for** ($col = 0; $col < 3; $col++) {

    echo $emp[$row][$col]."  ";

  }

 echo "<br/>";

}

# Php Loops

the following loop types:

* 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

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

### Syntax

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

### Examples

The example below displays the numbers from 1 to 5:

### Example

<?php  
$x = 1;  
  
while($x <= 5) {  
  echo "The number is: $x <br>";  
  $x++;  
}  
?>

## 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*);

### Examples

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

### Example

<?php  
$x = 1;  
  
do {  
  echo "The number is: $x <br>";  
  $x++;  
} while ($x <= 5);  
?>

## for Loop

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

### Syntax

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

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

### Examples

The example below displays the numbers from 0 to 10:

### Example

<?php  
for ($x = 0; $x <= 10; $x++) {  
  echo "The number is: $x <br>";  
}  
?>

## foreach Loop

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

### Syntax

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.

### Examples

The following example will output the values of the given array ($colors):

### Example

<?php  
$colors = array("red", "green", "blue", "yellow");  
  
foreach ($colors as $value) {  
  echo "$value <br>";  
}  
?>

## Break

You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of a switch statement.

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

This example jumps out of the loop when **x** is equal to **4**:

### Example

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

## Continue

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

This example skips the value of **4**:

### Example

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