UNIT 4 - SERVER SIDE PROGRAMMING

Introduction

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- 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"

Features

- 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

PHP Syntax

A PHP script can be placed anywhere in the document.

A PHP script starts with <?php and ends with ?>:

```
<?php
// PHP code goes here
?>
```

Example:

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

<h1>My first PHP page</h1>

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

</body>
</html>
```

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

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.

PHP supports several ways of commenting:

```
// 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
*/
```

Variables

Variables are "containers" for storing information.

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)

Example1:

```
<?php
$txt = "Example";
$x = 2;
$y = 3.45;
?>
```

Example2:

```
<?php
$x = 5;
$y = 4;
echo $x + $y;
?>
```

Output: 9

Variables Scope

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 and Local Scope

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

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

global Keyword

The global keyword is used to access a global variable from within a function.

Example:

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

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

myTest();
echo $y;
?>
```

Output

```
15
```

static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, if you want a local variable NOT to be deleted, use the static keyword when you first declare the variable.

Example:

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

myTest();
myTest();
myTest();
?>
```

Output:

```
0

1
2
```

echo and print Statements

With 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.

Both echo and print statement can be used with or without parentheses: echo or echo(), print or print().

Example

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

<?php
$txt1 = "PHP";
$txt2 = "Example";

print  $txt1."</br>" ;
echo  $txt2;
?>

</body>
</html>
```

Output:

```
PHP
Example
```

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

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:

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

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

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

Example

```
        <?php
        $x = 5985;
        var_dump($x);
        ?>
```

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

Output: int(10)

Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

Example

```
<?php
$x = 10.365;
var_dump($x);
?>
```

Output: float(10.365)

Boolean

A Boolean represents two possible states: TRUE or FALSE.

Example

```
$x = true;
$y = false;
```

Array

An array stores multiple values in one single variable.

In the following example $cars is an array.

Example

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

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.

Example

```
$x = null;
```

Numerical Strings

The PHP is_numeric() function can be used to find whether a variable is numeric. The function returns true if the variable is a number or a numeric string, false otherwise.

```
<?php
$x = 5985;
var_dump(is_numeric($x));

$x = "5985";
var_dump(is_numeric($x));
?>
```

Output:

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

Math Functions

sqrt( ), pi( ), cos( ), sin( ), tan( ), log( ), floor( ), ceil( ), min( ), max( ), round( ), rand( ), pow( )

is_nan() function checks whether a value is 'not a number'. This function returns true (1) if the specified value is 'not-a-number', otherwise it returns false/nothing.

Example:

```
    <?php

    echo is_nan(200) . "<br>";

    echo(ceil(0.40) . "<br>");

    echo(max(2,4,6,8,10) . "<br>");

    echo(min(2,4,6,8,10) . "<br>");

    echo(rand() . "<br>");

    ?>
```

Output:

```
    1
    10
    2
    894007043
```

Constants

Constants are like variables except that once they are defined they cannot be changed or undefined. A valid constant name starts with a letter or underscore (no $ sign before the constant name).

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

Example:

```php
<?php
define("SHOW", "PHP CODE");
echo SHOW;
?>
```

Output: PHP CODE

PHP Operators

Operators are used to perform operations on variables and values.

Operators in PHP:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- Conditional assignment operators

Comparison operators

1. !== - Not identical

    $x !== $y - Returns true if $x is not equal to $y, or they are not of the same type

Example:

```
<?php
$x = 100;
$y = "100";

echo($x !== $y); // returns true because types are not equal
?>
```

Output : 1

2. <=> - 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.

Example:

```
<?php
$x = 5;
$y = 10;
echo ($x <=> $y); // returns -1 because $x is less than $y
echo "<br>";

$x = 10;
$y = 10;
echo ($x <=> $y); // returns 0 because values are equal
echo "<br>";

$x = 15;
$y = 10;
echo ($x <=> $y); // returns +1 because $x is greater than $y
?>
```

Output

```
1
0
1
```

String Operators

PHP has two operators that are specially designed for strings.

.	Concatenation
.=	Concatenation assignment

Example

```
<?php
$txt1 = "Hello";
$txt2 = " world!";
echo $txt1 . $txt2;
$txt1 .= $txt2;
```

```
echo $txt1;
?>
```

Output

```
Hello world!
Hello world!
```

Conditional assignment operators

1. ?: - Ternary

$x = *expr1* ? *expr2* : *expr3*

Returns the value of $x.
The value of $x is *expr2* if *expr1* = TRUE.
The value of $x is *expr3* if *expr1* = FALSE

Example:

```
<?php
        $a = 10;
        $b = 20;
        $result = ($a > $b ) ? $a :$b;
        echo “ Value of result is $result";
?>
```

Output:

```
Value of result is 20
```

2. ?? - Null coalescing

$x = *expr1* ?? *expr2*

Returns the value of $x.
The value of $x is *expr1* if *expr1* exists, and is not NULL.
If *expr1* does not exist, or is NULL, the value of $x is *expr2*.

Example:

```
<?php
  $a = 10;
  $b = 20;
  $result1 = $a ?? $b;
  $result2 = $c ?? $b;

  echo "Value of result1 is $result1";
  echo "Value of result1 is $result2";
?>
```

Output:

```
Value of result1 is 10
Value of result1 is 20
```

Control Statements

Conditional statements are used to perform different actions based on different conditions.

Conditional statements in PHP

- if
- if...else
- if...elseif...else
- nested if Statement
- switch

if Statement

If statement is used to executes the block of code exist inside the if statement only if the specified condition is true.

Syntax

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

Example

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

Output:

```
12 is less than 100
```

If-else Statement

If-else statement is slightly different from if statement. It executes one block of code if the specified condition is true and another block of code if the condition is false.

Syntax

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

Example:

```
<?php
$num=12;
if($num%2==0){
echo "$num is even number";
}else{
echo "$num is odd number";
}
?>
```

Output:

```
    12 is even number
```

if...elseif...else Statement

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;
}
```

Example

```
<?php
$t = date("H");
echo $t;
if ($t < "10") {
  echo "Have a good morning!";
} elseif ($t < "20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>
```

Output

```
10
Have a good day
```

nested if Statement

The nested if statement contains the if block inside another if block. The inner if statement executes only when specified condition in outer if statement is **true**.

Syntax

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

Example

```
<?php
               $age = 23;
    $nationality = "Indian";
    //applying conditions on nationality and age
    if ($nationality == "Indian")
    {
        if ($age >= 18) {
            echo "Eligible to give vote";
        }
        else {
            echo "Not eligible to give vote";
        }
    }
?>
```

Output

```
Eligible to give vote
```

Switch

Switch statement is used to execute one statement from multiple conditions. It works like PHP if-else-if statement.

Syntax

```
switch(expression){
case value1:
 //code to be executed
 break;
case value2:
 //code to be executed
 break;
......
default:
 code to be executed if all cases are not matched;
}
```

Example

```
<?php
$num=20;
switch($num){
case 10:
echo("number is equals to 10");
break;
case 20:
echo("number is equal to 20");
break;
case 30:
echo("number is equal to 30");
break;
default:
echo("number is not equal to 10, 20 or 30");
}
?>
```

Output

```
number is equal to 20
```

Looping Statements

PHP supports following four loop types.

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

for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

Syntax

```
for (initialization; condition; increment){
  code to be executed;
}
```

Example

```
<?php
for($n=1;$n<=5;$n++){
echo "$n<br/>";
}
?>
```

Output

```
1
2
3
4
5
```

while loop statement

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

Syntax

```
while (condition) {
  code to be executed;
}
```

Example

```
<?php
$x = 1;

while($x <= 5) {
  echo " $x <br>";
  $x++;
}
?>
```

Output

```
1
2
3
4
5
```

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);
```

Example

```
<?php
$x = 1;

do {
  echo " $x <br>";
  $x++;
} while ($x <= 5);
?>
```

Output

```
1
2
3
4
5
```

foreach Loop

The foreach loop works only on arrays, and is used to loop through each value 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.

Example

```
<?php
 $array = array( 1, 2, 3, 4, 5);

 foreach( $array as $value ) {
   echo "Value is $value <br />";
 }
?>
```

Output

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

break statement

The break keyword is used to terminate the execution of a loop prematurely.

continue statement

The PHP continue keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Arrays

An array is a data structure that stores one or more similar type of values in a single variable.

Creating an Array

In PHP, the array() function is used to create an array.

Example:

```
<?php
$numbers = array( 1, 2, 3);
echo count($cars);
?>
```

Output:

```
3
```

count() function is used to return the length (the number of elements) of an array.

There are three different kind of arrays and each array value is accessed using array index.

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

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.

Two ways to define indexed array

1.

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

2.

```
$season[0]="summer";
$season[1]="winter";
$season[2]="spring";
$season[3]="autumn";
```

Example:

```
<?php
$season=array("summer","winter","spring","autumn");
echo "Seasons are: $season[0], $season[1], $season[2] and $season[3]";
?>
```

Output

```
Seasons are summer, winter, spring, autumn
```

Associative Array

Associative array will have their index as string so that you can establish a strong association between key and values.

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

Example

```
<html>
  <body>

    <?php
      /* First method to associate create array. */
      $salaries = array("mohammad" => 2000, "qadir" => 1000, "zara" => 500);

      echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
      echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
      echo "Salary of zara is ".  $salaries['zara']. "<br />";

      /* Second method to create array. */
      $salaries['mohammad'] = "high";
      $salaries['qadir'] = "medium";
      $salaries['zara'] = "low";

      echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
      echo "Salary of qadir is ".  $salaries['qadir']. "<br />";
      echo "Salary of zara is ".  $salaries['zara']. "<br />";
    ?>

  </body>
</html>
```

Output

```
Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low
```

Multidimensional Arrays

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

Example1:

```
<?php
$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/>";
}
?>
```

Output

```
1 sonoo 400000
2 john 500000
3 rahul 300000
```

Example2:

```
<html>
  <body>

    <?php
      $marks = array(
        "mohammad" => array (
          "physics" => 35,
            "maths" => 30,
            "chemistry" => 39
         ),

         "qadir" => array (
            "physics" => 30,
            "maths" => 32,
            "chemistry" => 29
         ),

         "zara" => array (
            "physics" => 31,
            "maths" => 22,
            "chemistry" => 39
         )
      );

      /* Accessing multi-dimensional array values */
      echo "Marks for mohammad in physics : " ;
      echo $marks['mohammad']['physics'] . "<br />";

      echo "Marks for qadir in maths : ";
      echo $marks['qadir']['maths'] . "<br />";

      echo "Marks for zara in chemistry : " ;
      echo $marks['zara']['chemistry'] . "<br />";
   ?>

  </body>
</html>
```

Output

```
Marks for mohammad in physics : 35
Marks for qadir in maths : 32
Marks for zara in chemistry : 39
```

Strings

String is a sequence of characters i.e., used to store and manipulate text.

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
$str1='Hello text within single quote';
$str2='Using double "quote" directly inside single quoted string';
$str3="Hello text within double quote";
```

Example

```
<?php
$str1="Hello text
multiple line
text within double quoted string";
$str2="Using double \"quote\" with backslash inside double quoted 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
```

String Functions

PHP provides various string functions to access and manipulate strings.

1. strlen()

strlen() function returns the length of a string.

Example

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

Output

```
12
```

2. str_word_count() - Count Words in a String

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

Example

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

Output

```
 2
```

3.strrev()

strrev() function reverses a string.

Example

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

Output

```
!dlrow olleH
```

4. strpos()

strpos() function searches for a specific text within a string. If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.

Example

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

Output

```
6
```

5. str_replace()

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

Example

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

Output

```
Hello Dhoni!
```

6. strtolower()

returns string in lowercase letter.

Example

```
<?php
$str="PHP";
$str=strtolower($str);
echo $str;
?>
```

Output

```
php
```

7.strtoupper()

Function returns string in uppercase letter.

Example

```
<?php
$str="php";
$str=strtoupper($str);
echo $str;
?>
```

Output

```
PHP
```

8. ucfirst()

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

Example

```
<?php
$str="example ";
$str=ucfirst($str);
echo $str;
?>
```

Output

```
Example
```

9. lcfirst()

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

Example

```
<?php
$str="Example ";
$str=ucfirst($str);
echo $str;
?>
```

Output

```
example
```

10.ucwords()

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

Example

```
<?php
$str="string function example";
$str=ucwords($str);
echo $str;
?>
```

Output

```
String Function Example
```

Number Functions

PHP has the following functions to check if the type of a variable is integer.

- is_int()
- is_integer() - alias of is_int()
- is_long() - alias of is_int()

Example

```
<?php
$x = 5985;
var_dump(is_int($x));

$x = 59.85;
var_dump(is_int($x));
?>
```

Output

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

Functions to check if the type of a variable is float:

- is_float()
- is_double() - alias of is_float()

Example

```
<?php
$x = 10.365;
var_dump(is_float($x));
?>
```

Output

```
bool(true)
```

NaN

NaN stands for Not a Number.

NaN is used for impossible mathematical operations.

PHP has the following functions to check if a value is not a number:

- is_nan()

Example

```
<?php
$x = acos(8);
var_dump($x);
?>
```

Output

```
float(NAN)
```

Numerical Strings

is_numeric() function can be used to find whether a variable is numeric. The function returns true if the variable is a number or a numeric string, false otherwise.

Casting Strings and Floats to Integers

Sometimes you need to cast a numerical value into another data type.

The (int), (integer), or intval() function are often used to convert a value to an integer.

Example

```
<?php
// Cast float to int
$x = 23465.768;
$int_cast = (int)$x;
echo $int_cast;

echo "<br>";

// Cast string to int
$x = "23465.768";
$int_cast = (int)$x;
echo $int_cast;
?>
```

Output

```
23465
23465
```

FUNCTIONS

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.

Creating a User Defined Function

A user-defined function declaration starts with the word function.

Syntax

```
function functionName() {
  code to be executed;
}
```

A function name must start with a letter or an underscore. Function names are NOT case-sensitive.

Example

```
<?php
function writeMsg() {
  echo "Example for Function";
}

writeMsg();
?>
```

Output

```
Example for Function
```

Function Arguments

PHP supports Call by Value ,Call by Reference, Default argument values and Variable-length argument list.

1.Call by Value

Example:

```
<?php
      function addFunction($num1, $num2) {
        $sum = $num1 + $num2;
        echo "Sum of the two numbers is : $sum";
      }

      addFunction(10, 20);
    ?>
```

Output

```
Sum of the two numbers is : 30
```

2.Call by Reference

In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed.

When a function argument is passed by reference, changes to the argument also change the variable that was passed in. To turn a function argument into a reference, the & operator is used.

Example

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

Output

```
Hello Call By Reference
```

3.Default Argument Value

We can specify a default argument value in function. While calling the function if you don't specify any argument, it will take the default argument.

Example

```
<html>
  <head>
    <title>Writing PHP Function which returns value</title>
  </head>
  <body>
    <?php
      function printMe($param = NULL) {
        print $param;
      }

      printMe("This is test");
      printMe();
    ?>
  </body>
</html>
```

Output:

```
This is test
```

Functions returning value

A function can return a value using the return statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using return array(1,2,3,4).

Example 1:

```
<html>

  <head>
    <title>Writing PHP Function which returns value</title>
  </head>

  <body>

    <?php
      function addFunction($num1, $num2) {
        $sum = $num1 + $num2;
        return $sum;
      }
      $return_value = addFunction(10, 20);

      echo "Returned value from the function : $return_value";
    ?>

  </body>
</html>
```

Output:

```
30
```

Example 2:

```
<html>

  <head>
    <title>Writing PHP Function which returns value</title>
  </head>

  <body>

<?php
function addNumbers(int $a, int $b) {
  return array($a,$b,$a + $b);
}
$s= addNumbers(5, 4);
```

```
foreach($s as $v){
  echo "$v <br>";
}

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

Output:

```
5
4
9
```

Example 3:

```
<html>

   <head>
      <title>Writing PHP Function which returns value</title>
   </head>

   <body>

     <?php
function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "10 dys");

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

Output:

```
15
```

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives us an option to specify the expected data type when declaring a function, and by adding the strict declaration, it will throw a "Fatal Error" if the data type mismatches.

Example 4:

```
<?php declare(strict_types=1); // strict requirement

function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is enabled and "5 days" is not an integer, an error will be thrown
?>
```

Output:

```
Uncaught TypeError: Argument 2 passed to addNumbers() must be of the type integer, string given
```

Dynamic Function Calls

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

Example

```
<html>

  <head>
    <title>Dynamic Function Calls</title>
  </head>

  <body>

    <?php
      function greet() {
        echo "Hello <br />";
      }

      $function_holder = "greet";
      $function_holder();
    ?>

  </body>
</html>
```

Output:

```
Hello
```

FORM VALIDATION

Validation means check the input submitted by the user. There are two types of validation are available in PHP. They are as follows −

- Client-Side Validation − Validation is performed on the client machine web browsers.
- Server Side Validation − After submitted by data, The data has sent to a server and perform validation checks in server machine.

Filter

The filter_var() function filters a single variable with a specified filter. It takes two pieces of data:

- The variable you want to check
- The type of check to use

The filter_var() function both validate and sanitize data.

Sanitize a String

The following example uses the filter_var() function to remove all HTML tags from a string:

```
<?php
$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str, FILTER_SANITIZE_STRING);
echo $newstr;
?>
```

Output: Hello World!

Validate an Integer

The filter_var() function checks if the variable $int is an integer. If $int is an integer, the output of the code below will be: "Integer is valid". If $int is not an integer, the output will be: "Integer is not valid":

Example

```
<?php
$int = 100;

if (!filter_var($int, FILTER_VALIDATE_INT) === false) {
  echo("Integer is valid");
} else {
  echo("Integer is not valid");
}
?>
```

Output : Integer is valid

Validate an IP Address

To check if the given value is a valid IP address.

Example

```
<?php
$ip = "127.0.0.1";

if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {
  echo("$ip is a valid IP address");
} else {
  echo("$ip is not a valid IP address");
}
?>
```

Output : 127.0.0.1 is a valid IP address

Sanitize and Validate an Email Address

The filter_var() function will first remove all illegal characters from the $email variable, then check if it is a valid email address.

Example

```
<?php
$email = "kohli@bcci.com";

// Remove all illegal characters from email
$email = filter_var($email, FILTER_SANITIZE_EMAIL);

// Validate e-mail
if (!filter_var($email, FILTER_VALIDATE_EMAIL) === false) {
  echo("$email is a valid email address");
} else {
  echo("$email is not a valid email address");
}
?>
```

Output: kohli@bcci.com is a valid email address

Sanitize and Validate a URL

The filter_var() function will first remove all illegal characters from a URL, then check if $url is a valid URL.

Example

```
<?php
$url = "https://www.abc.com";

// Remove all illegal characters from a url
$url = filter_var($url, FILTER_SANITIZE_URL);
```

```
// Validate url
if (!filter_var($url, FILTER_VALIDATE_URL) === false) {
  echo("$url is a valid URL");
} else {
  echo("$url is not a valid URL");
}
?>
```

Output: https://www.abc.com is a valid URL

Form Element

The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.

The $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

$_POST is a PHP super global variable which is used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

htmlspecialchars() function

The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Example

```
<html>

  <head>
    <style>
      .error {color: #FF0000;}
    </style>
  </head>

  <body>
    <?php
      // define variables and set to empty values
      $nameErr = $emailErr = $genderErr = $websiteErr = "";
      $name = $email = $gender = $comment = $website = "";

      if ($_SERVER["REQUEST_METHOD"] == "POST") {
        if (empty($_POST["name"])) {
           $nameErr = "Name is required";
        }else {
           $name = test_input($_POST["name"]);
        }

        if (empty($_POST["email"])) {
           $emailErr = "Email is required";
        }else {
           $email = test_input($_POST["email"]);

           // check if e-mail address is well-formed
           if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
              $emailErr = "Invalid email format";
           }
        }

        if (empty($_POST["website"])) {
           $website = "";
        }else {
           $website = test_input($_POST["website"]);
        }

        if (empty($_POST["comment"])) {
           $comment = "";
        }else {
           $comment = test_input($_POST["comment"]);
        }

        if (empty($_POST["gender"])) {
           $genderErr = "Gender is required";
        }else {
           $gender = test_input($_POST["gender"]);
        }
      }

      function test_input($data) {
        $data = trim($data);
        $data = stripslashes($data);
        $data = htmlspecialchars($data);
        return $data;
      }
    ?>

    <h2>Registration Form</h2>

    <p><span class = "error">* required field.</span></p>

    <form method = "post" action = "<?php
      echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">
      <table>
```

```
    <tr>
      <td>Name:</td>
      <td><input type = "text" name = "name">
        <span class = "error">* <?php echo $nameErr;?></span>
      </td>
    </tr>

    <tr>
      <td>E-mail: </td>
      <td><input type = "text" name = "email">
        <span class = "error">* <?php echo $emailErr;?></span>
      </td>
    </tr>

    <tr>
      <td>Website:</td>
      <td> <input type = "text" name = "website">
        <span class = "error"><?php echo $websiteErr;?></span>
      </td>
    </tr>

    <tr>
      <td>Comment:</td>
      <td> <textarea name = "comment" rows = "5" cols = "40"></textarea></td>
    </tr>

    <tr>
      <td>Gender:</td>
      <td>
        <input type = "radio" name = "gender" value = "female">Female
        <input type = "radio" name = "gender" value = "male">Male
        <span class = "error">* <?php echo $genderErr;?></span>
      </td>
    </tr>

    <td>
      <input type = "submit" name = "submit" value = "Submit">
    </td>

  </table>

</form>

<?php
  echo "<h2>Your given values are as:</h2>";
  echo $name;
  echo "<br>";

  echo $email;
  echo "<br>";
```

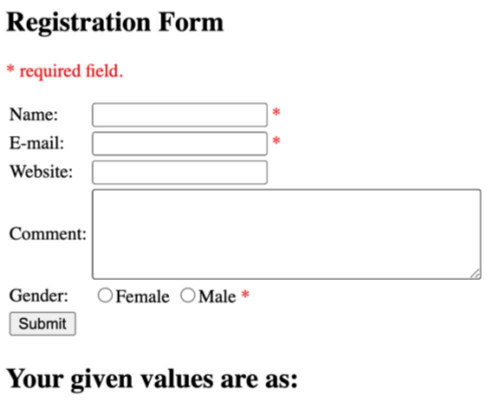
```
            echo $website;
            echo "<br>";

            echo $comment;
            echo "<br>";

            echo $gender;
          ?>

        </body>
      </html>
```

Output:



Registration Form

* required field.

Name:	*
E-mail:	*
Website:	
Comment:	
Gender:	○Female ○Male *

Your given values are as:

REGULAR EXPRESSIONS

A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.

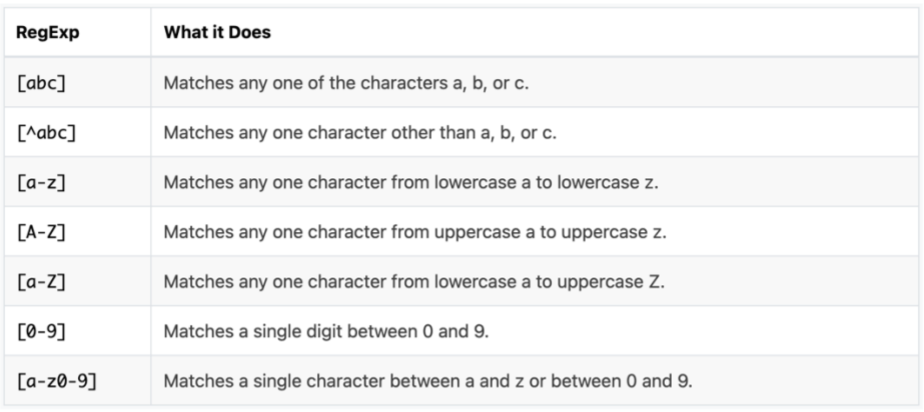
A regular expression can be a single character, or a more complicated pattern. Regular expressions can be used to perform all types of text search and text replace operations.

In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers.

Character Classes

Square brackets surrounding a pattern of characters are called a character class e.g. [abc]. A character class always matches a single character out of a list of specified characters that means the expression [abc] matches only a, b or c character.

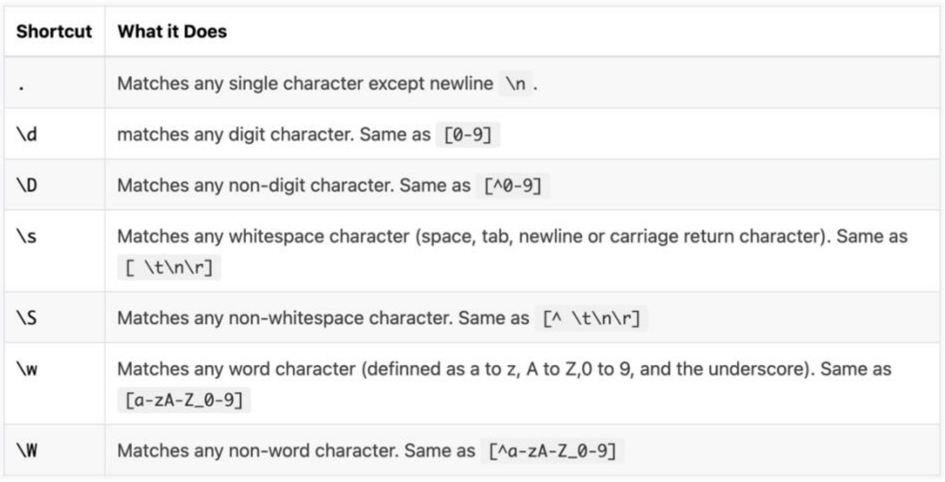
Negated character classes can also be defined that match any character except those contained within the brackets. A negated character class is defined by placing a caret (^) symbol immediately after the opening bracket, like this [^abc].



RegExp	What it Does
[abc]	Matches any one of the characters a, b, or c.
[^abc]	Matches any one character other than a, b, or c.
[a-z]	Matches any one character from lowercase a to lowercase z.
[A-Z]	Matches any one character from uppercase a to uppercase z.
[a-Z]	Matches any one character from lowercase a to uppercase Z.
[0-9]	Matches a single digit between 0 and 9.
[a-z0-9]	Matches a single character between a and z or between 0 and 9.

Predefined Character Classes

Some character classes such as digits, letters, and whitespaces are used so frequently that there are shortcut names for them. The following table lists those predefined character classes:

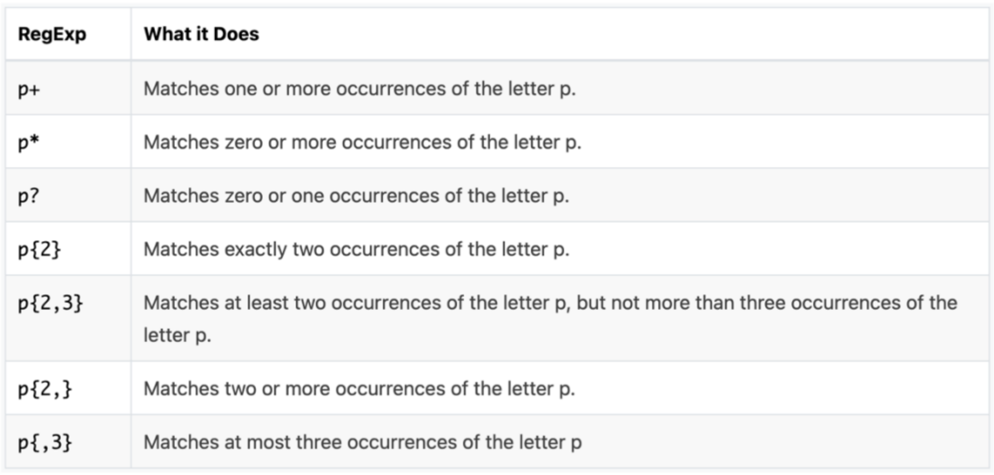


Shortcut	What it Does
.	Matches any single character except newline `\n` .
\d	matches any digit character. Same as `[0-9]`
\D	Matches any non-digit character. Same as `[^0-9]`
\s	Matches any whitespace character (space, tab, newline or carriage return character). Same as `[ \t\n\r]`
\S	Matches any non-whitespace character. Same as `[^ \t\n\r]`
\w	Matches any word character (definned as a to z, A to Z,0 to 9, and the underscore). Same as `[a-zA-Z_0-9]`
\W	Matches any non-word character. Same as `[^a-zA-Z_0-9]`

Quantifiers

With quantifiers you can specify how many times a character in a regular expression should match.

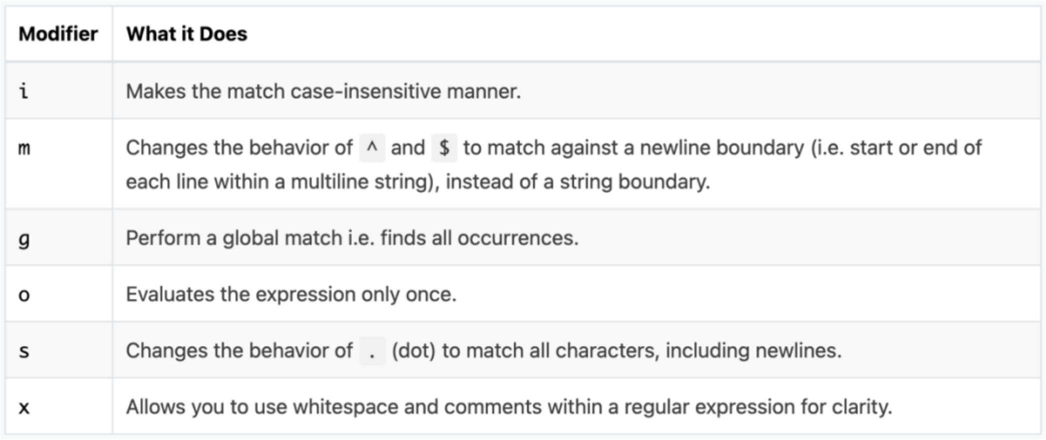
The following table lists the various ways to quantify a particular pattern:



RegExp	What it Does
p+	Matches one or more occurrences of the letter p.
p*	Matches zero or more occurrences of the letter p.
p?	Matches zero or one occurrences of the letter p.
p{2}	Matches exactly two occurrences of the letter p.
p{2,3}	Matches at least two occurrences of the letter p, but not more than three occurrences of the letter p.
p{2,}	Matches two or more occurrences of the letter p.
p{,3}	Matches at most three occurrences of the letter p

Modifiers

A pattern modifier allows you to control the way a pattern match is handled. Pattern modifiers are placed directly after the regular expression, for example, if you want to search for a pattern in a case-insensitive manner, you can use the i modifier, like this: /pattern/i. The following table lists some of the most commonly used pattern modifiers.



Modifier	What it Does
i	Makes the match case-insensitive manner.
m	Changes the behavior of `^` and `$` to match against a newline boundary (i.e. start or end of each line within a multiline string), instead of a string boundary.
g	Perform a global match i.e. finds all occurrences.
o	Evaluates the expression only once.
s	Changes the behavior of `.` (dot) to match all characters, including newlines.
x	Allows you to use whitespace and comments within a regular expression for clarity.

Commonly used PHP's built-in pattern-matching functions

- preg_match()
- preg_match_all()
- preg_replace()

Example 1:

The preg_match() function will tell you whether a string contains matches of a pattern. Returns 1 if the pattern was found in the string and 0 if not.

```
<?php
$str = "Exp Example";
$pattern = "/example/i";
echo preg_match($pattern, $str);
?>
```

Output: 1

Example 2:

The preg_match_all() function will tell you how many matches were found for a pattern in a string. Returns the number of times the pattern was found in the string, which may also be 0.

```
<?php
$str = "The rain in SPAIN falls mainly on the plains.";
$pattern = "/ain/i";
echo preg_match_all($pattern, $str);
?>
```

Output: 4

Example 3:

The preg_replace() function will replace all of the matches of the pattern in a string with another string. Returns a new string where matched patterns have been replaced with another string.

```
<?php
$str = "Expression Demo";
$pattern = "/demo/i";
echo preg_replace($pattern, "Example", $str);
?>
```

Output: Expression Example

FILE HANDLING

PHP is a server side programming language, it allows you to work with files and directories stored on the web server.

File operations

- Opening a file
- Reading a file
- Writing a file
- Closing a file

Opening and Closing Files

fopen() function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Syntax

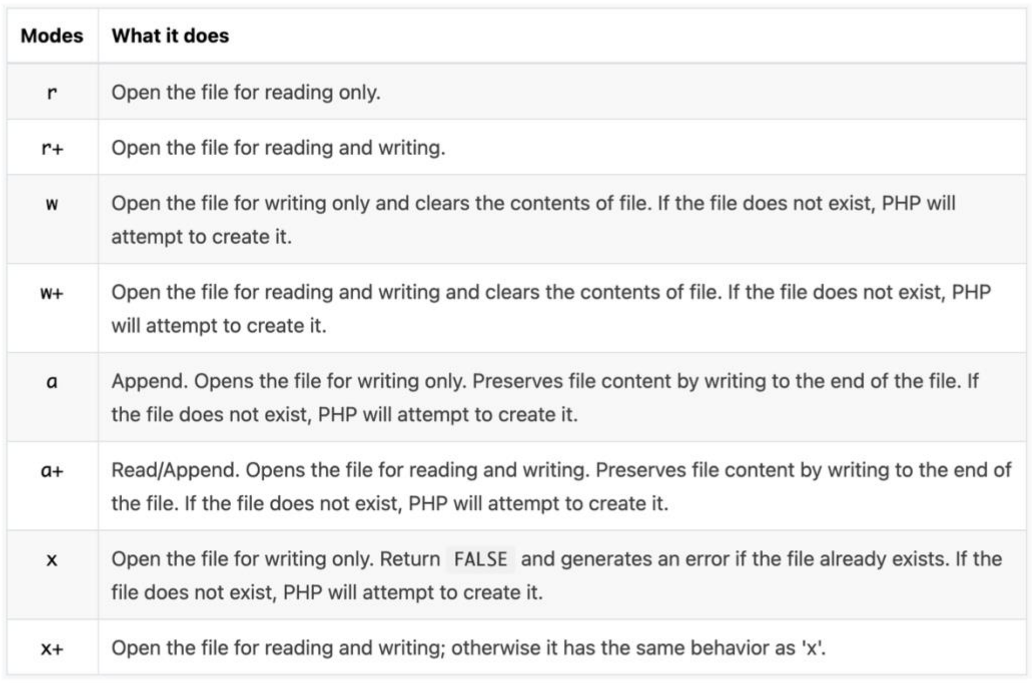
```
fopen(filename, mode)
```

fclose() function is used to close the file.

Example

```
<?php
$handle = fopen("data.txt", "r");
fclose($handle);
?>
```

File Modes



Modes	What it does
r	Open the file for reading only.
r+	Open the file for reading and writing.
w	Open the file for writing only and clears the contents of file. If the file does not exist, PHP will attempt to create it.
w+	Open the file for reading and writing and clears the contents of file. If the file does not exist, PHP will attempt to create it.
a	Append. Opens the file for writing only. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it.
a+	Read/Append. Opens the file for reading and writing. Preserves file content by writing to the end of the file. If the file does not exist, PHP will attempt to create it.
x	Open the file for writing only. Return `FALSE` and generates an error if the file already exists. If the file does not exist, PHP will attempt to create it.
x+	Open the file for reading and writing; otherwise it has the same behavior as 'x'.

If you try to open a file that doesn't exist, PHP will generate a warning message. So, to avoid these error messages you should always implement a simple check whether a file or directory exists or not before trying to access it, with the PHP file_exists() function.

Example:

```
<?php
$file = "data.txt";

// Check the existence of file
if(file_exists($file)){
    echo "The file $file exists." . "<br>";

    // Attempt to open the file
    $handle = fopen($file, "r") or die("ERROR: Cannot open the file.");
    if($handle){
        echo "File opened successfully.";

        // Closing the file handle
        fclose($handle);
    }
} else{
   echo "ERROR: The file $file does not exist.";
}
?>
```

Reading from Files

a.Using **fread()** Function

We can read content from an already open file using the fread() function. The fread() function takes two arguments:

1. first is the filename
2. and the second argument specifies the size in bytes for the content to be read.

Example:

```
<?php
// Opening a file
$myfile = fopen("abc.txt", "r");

// reading the entire file using
// fread() function
echo fread($myfile, filesize("abc.txt"));

// closing the file
fclose($myfile);
?>
```

Output

```
C Language
C++
Data Structure
```

b.Using **readfile()**

The readfile() function reads the entire content of a file and writes it to the output buffer.

Example:

```
<?php
echo readfile("studytonight-info.txt");
?>
```

Output

```
Text from file
```

Read File Line by Line using fgets()

The fgets() function reads a single line(till the new line character) from any given file.

Example

```
<?php
// Opening a file
$myfile = fopen("studytonight-info.txt", "r");

// reading a single line using fgets()
echo fgets($myfile);

// closing the file
fclose($myfile);
?>
```

Output

```
Text from file
```

After a call to the fgets() function the file pointer moves to the next line, so if we call the fgets() function twice, we will get two lines from the file.

Get End of File using feof()

The function feof() returns true if the file pointer is at the end of file, else it returns false.

This function can be used to loop through file of unknown size because feof() function can be used to check if the end-of-file is reached.

Example

```
<?php
// Opening a file
$myfile = fopen("studytonight-info.txt", "r");

// loop around the file to output the content
// line by line
while(!feof($myfile)) {
   echo fgets($myfile) . "<br>";
}

// closing the file
fclose($myfile);
?>
```

Output

```
C Language
C++
Data Structure
```

Read File Character by Character - fgetc()

We can use the function fgetc() to read single character from any file resource starting from the beginning.

Write to a File

fwrite() function is used to write content to a file when a file is already open in write mode.

Example

```
<?php
$file_name = 'movies.txt';
//opens the file.txt file or implicitly creates the file
$myfile = fopen($file_name, 'w');
$movie_name = "Interstellar \n";
// write name to the file
fwrite($myfile, $movie_name);

// lets write another movie name to our file
$movie_name = "Lord of the rings \n";
fwrite($myfile, $movie_name);
// close the file
fclose($myfile);
?>
```

If we open the file, it will look like following:

```
Interstellar
Lord of the rings
```

When a file is opened in write mode, all the existing data in the file is erased and new data can be written to the file using the fwrite() function.

Append

If we wish to add more movie names to the file movies.txt then we need to open the file in append mode.

Example

```
<?php
$file_name = 'movies.txt';

//opens the file.txt file or implicitly creates the file
$myfile = fopen($file_name, 'a');
$movie_name = "Avengers \n";

// write name to the file
fwrite($myfile, $movie_name);

// close the file
fclose($myfile);
?>
```

If we open the file, it will look like following:

```
Interstellar
Lord of the rings
Avengers
```

Delete File

In PHP, we can delete any file using unlink() function. The unlink() function accepts one argument only: file name.

PHP unlink() generates E_WARNING level error if file is not deleted. It returns TRUE if file is deleted successfully otherwise FALSE.

Example

```
<?php
$status=unlink('data.txt');
if($status){
echo "File deleted successfully";
}else{
echo "Sorry!";
}
?>
```

STATE MANAGEMENT

HTTP is a stateless protocol which means every user request is processed independently and it has nothing to do with the requests processed before it. Hence there is no way to store or send any user specific details using HTTP protocol.

But in modern applications, user accounts are created and user specific information is shown to different users, for which we need to have knowledge about who the user(or what he/she wants to see etc) is on every webpage.

PHP provides for two different techniques for state management of your web application, they are:

1. Server Side State Management
2. Client Side Server Management

Server Side State Management

In server side state management we store user specific information required to identify the user on the server. And this information is available on every webpage.

In PHP we have Sessions for server side state management. PHP session variable is used to store user session information like username, userid etc and the same can be retrieved by accessing the session variable on any webpage of the web application until the session variable is destroyed.

Client Side State Management

In client side state management the user specific information is stored at the client side i.e. in the bowser. Again, this information is available on all the webpages of the web application.

In PHP we have Cookies for client side state management. Cookies are saved in the browser with some data and expiry date(till when the cookie is valid).

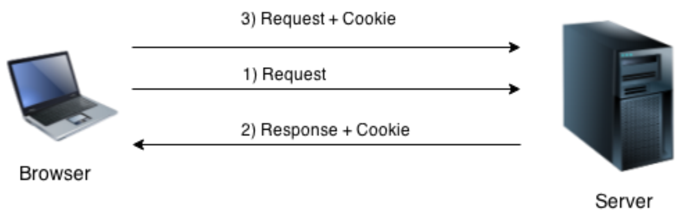
One drawback of using cookie for state management is the user can easily access the cookie stored in their browser and can even delete it.

COOKIES

Cookie is a small piece of information which is stored at client browser. It is used to recognize the user. Cookie is created at server side and saved to client browser.

Steps

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.



Use of Cookies

- To store user information like when he/she visited, what pages were visited on the website etc, so that next time the user visits your website you can provide a better user experience.
- To store basic website specific information to know this is not the first visit of user.
- You can use cookies to store number of visits or view counter.

Types of Cookies

There are two types of cookies, they are:

1. Session Cookie: This type of cookies are temporary and expire as soon as the session ends or the browser is closed.
2. Persistent Cookie: To make a cookie persistent we must provide it with an expiration time. Then the cookie will only expire after the given expiration time, until then it will be a valid cookie.

Creating a Cookie

setcookie() function is used to set cookie with HTTP response. Once cookie is set, you can access it by $_COOKIE superglobal variable.

Syntax

setcookie(*name, value, expire, path, domain, secure*);

name	Used to specify the name of the cookie. It is a mandatory argument. Name of the cookie must be a string.
value	Used to store any value in the cookie. It is generally saved as a pair with name. For example, name is userid and value is 7007, the userid for any user.
expire	Used to set the expiration time for a cookie. if you do not provide any value, the cookie will be treated as a session cookie and will expire when the browser is closed.
path	Used to set a web URL in the cookie. If set, the cookie will be accessible only from that URL. To make a cookie accessible through a domain, set '/' as cookie path.
domain	The domain of your web application. It can be used to limit access of cookie for sub-domains. For example, if you set the domain value as wwww.abc.com, then the cookie will be inaccessible from xyz.abc.com
secure	If you set this to 1, then the cookie will be available and sent only over HTTPS connection.

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(isset($_COOKIE[$cookie_name])) {
  echo "Cookie named '" . $cookie_name . "' is set.";
  echo "Value is: " . $_COOKIE[$cookie_name];
} else {
  echo "Cookie '" . $cookie_name . "' is  not set!<br>";
}
?>

</body>
</html>
```

Output

```
Cookie named 'user' is set.
Value is: John
```

The isset() method is used to check whether the cookie is set or not. The setcookie() function must appear before the <html> tag.

Updating Cookie

To update/modify a cookie, simply set it again. For example, if we want to update the username stored in the cookie created above, we can do it using setcookie() method again.

Deleting a Cookie

To delete/remove a cookie, we need to expire the cookie, which can be done by updating the cookie using the setcookie() function with expiration date in past.

Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

Output

```
Cookie 'user' is deleted.
```

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable.

Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
  echo "Cookies are enabled.";
} else {
  echo "Cookies are disabled.";
}
?>

</body>
</html>
```

Output

```
Cookies are enabled.
```

SESSIONS

When you log into your facebook account, by providing your email address and password, until and unless you logout, the web application remembers who you are and display what your friends are posting and liking on your News Feed, you can update your profile, send someone message, join a group etc, this is accomplished by Session.

PHP session is used to store and pass information from one page to another temporarily(until user close the website).

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

Starting a Session

A PHP session is easily started by making a call to the session_start() function.This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to session_start() at the beginning of the page.

Session variables are stored in associative array called $_SESSION[]. These variables can be accessed during lifetime of a session.

Example

```
<?php
  session_start();

  if( isset( $_SESSION['counter'] ) ) {
    $_SESSION['counter'] += 1;
  }else {
    $_SESSION['counter'] = 1;
  }

  $msg = "You have visited this page ".  $_SESSION['counter'];
  $msg .= "times in this session.";
?>

<html>

  <head>
    <title>Setting up a PHP session</title>
  </head>

  <body>
    <?php  echo ( $msg ); ?>
  </body>

</html>
```

The example starts a session then register a variable called counter that is incremented each time the page is visited during the session.

Output

```
You have visited this page 1 times in this session.
```

Destroying a Session

A PHP session can be destroyed by session_destroy() function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use unset() function to unset a session variable.

Unset a single variable

```
<?php
  unset($_SESSION['counter']);
?>
```

Destroy all the session variables

```
<?php
  session_destroy();
?>
```

CONNECTIVITY

Introduction to MySQL Database

- MySQL is an open-source relational database management system
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use

Opening a Connection to MySQL

Before we can access data in the MySQL database, we need to be able to connect to the server.

Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);

// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}
```

```
echo "Connected successfully";
?>
```

Closing the Connection

```
mysqli_close($conn);
```

Creating a MySQL Database

The CREATE DATABASE statement is used to create a database in MySQL.

Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Creating a Table

The CREATE TABLE statement is used to create a table in MySQL.

Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

// sql to create table
$sql = "CREATE TABLE Student (
id INT(6) AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50))";

if (mysqli_query($conn, $sql)) {
  echo "Table Student created successfully";
} else {
  echo "Error creating table: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Inserting Data

After a database and a table have been created, we can start adding data in them.

Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

The INSERT INTO statement is used to add new records to a MySQL table.

Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO Student (firstname, lastname, email)
VALUES (‘Raj’, 'Kumar', 'raj@gmail.com')";

if (mysqli_query($conn, $sql)) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Inserting Multiple Records

Multiple SQL statements must be executed with the mysqli_multi_query() function.

Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO Student (firstname, lastname, email)
VALUES ('Raj', 'Kumar', 'raj@gmail.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Virat', 'Kohli', 'virat@gmail.com');";

if (mysqli_multi_query($conn, $sql)) {
  echo "New records created successfully";
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn);
?>
```

Prepared Statements

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

Compared to executing SQL statements directly, prepared statements have three main advantages:

- Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
- Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
- Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

Example:

```
<?php

$link = mysqli_connect("localhost", "root", "", "demo");

// Check connection
if($link === false)
{
        die("ERROR: Could not connect. " . mysqli_connect_error());
}

// Prepare an insert statement
$sql = "INSERT INTO student (first_name, last_name, email) VALUES (?, ?, ?)";

if($stmt = mysqli_prepare($link, $sql)){

        // Bind variables to the prepared statement as parameters
        mysqli_stmt_bind_param($stmt, "sss", $first_name, $last_name, $email);

        /* Set the parameters values and execute the statement again to insert another row */
        $first_name = "Dhoni";
        $last_name = "MS";
        $email = "msd@gmail.com";
        mysqli_stmt_execute($stmt);
        /* Set the parameters values and execute the statement to insert a row */
        $first_name = "Kapil";
        $last_name = "Dev";
        $email = "kapil@gmail.com";
        mysqli_stmt_execute($stmt);
        echo "Records inserted successfully.";
}
else{
        echo "ERROR: Could not prepare query: $sql. " . mysqli_error($link);
}
// Close statement mysqli_stmt_close($stmt);
// Close connection
 mysqli_close($link);
?>
```

Inside the SQL INSERT statement (line no-12) of the example above, the question marks is used as the placeholders for the first_name, last_name, email fields values.

The mysqli_stmt_bind_param() function (line no-16) bind variables to the placeholders (?) in the SQL statement template. The placeholders (?) will be replaced by the actual values held in the variables at the time of execution. The type definition string provided as second argument i.e. the "sss" string specifies that the data type of each bind variable is string.

The type definition string specify the data types of the corresponding bind variables and contains one or more of the following four characters:

- b — binary (such as image, PDF file, etc.)
- d — double (floating point number)
- i — integer (whole number)
- s — string (text)

The number of bind variables and the number of characters in type definition string must match the number of placeholders in the SQL statement template.

Select Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
  die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM Student";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
  // output data of each row
  while($row = mysqli_fetch_assoc($result)) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
  }
}

else {
  echo "0 results";
}
```

```
mysqli_close($conn);
?>
```

First, we set up an SQL query that selects the id, firstname and lastname columns from the table. The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the function mysqli_num_rows() checks if there are more than zero rows returned.

If there are more than zero rows returned, the function mysqli_fetch_assoc() puts all the results into an associative array that we can loop through. The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.