
php

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 Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today.
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource
- PHP runs efficiently on the server side

Basic PHP Syntax

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

Example

```
<!DOCTYPE html>
<html>
    <body>
        <h1>My first PHP page</h1>
        <?php
                echo "Hello World!";
        ?>
    </body>
</html>
```

Comments in PHP

//	- This is a single-line comment
#	- This is also a single-line comment
/* ——————— */	- This is a multiple-lines comment block

PHP Case Sensitivity

- In PHP, NO keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are case-sensitive.
- Eg:
  - *ECHO "Hello Manipal*
  - *echo "Hello Manipal";*
  - *EcHo "Hello Manipal";*

PHP Variables

- Variables are "containers" for storing information.
- Creating (Declaring) PHP Variables
  - *All variables in PHP are denoted with a leading dollar sign ($).*
  - *Eg:* <?php

```
$txt = "Hello world!";
$x = 5;
$y = 10.5;

?>
```

- **Note:** When you assign a text value to a variable, put quotes around the value.
- **Note:** Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

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)

PHP Variables(Cont..)

- Output Variables
  - *The PHP echo statement is often used to output data to the screen.*
  - *Eg:*
```
<?php
    $txt = “Manipal";
    echo “Welcome to $txt !";
?>
```
- The below example will produce the same output as the example above:
  - *Eg:*
```
<?php
    $txt = “Manipal";
    echo “Welcome to” . $txt . “!";
?>
```

PHP Variables Scope

- In PHP, variables can be declared anywhere in the script.
- The scope of a variable is the part of the script where the variable can be referenced/used.
- PHP has three different variable scopes:
  - *local*
  - *global*
  - *static*

PHP Variables Scope (cont..)

- A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function
  - *The global keyword is used to access a global variable from within a function.*
  - *To do this, use the global keyword before the variables (inside the function)*
  - PHP also stores all global variables in an array called $GLOBALS[*index*].
  - The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.
- A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function
- Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. To do this, use the static keyword when you first declare the variable

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*
  - *echo can take multiple parameters (although such usage is rare)*
  - *echo is marginally faster than print.*
  - *print has a return value of 1 so it can be used in expressions.*
  - *print can take one argument.*

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*

PHP Operators

■ Operators are used to operate on values. There are four classifications of operators:

- *Arithmetic*
- *Assignment*
- *Comparison*
- *Logical*

PHP Operators

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

PHP Operators

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

PHP Operators

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
<>	is not equal	5<>8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

PHP Operators

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

PHP Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions.
- You can use conditional statements in your code to do this.
- In PHP we have the following conditional statements…

PHP Conditional Statements

- **if** statement - use this statement to execute some code only if a specified condition is true
- **if...else** statement - use this statement to execute some code if a condition is true and another code if the condition is false
- **if...elseif....else** statement - use this statement to select one of several blocks of code to be executed
- **switch** statement - use this statement to select one of many blocks of code to be executed

Cont..

- The if statement executes some code if one condition is true.
  - if (*condition*) {
    code to be executed if condition is true;
    }
- The if....else statement executes some code if a condition is true and another code if that condition is false.
  - if (*condition*) {
    code to be executed if condition is true;
    } else {
    code to be executed if condition is false;
    }

Cont..

- The if….elseif…else statement executes different codes for more than two conditions.
  - if (*condition*) {
    code to be executed if this condition is true;
    } elseif (*condition*) {
    code to be executed if this condition is true;
    } else {
    code to be executed if all conditions are false;
    }

Cont..

- Use the switch statement to select one of many blocks of code to be executed.
  - switch (*n*) {
    case *label1:*
    code to be executed if n=label1;
    break;
    case *label2:*
    code to be executed if n=label2;
    break;
    case *label3:*
    code to be executed if n=label3;
    break;
    ...
    default:
    code to be executed if n is different from all labels;
    }

PHP Loops

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

PHP While and do while loops

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

PHP for and foreach Loops

- The for loop is used when you know in advance how many times the script should run.
  - for (*init counter; test counter; increment counter*) {
    code to be executed;
    }
    - *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
- The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.
  - foreach ($*array* as $*value*) {
    code to be executed;
    }

PHP functions

- PHP has more than 1000 built-in functions.
- Besides the built-in PHP functions, we can create our own functions.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.

  - function *functionName*() {
    code to be executed;
    }

Cont..

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

    <?php
        function writeMsg() {
        echo "Hello world!";
        }

        writeMsg();
    ?>

</body>
</html>
```

PHP Function Arguments

- Information can be passed to functions through arguments. An argument is just like a variable.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

```
<?php
    function firstName($fname) {
        echo "$fname.<br>";
    }
    firstName(“John");
    firstName(“Smith");
?>
```

Function with two arguments

```
<?php
    function myName($fname, $year)
    {
        echo "$fname. Born in $year <br>";
    }
    myName(“John", "1990");
    myName(“Smith", "1990");
?>
```

Default Argument Value

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

    setHeight(350);
    setHeight(); // will use the default value of 50
    setHeight(135);
    setHeight(80);
?>
```

Functions - Returning values

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

Arrays

- An array is a special variable, which can hold more than one value at a time.

```
<?php
    $myName = array(“fname", “mname", “lname");
    echo “My Name is" . $ fname[0] . ", " . $ mname[1] . " and " . $ lname[2] . ".";
?>
```

Create an Array in PHP

- In PHP, the array() function is used to create an array
  - array();
- In PHP, there are three types of arrays:
  - *Indexed arrays - Arrays with a numeric index*
  - *Associative arrays - Arrays with named keys*
  - *Multidimensional arrays - Arrays containing one or more arrays*

Array Type	Syntax	Example
Indexed arrays	$cars = array("Volvo", "BMW", "Toyota");	$cars = array("Volvo", "BMW", "Toyota"); the index can be assigned manually: Eg: $cars[0] = "Volvo"; $cars[1] = "BMW"; $cars[2] = "Toyota";
Associative arrays	$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");	$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43"); or: $age['Peter'] = "35"; $age['Ben'] = "37"; $age['Joe'] = "43";
Multidimensional arrays	$cars = array (array("Volvo",22,18), array("BMW",15,13), array("Saab",5,2), array("Land Rover",17,15));	echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].; echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].;

Sorting Arrays

- The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.
  - *sort() - sort arrays in ascending order*
  - *rsort() - sort arrays in descending order*
  - *asort() - sort associative arrays in ascending order, according to the value*
  - *ksort() - sort associative arrays in ascending order, according to the key*
  - *arsort() - sort associative arrays in descending order, according to the value*
  - *krsort() - sort associative arrays in descending order, according to the key*

Array object methods

- array_push($arr, $val)- pushes value to a array
- array_pop($arr)- removes an element from the end of an array and returns its value
- array_reverse($arr)- reverses the order of the elements in an array.
- array_flip($arr) - interchanges the keys and the values of an Associative array
- array_unique($arr) - remove all duplicate entries in the array.
- array_keys($arr)- recover the keys from an associative array.
- array_values($arr)- receives a PHP array.
- sort($arr) - Sorts scalar array in ascending order.
- rsort($arr) - Sorts scalar array in reverse order.
- asort($arr) - Sorts associative array by values.
- arsort($arr) - Sorts associative array by values in reverse order.
- ksort($arr) - Sorts associative array by ‘Keys’.
- krsort($arr) - Sorts associative array by ‘Keys’ in reverse order

PHP Basics - String functions

- **strlen()** – to return the length of the string
- **trim()** - removes any blank characters from the beginning and end of the string
- **strtolower()** - String to Lower converts any uppercase letters to lowercase
- **strtoupper()** - String to Upper converts any lowercase letters to uppercase
- **htmlspecialchars()** - takes a string and converts &, <, >, and double quotes to proper HTML entities

  - *<?php*
    $str = "This is some bold text   space before";
    echo htmlspecialchars($str);
    ?>

Class and Objects

- An object is a data type which stores data and information on how to process that data.
- In PHP, an object must be explicitly declared.
- First we must declare a class of object. For this, we use the class keyword.

```php
<?php
class Fruit {
  // code goes here...
}
?>
```

Class and Objects

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

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

Cont..

```php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
```

```php
$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
```

Cont..

- Here are some important definitions related to objects:
  - ***Classes*** *define how objects behave. Classes do not contain data.*
  - ***Objects*** *are instances of classes, which contain data.*
  - ***Members*** *are variables that belong to an object.*
  - ***Methods*** *are functions that belong to an object, and have access to its members.*
  - ***Constructor*** *is a special method that is executed when an object is created.*

Inheritance

- The most important feature of object oriented programming is inheritance. This feature allows us to reuse code we've written and extend it. For example, let's say we want to be able to define a math student, which also knows how to sum two numbers.

```
class Parent {
  // The parent’s class code
}

class Child extends Parent {
  // The  child can use the parent's class code
}
```

PHP time() Function

- The time() function returns the current time in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).
- Syntax
  - time();
- Eg:
  - <?php
    $t=time();
    echo($t . "
");
    echo(date("Y-m-d",$t));
    ?>

PHP Date() Function

- The PHP date() function formats a timestamp to a more readable date and time.
- Syntax
  - date(*format*,*timestamp*)
  - Eg:
  - <?php
    echo "Today is " . date("Y/m/d") . "
";
    ?>
- some characters that are commonly used for dates:
  - *d - Represents the day of the month (01 to 31)*
  - *m - Represents a month (01 to 12)*
  - *Y - Represents a year (in four digits)*
  - *l (lowercase 'L') - Represents the day of the week*
  - *F = Full name for the month.*
  - *j = The day of the month.*

Cont..

- Here are some characters that are commonly used for times:
  - *H - 24-hour format of an hour (00 to 23)*
  - *h - 12-hour format of an hour with leading zeros (01 to 12)*
  - *i - Minutes with leading zeros (00 to 59)*
  - *s - Seconds with leading zeros (00 to 59)*
  - *a - Lowercase Ante meridiem and Post meridiem (am or pm)*
- Eg
  - <?php
    echo "The time is " . date("h:i:sa");
    ?>

Visit: https://www.w3schools.com/php/php_ref_date.asp

Date With PHP mktime()

- The optional timestamp parameter in the date() function specifies a timestamp. If you do not specify a timestamp, the current date and time will be used.
- The mktime() function returns the Unix timestamp for a date.
- Syntax
  - mktime(*hour,minute,second,month,day,year*)
- Eg
  - <?php
    $d=mktime(11, 14, 54, 8, 12, 2014);
    echo "Created date is " . date("Y-m-d h:i:sa", $d);
    ?>

PHP strtotime()

- The PHP strtotime() function is used to convert a human readable string to a Unix time.
- Syntax
  - strtotime(*time,now*)
- Eg
  - <?php
    $d=strtotime("10:30pm April 15 2014");
    echo "Created date is " . date("Y-m-d h:i:sa", $d);
    ?>

```
<!DOCTYPE html>
<html>
<body>
<?php
$startdate=strtotime("Saturday");
$enddate=strtotime("+6 weeks", $startdate);
    while ($startdate < $enddate)
    {
      echo date("M d", $startdate) . "<br>";
      $startdate = strtotime("+1 week", $startdate);
    }
?>
</body>
</html>
```

Output:

Mar 09
Mar 16
Mar 23
Mar 30
Apr 06
Apr 13

PHP | Superglobals

- Superglobals are built-in variables that are always available in all scopes.
- you can access them from any function, class or file without having to do anything special.
- The PHP superglobal variables are:
  - *$GLOBALS*
  - *$_SERVER*
  - *$_REQUEST*
  - *$_POST*
  - *$_GET*
  - *$_FILES*
  - *$_ENV*
  - *$_COOKIE*
  - *$_SESSION*

$GLOBALS

- $GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script
- PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.

```
<?php
$x = 75;
$y = 25;
    function addition() {
        $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
    }
addition();
echo $z;
?>
```

$_SERVER

- $_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

```
<?php
    echo $_SERVER['PHP_SELF'];
    echo $_SERVER['SERVER_NAME'];
    echo $_SERVER['HTTP_HOST'];
    echo $_SERVER['HTTP_REFERER'];
    echo $_SERVER['HTTP_USER_AGENT'];
    echo $_SERVER['SCRIPT_NAME'];
?>
```

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request

$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script

$_REQUEST

PHP $_REQUEST is used to collect data after submitting an HTML form.

```
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>
```

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_REQUEST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>
```

$_POST

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

```
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>
```

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>
```

PHP $_GET

- PHP $_GET can also be used to collect form data after submitting an HTML form with method="get".
- $_GET can also collect data sent in the URL.

```
<html>
<body>
    <a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET
    </a>
</body>
</html>
```

```
<html>
<body>
    <?php
    echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
    ?>
</body>
</html>
```

GET vs. POST

- Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.
- Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- $_GET is an array of variables passed to the current script via the URL parameters.
- $_POST is an array of variables passed to the current script via the HTTP POST method.

GET vs. POST

- Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.
- Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- $_GET is an array of variables passed to the current script via the URL parameters.
- $_POST is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

- Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
- GET may be used for sending non-sensitive data.
- **Note:** GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

- Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.
- Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

Cont..

- **$_COOKIE:** The $_COOKIE Superglobal represents data available to a PHP script via HTTP cookies.
- **$_SESSION:** The $_SESSION Superglobal represents data available to a PHP script that has previously been stored in a session.
- **$_ENV:** The $_ENV Superglobal represents data available to a PHP script from the environment in which PHP is running.
- **$_FILES:** The $_FILES Superglobal represents data available to a PHP script from HTTP POST file uploads. Using $_FILES is the currently preferred way to handle uploaded files in PHP.

$_COOKIE

- A cookie is often used to identify a user.
- A cookie is a small file that the server embeds on the user's computer.
- Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.
- Syntax
  - setcookie(*name, value, expire, path, domain, secure, httponly*);
- Only the *name* parameter is required. All other parameters are optional.

$_SESSION

- When you work with an application, you open it, do some changes, and then you close it. This is much like a Session.
- The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.
- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.
- So; Session variables hold information about one single user, and are available to all pages in one application.
- A **session** is a temporary and interactive information interchange between two or more communicating devices.

PHP STATE MANAGEMENT

Introduction

- State management is used to help web applications to maintain their state in several HTTP requests when needed. PHP provide two different techniques to manage the state of your web application.
  - *Server Side State Management*
    - Session
  - *Client Side State Management*
    - Query String
    - Cookies

PHP COOKIES

Cookies: Introduction

- What is a Cookie?
  - *A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer.*
- Create Cookies With PHP
  - *A cookie is created with the setcookie() function*
- Syntax
  - *setcookie(name, value, expire, path, domain, secure, httponly);*

Cookies: Introduction

- **Name** – This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value** – This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** – This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** – This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** – This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Secure** – This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

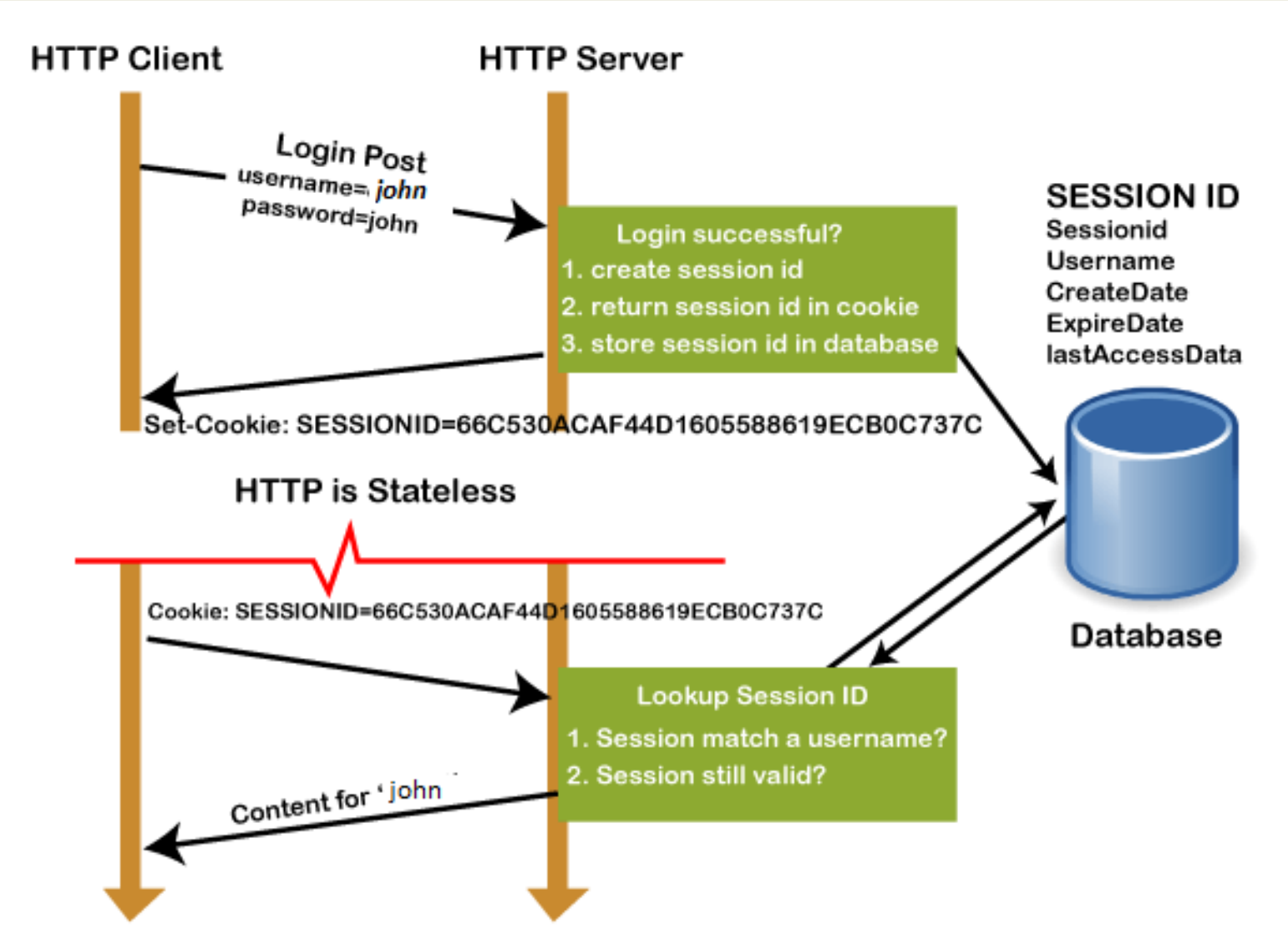
Cookies: Example

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

Within Body

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

PHP SESSION


HTTP Client
HTTP Server
Login Post
username=john
password=john
Login successful?
1. create session id
2. return session id in cookie
3. store session id in database
SESSION ID
Sessionid
Username
CreateDate
ExpireDate
lastAccessData
Set-Cookie: SESSIONID=66C530ACAF44D1605588619ECB0C737C
HTTP is Stateless
Cookie: SESSIONID=66C530ACAF44D1605588619ECB0C737C
Database
Lookup Session ID
1. Session match a username?
2. Session still valid?
Content for 'john'

Session: Introduction

- A PHP session is used to store data on a server rather than the computer of the user.
- Session identifiers or SID is a unique number which is used to identify every user in a session based environment.
- Session identifiers is used to link the user with his information on the server like posts, emails etc.

Session: Starting & Storing data

- The first step is to start up a session. After a session is started, session variables can be created to store information.
- The PHP session_start() function is used to begin a new session. It also creates a new session ID for the user.
- Storing Session Data: Session data in key-value pairs using the $_SESSION[] superglobal array.The stored data can be accessed during lifetime of a session.

```
<?php

session_start();

$_SESSION["Rollnumber"] = "11";
$_SESSION["Name"] = "Ajay";

?>
```

Accessing and Destroy Session

- Data stored in sessions can be easily accessed by firstly calling **session_start()** and then by passing the corresponding key to the **$_SESSION** associative array.
- The **session_destroy()** funct ion is used to completely destroy a session. The session_destroy() function does not require any argument.

```
<?php

session_start();

echo 'Name:' .
$_SESSION["Name;
echo 'Roll number:' .
$_SESSION["Rollnumber"];

session_destroy();

?>
```

session_regenerate_id

- Update the current session id with a newly generated one
- Description
  - *bool session_regenerate_id ([bool $delete_old_session = false])*
  - *session_regenerate_id() will replace the current session id with a new one, and keep the current session information.*

Example

```
<?php
session_start();
$old_sessionid = session_id();
session_regenerate_id();
$new_sessionid = session_id();
echo "Old Session: $old_sessionid<br />";
echo "New Session: $new_sessionid<br />";
?>
```

PHP FILE UPLOAD

$_FILES

- $_FILES is a two dimensional associative superglobal of items which are being uploaded by via HTTP POST method.
- Attributes of $_FILES
  - *[name] - Name of file which is uploading*
  - *[size] - Size of the file*
  - *[type] - Type of the file (like .pdf, .zip, .jpeg.....etc)*
  - *[tmp_name] - A temporary address where the file is located before processing the upload request*
  - *[error] -Types of error occurred when the file is uploading*
- How to Use
  - *$_FILES[input-field-name][name]*
  - *$_FILES[input-field-name][tmp_name]*
  - *$_FILES[input-field-name][size]*
  - *$_FILES[input-field-name][type]*
  - *$_FILES[input-field-name][error]*

Create The HTML Form

```
<form action="upload.php" me
thod="post" enctype="multipar
t/form-data">


    Select image to upload:
    <input type="file" name="file
ToUpload" id="fileToUpload">


    <input type="submit" value=
"Upload
Image" name="submit">
</form>
```

Mandatory Attributes

- form uses *method= "post"*
- *enctype="multipart/form-data",* specifies how the form-data should be encoded when submitting it to the server.
- *type="file"* attribute of the <input> tag shows the input field as a file-select control.

Upload File PHP Script

```
<?php   $target_dir = "uploads/";
$target_file = $target_dir .
basename($_FILES["fileToUpload"]["nam
e"]);
$imageFileType= strtolower(pathinfo($tar
get_file,PATHINFO_EXTENSION));
if(isset($_POST["submit"])) {
$check = getimagesize

($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
echo “Image File" . $check["mime"] . ".";
    }

else {
        echo "File is not an image.";
        $uploadOk = 0; }
}
?>
```

- $target_dir = "uploads/": specifies the directory where the file is going to be placed
- $target_file: specifies the path of the file to be uploaded
- $imageFileType: holds the file extension of the file.

Check File Existence, Size and Format

```
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}


// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}


// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
```

PHP VALIDATION

Validation: Introduction

- Validation means check the input submitted by the user. There are two types of validation. 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.*

Validation: Functions

- **empty():** Function used to check whether a variable is empty or not.
- **preg_match():** Function searches a string for pattern, returning true if the pattern exists, and false otherwise.
- **isset ():** Function is used to check whether a variable is set or not.

Example: To Check the empty fields

```
<?php
	$var = "";
			if( empty($var ) )
			{
				echo "The variable is empty";
			}
			else
			 {
				echo "The variable is having some value";
			}

?>
```

Preg_match() Function

- This function matches the value given by the user and defined in the regular expression.
- If the regular expression and the value given by the user, becomes equal, the function will return true, false otherwise.

Syntax:

preg_match($Pattern , $Subject , $regs)

- Pattern – Pattern is used to search the string.
- Subject – input given by the user.
- Regs

  *If matches are found for parenthesized substrings of **pattern** and the function is called with the third argument **regs**, the matches will be stored in the elements of the array **regs**.*

Example:

- preg_match(" /^Manipal/ " , " Manipal Academy of Higher Education")

Example

```
<?php

$pattern = /^Manipal/";

$subject = “Manipal Academy of Higher Education";

        if (preg_match( $pattern , $subject ) )

        {

                echo "Pattern Matched";

        }

          else

        {

                echo "Pattern Mismatched";

        }
?>
```

PHP WITH MYSQL

Basic PHP MySQL functions

- Connecting to a Database
- Selecting a database
- Running a query
- Using results of a query
- Closing the connection

Database Connection

- **mysqli_connect(server, username, password)**
  - ***server*** *default is the string* ***"localhost"***
  - ***username*** *is a string for the user name (“ABCD")*
  - ***password*** *is a string for the password (“abcd")*

$conn= mysqli_connect("localhost", “ABCD", “abcd");

For WAMP/MAMP/XAMPP with default password

$conn= mysqli_connect("localhost", "root", "");

Sample Connection Code

```
<?php
$conn = mysqli_connect("localhost", “root", "")
    or die("Could not connect: " . mysqli_error($conn));
print "Successful Connection";
mysqli_close($conn);
?>
```

Connection: MySQLi Object-Oriented

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

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

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

Connection: MySQLi Procedural

```php
<?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";
?>
```

Connection: PDO (PHP Data Objects)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
try {
    $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
    }
catch(PDOException $e)
    {
    echo "Connection failed: " . $e->getMessage();
    }
?>
```

Selecting a database

- **mysqli_select_db(connection , name)**
- select a database given by the string name
- The connection variable is required

e.g.

```
mysqli_select_db($conn, "labuser");
```

Connect & Select database code

```
<?php
$conn = mysqli_connect('localhost', ‘root', '')
or die ('No connection : ' . mysqli_error($conn));
mysqli_select_db($conn , ‘user') or die ('db will not open' .
    mysqli_error($conn));
print "Database  Connected";
mysqli_close($conn);
?>
```

Error Messages

- **mysqli_error(connection)**
- Returns an error string or error number (connection is optional - with last opened connection used if none supplied)
- Empty string is returned if there is no error.

Example

mysqli_error($conn);

Making A Query

- **mysqli_query(connection , query)**
- makes a select query
- query is a string for the MySQL query (in SQL)
- semicolon (;)should NOT be used to terminate query
- query uses valid SQL command

e.g.

```
$query = "SELECT * FROM student";
$result = mysqli_query($conn, $query);
```

Connect, Select db & query

```
<?php
$conn = mysqli_connect('localhost', ‘root', '')
or die ('No connection');
mysqli_select_db($conn ,‘user') or die ('db will not open');
$query = "SELECT surname FROM student";
$result = mysqli_query($conn, $query) or die("Invalid query");
print "Successful Query";
mysqli_close($conn);
?>
```

Closing a connection

- **mysqli_close(connection)**
- closes the database connection with the link

e.g.

mysqli_close($conn);

- **mysqli_free_result(result)**
- frees up memory during a program

mysqli_free_result($result);

Other mysql Functions (1)

- mysqli_num_rows(result)
- returns number of rows from a select query
- mysqli_fetch_row(result)
- each call returns the next row as an indexed array

e.g.

```
$result = mysqli_query($conn, "select * from module");
$num = mysqli_num_rows($result);
for($i=1; $i<=$num; $i++)
  {
$row = mysqli_fetch_row($result);
echo $row[0]  ." "  .  $row[1];
  }
```

Other mysql Functions (2)

- **mysqli_affected_rows(result)**
- used after an INSERT, UPDATE, or DELETE query to return the number of rows affected
- **mysqli_free_result(result)**

  frees memory within a php program
- **die**()
  Use to print message and exit from the current **php** script.

Other mysql Functions (3)

- **mysqli_num_fields(result)**
- returns the name of the table column whose position is given by index (0,1,...)

e.g.

```
$res = mysqli_query($conn, 'select  *  from module');

$numfields = mysqli_num_fields($res);

echo "number of fields in module is " . $numfields;
```

Other mysql Functions (4)

- **mysqli_fetch_array(result)**
- returns row information as an array

e.g.

```
$result = mysqli_query($conn, "select * from module");
while($row = mysqli_fetch_array($result))
 {
echo $row['modulecode']  ." "  . $row['modulename'];
 }
```

- Function returns TRUE while a row is fetched & FALSE when there are no rows available
- Function will fetch an array as associative or sequential

PHP MySQL Prepared Statements

- A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.
- Steps to execute Prepared statements:
  - *Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO user VALUES(?, ?, ?)*
  - *The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it*
  - *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*

Example: Prepared Statement

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
$conn = new mysqli($servername,
$username, $password, $dbname);

if ($conn->connect_error) {
    die("Connection failed: " . $conn-
>connect_error);
}

$stmt = $conn->prepare("INSERT INTO
user (firstname, lastname, email) VALUES
(?, ?, ?)");


$stmt->bind_param("sss", $firstname,
$lastname, $email);
```

```
// set parameters and execute
$firstname = “ABC";
$lastname = “DEF";
$email = “abc@example.com";
$stmt->execute();

$firstname = “UVW";
$lastname = “XYZ";
$email = “uvw@example.com";
$stmt->execute();

echo "created successfully";

$stmt->close();
$conn->close();
?>
```

Sample Code 1

```
<?php
$conn = mysqli_connect('localhost', 'root', '') or die ('No connection');
mysqli_select_db($conn, 'projemp') or die ('DB will not open');
$query = "SELECT eno, ename, salary from emp order by ename asc";
$result = mysqli_query($conn, $query) or die("Invalid query");
$num = mysqli_num_rows($result);
echo "<table border='1'><tr><th>eno</th><th>ename</th>
            <th>salary</th></tr>";
for($i=0;  $i<$num;  $i++)   {
      $row = mysqli_fetch_row($result);
      echo "<tr><td>" . $row[0] . "</td><td>" . $row[1];
      echo "</td><td>" . $row[2] . "</td></tr>";
 }
echo "</table>";
mysqli_close($conn);
?>
```

For Loop

```
//counts number of rows in $result
$num = mysqli_num_rows($result);

for($i=0;  $i<$num;  $i++)   {      // goes round  once for each row
      $row = mysqli_fetch_row($result);
                                                  //  puts next row into $row array
      echo "<tr><td>" . $row[0] . "</td><td>" . $row[1];
      echo "</td><td>" . $row[2] . "</td></tr>";
}    // prints the first, second & third column of the current row
    // as sequential array values ...  0,1,2
```

Sample Code 2

```
<?php
$conn = mysqli_connect('localhost', 'root', '') or die ('No connection');
mysqli_select_db($conn,'projemp') or die (' DB will not open');
$query = "SELECT eno, ename, salary from emp order by ename desc";
$result = mysqli_query($conn, $query) or die('Invalid query');
echo  "<table border='1'><tr><th>eno</th><th>ename</th>
            <th>salary</th></tr>";
while($row = mysqli_fetch_array($result))
 {
 echo "<tr><td>" . $row['eno'] . "</td><td>" . $row['ename'];
 echo "</td><td>" . $row['salary'] . "</td></tr>";
 }
echo "</table>";
mysqli_close($conn);
?>
```

While Loop

```
// while there is a new row to fetch
while($row = mysqli_fetch_array($result))
  {
  echo "<tr><td>" . $row['eno'] . "</td><td>" . $row['ename'];
  echo "</td><td>" . $row['salary'] . "</td></tr>";
  }   // print the columns using associative array index values
        //   'eno', 'ename'  & 'salary'
```

Next Step

- Having connected to mysql (mysqli_connect)
- Selected a database (mysqli_select_db)
- Run a query (mysqli_query)
- Process the result (mysqli_fetch_row, mysqli_fetch_array)

Adding variable to a query string

```
$value=$_POST["valuelist"];
$query = "SELECT * from emp where salary > " . $value;
```

[Whatever value is entered in the html front end is posted and plugged into the query string]

Where the where clause uses a text attribute inverted commas must be placed round the value

e.g. SELECT * FROM emp WHERE ename = ‘ABC’

```
$query = "SELECT * from emp where ename = '" . $value . "'";
```

[Note the single quotes placed round the variable]

Text Box

In an html form:

```
Enter  a salary value: <input  type = "text"  name = "textvalue">
```

Backend php file dbconnect1.php:

```
<?php
$sal = $_POST["textvalue"];
.....
$query = "SELECT * from emp where salary > " . $sal;
?>
```

dbconnect1.php

```
<?php
$sal=$_POST["textvalue"];   // receives posted value entered in textbox  called textvalue


$conn = mysqli_connect('localhost', 'root', '', 'labuser') or die ('No connection');

$query = "SELECT eno, ename, salary from emp where salary > " . $sal;
$result = mysqli_query($conn, $query) or die("Invalid query");

echo "<table border='1'><tr><th>eno</th><th>ename</th><th>salary</th></tr>";

while ($row = mysqli_fetch_row($result))  {
     echo "<tr><td>" . $row[0] . "</td><td>" . $row[1];
     echo "</td><td>" . $row[2] . "</td></tr>";
 }
echo  "</table>";

mysqli_close($conn);
?>
```

Select Box

In an html form:

```
<select name="valuelist">
<option value="24000">24000</option>
<option value="28000">28000</option>
<option value="32000">32000</option>
</select>
```

- Chosen valuelist option value posted to backend php file (dbconnect2.php)

```
$sal = $_POST["valuelist"];

$query = "SELECT * from emp where salary > " . $sal;
```

dbconnect2.php

```
<?php
$sal = $_POST["valuelist"];   // receives posted value option from selectbox valuelist


$conn = mysqli_connect('localhost', 'labuser', 'mcadb2020') or die ('No connection');

mysqli_select_db($conn, 'labuser') or die (' test will not open');

$query = "SELECT eno, ename, salary from emp where salary > " . $sal;
$result = mysqli_query($conn, $query) or die("Invalid query");

echo "<table border='1'><tr><th>eno</th><th>ename</th><th>salary</th></tr>";

while ($row = mysqli_fetch_row($result))  {
     echo "<tr><td>" . $row[0] . "</td><td>" . $row[1];
     echo "</td><td>" . $row[2] . "</td></tr>";
 }
echo  "</table>";

mysqli_close($conn);
?>
```

CRUD

- Create – Insert Query
- Read (Retrieve) – Select Query
- Update – Update Query
- Delete – Delete Query

CREATE

- **INSERT** is used to create a new record in the database
- Example
  - *INSERT into user VALUES (1,'ABC', 'IJK');*

READ

- SELECT is used to retrieve a record in the database
- Example
  - *SELECT * FROM user;*
  - *SELECT username, password FROM user WHERE ID = 1*

UPDATE

- UPDATE is used to change record(s) in the database
- Example
  - *UPDATE users SET username = ‘ABC’ WHERE ID = 1*
  - *UPDATE users SET username = ‘ABC’ WHERE username = ‘DEF’*

DELETE

- DELETE is used to remove records from the database
- Example
  - *DELETE FROM users WHERE ID = 1*

PHP include and require Statements

- It is possible to insert the content of one PHP file into another PHP file, with the include or require statement.
- Syntax
  - *include 'filename';*
  - *require 'filename';*

Example

```
<html>
<body>

    <h1>Welcome to my home page!</h1>
    <p>Some text.</p>
    <p>Some more text.</p>
    <?php include 'footer.php';?>

</body>
</html>
```

Cont..

- Difference between include and require
- When a file is included with the include statement and PHP cannot find it, the script will continue to execute
- When we using the require statement, the statement will not be executed because the script execution dies after the require statement

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

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

</body>
</html>
```

Example of CRUD: Code for CREATE

```
<?php
        if(isset($_POST['Submit'])) {
                $name = $_POST['name'];
                $email = $_POST['email'];
                $mobile = $_POST['mobile'];
                include_once("config.php");


$result = mysqli_query($mysqli, "INSERT INTO
users(name, email, mobile)
VALUES('$name','$email','$mobile')");


echo "User added successfully.
<a href='index.php'>View Users</a>";
        }
        ?>
```

*Note: The **isset()** function is an inbuilt function in **PHP** which checks whether a variable is set and is not NULL*

Example of CRUD: Code for READ

```
<?php
include_once("config.php");
$result = mysqli_query($mysqli, "SELECT * FROM users ORDER BY id DESC");
?>
```

Add New User

Name	Mobile	Email	Update
ABCDEF	1234567890	abc@example.com	Edit | Delete

Example of CRUD: Code for UPDATE

```php
<?php

include_once("config.php");

if(isset($_POST['update']))

{

        $id = $_POST['id'];

        $name=$_POST['name'];

        $mobile=$_POST['mobile'];

        $email=$_POST['email'];

        $result = mysqli_query($mysqli, "UPDATE users SET name='$name',email='$email',mobile='$mobile' WHERE id=$id");

        header("Location: index.php");

}

?>
```

```php
<?php

$id = $_GET['id'];

$result = mysqli_query($mysqli, "SELECT * FROM users WHERE id=$id");

while($user_data = mysqli_fetch_array($result))

{

        $name = $user_data['name'];

        $email = $user_data['email'];

        $mobile = $user_data['mobile'];

}

?>
```

Example of CRUD: Code for DELETE

```
<?php
include_once("config.php");
$id = $_GET['id'];
$result = mysqli_query($mysqli, "DELETE FROM users WHERE id=$id");
header("Location:index.php");
?>
```

Cont..

- A session is started with the **session_start()** function.
- Session variables are set with the PHP global variable: $_SESSION.
- To change a session variable, just overwrite it
- To remove all global session variables and destroy the session, use **session_unset()** and **session_destroy()**

PHP Include Files

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

PHP include vs. require

- The require statement is also used to include a file into the PHP code.
- However, there is one big difference between include and require;
  - *when a file is included with the include statement and PHP cannot find it, the script will continue to execute:*
  - *In require statement, code will not be executed because the script execution dies after the require statement returned an error:*