CS3215: Web Technology TY Div C n D AY 2022-23

Study Material for Section-I-Part-III - PHP n MySQL

(**Resource:** *www.w3schools.com*)

Learning PHP Script

The primary difference between a scripting language and a programming language is in their execution – programming languages use a compiler to convert the high-level programming languages into machine language, on the other hand, scripting languages use an interpreter.

Is PHP an alternative to JavaScript?

PHP is server-side scripting language whereas JavaScript is a client-side scripting language. PHP doesn't execute within browser whereas JavaScript executes within browser. PHP supports database whereas JavaScript doesn't support databases. PHP accepts both upper case and lower case variables while JavaScript doesn't.

PHP has been one of the oldest and most used scripting languages for server-side development, and it's powering over more than **42 million sites today** on the World Wide Web, according to SimilarTech. PHP has certainly grown so much and matured over the course of 25 years, and the latest major release, PHP7 brought along many enhancements to the performance, boosting speed to up to 100% of that of PHP 5 and lowering the memory usage.

Alternatives for PHP

C# - ASP.Net web framework. ASP.Net is powering over 17 million sites today and Xfinity

JavaScript - Node.js - is currently the hottest technology in web development; it's a run-time environment for JavaScript on the server side, simply put, it allows JavaScript to be used for server-side development. The node community is growing very fast, and the node package manager (NPM) boasts the highest amount of packages with 650,000+ packages available for you to build your applications.

JAVA - frameworks use Java for server-side development like Spring and JEE.

Python - Django and Flask frameworks

Ruby – Rails (MVC), Sinatra, Cuba, Hanami and a few others.

Go- Go is a rich compiled programming language built by Google, and is said to be even simpler than Python. It's a concurrent programming language and since its compiled as well, this contributes to making it a speedy language for web development.

Erlang - is one of the most powerful options on this list. It's a concurrent functional programming language that was designed specifically to handle the massive amount of traffic in real-time applications. It's certainly ideal for building REST API on your backend.

Elixir - possesses a rich framework called Phoenix

C++ - The most famous frameworks are WT and Crow, which is C++'s version of Flask, a mini web framework.

Rust - is a popular language and a very beloved one, we ought to mention that you can develop web applications.

Before you continue you should have a basic understanding of the following:

- HTML
- CSS
- JavaScript

What is PHP?

PHP is a open source server side scripting language, and a powerful tool for making dynamic and interactive Web pages.

PHP is an acronym for "PHP: Hypertext Preprocessor"

PHP 7 is the latest stable release.

PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

PHP scripts are executed on the server

PHP is free to download and use

What is a PHP File?

PHP files can contain text, HTML, CSS, JavaScript, and PHP code

PHP code is executed on the server, and the result is returned to the browser as plain HTML

PHP files have extension ".php"

What Can PHP Do?

PHP can generate dynamic 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, and modify data in your database

PHP can be used to control user-access

PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)

PHP is compatible with almost all servers used today (Apache, IIS, etc.)

PHP supports a wide range of databases

PHP is free. Download it from the official PHP resource: www.php.net

PHP is easy to learn and runs efficiently on the server side

Basic 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

?>
```

The default file extension for PHP files is "`.php`".

A PHP file normally contains HTML tags, and some PHP scripting code.

PHP Case Sensitivity

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

Note: However; all variable names are case-sensitive!

`$color`, `$COLOR`, and `$coLOR` are treated as three different variables.

PHP Comments

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

# This is also a single-line comment
?>
```

```
<?php
/*
This is a multiple-lines comment block
that spans over multiple lines
*/
?>
```

Displaying Output – `echo/echo()` and `print/print()` statements are often used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters while print can take one argument. echo is marginally faster than print

<?php

echo "<h2>PHP is Fun!</h2>";

echo "Hello world!
";

echo "I'm about to learn PHP!
";

echo "This ", "string ", "was ", "made ", "with multiple parameters."; ?>

But

<?php

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

print "Hello world!
";

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

print "This ", "string ", "was ", "made ", "with multiple parameters.";

?> gives error

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

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

Creating (Declaring) PHP Variables

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

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

PHP Variables Scope

PHP has three different variable scopes:

- local
- global
- static

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

```
<?php
$x = 5; // global scope

function myTest() {

echo "<p>Variable x inside function is: $x</p>"; // will not dispaly
}


myTest();

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

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

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

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

PHP The global Keyword

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

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

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

myTest();
echo $y; // outputs 15
?>
```

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.

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

function myTest() {
  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
```

```php
echo $y; // outputs 15
?>
```

PHP The static Keyword

Normally, when a function is executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

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

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

My First PHP Script Program

<html>

<body>

<?php

echo "Hello World - My first PHP script!";

?>

</body>

</html>

Program for Addition of two numbers

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

PHP Data Types

String

Integer

Float (floating point numbers - also called double)

Boolean

Array

Object

NULL

Resource

String

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

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

Integer

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

Float

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

Boolean

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

Array

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

Class

```
<?php
class Car {
  public $color;
  public $model;
  public function __construct($color, $model) {
    $this->color = $color;
    $this->model = $model;
  }
  public function message() {
    return "My car is a " . $this->color . " " . $this->model . "!";
  }
}

$myCar = new Car("black", "Volvo");
echo $myCar -> message();
echo "<br>";
$myCar = new Car("red", "Toyota");
echo $myCar -> message();
?>
```

PHP NULL Value

If a variable is created without a value, it is automatically assigned a value of NULL.

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

PHP Strings

strrev() - Reverse a String

```
<?php
echo strrev("Hello world!"); // outputs !dlrow olleH
?>
```

str_word_count() - Count Words in a String

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

strpos() - Search For a Text Within a String

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

str_replace() - Replace Text Within a String

Replace the text "world" with "Dolly":

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello
Dolly!
?>
```

Richness of PHP

Function	Description
addcslashes()	Returns a string with backslashes in front of the specified characters

Note: Be careful using addcslashes() on 0 (NULL), r (carriage return), n (newline), f (form feed), t (tab) and v (vertical tab). In PHP, \0, \r, \n, \t, \f and \v are predefined escape sequences.

addslashes()	Returns a string with backslashes in front of predefined characters

The addslashes() function returns a string with backslashes in front of predefined characters.

single quote ('), double quote ("), backslash (\), NULL

This function can be used to prepare a string for storage in a database and database queries.

bin2hex()	Converts a string of ASCII characters to hexadecimal values

convert "Hello World!" to hexadecimal values:

```
<?php
$str = bin2hex("Hello World!");     //48656c6c6f20576f726c6421
```

```
echo($str);
?>
```

chop()	Removes whitespace or other characters from the right end of a string
chr()	Returns a character from a specified ASCII value
chunk_split()	Splits a string into a series of smaller parts
convert_cyr_string()	Converts a string from one Cyrillic character-set to another
convert_uudecode()	Decodes a uuencoded string [unix to unix coding / mime]
convert_uuencode()	Encodes a string using the uuencode algorithm
count_chars()	Returns information about characters used in a string
crc32()	Calculates a 32-bit CRC for a string

Print the result of crc32():

```
<?php
$str = crc32("Hello World!");     //472456355
printf("%u\n",$str);
?>
```

crypt()	One-way string hashing
The crypt() function returns a hashed string using DES, Blowfish, or MD5 algorithms.	
echo()	Outputs one or more strings
explode()	Breaks a string into an array
fprintf()	Writes a formatted string to a specified output stream
get_html_translation_table()	Returns the translation table used by htmlspecialchars() and htmlentities()
hebrev()	Converts Hebrew text to visual text
hebrevc()	Converts Hebrew text to visual text and new lines (\n) into
hex2bin()	Converts a string of hexadecimal values to ASCII

	characters
html_entity_decode()	Converts HTML entities to characters
htmlentities()	Converts characters to HTML entities
htmlspecialchars_decode()	Converts some predefined HTML entities to characters
htmlspecialchars()	Converts some predefined characters to HTML entities
implode()	Returns a string from the elements of an array
join()	Alias of implode()
lcfirst()	Converts the first character of a string to lowercase
levenshtein()	Returns the Levenshtein distance between two strings

Calculate the Levenshtein distance between two strings:

```php
<?php
echo levenshtein("Hello World","ello World");
echo "<br>";
echo levenshtein("Hello World","ello World",10,20,30);
?>
```

localeconv()	Returns locale numeric and monetary formatting information
For localization	
ltrim()	Removes whitespace or other characters from the left side of a string
md5()	Calculates the MD5 hash of a string

Calculate the MD5 hash of the string "Hello":

```
<?php
$str = "Hello";
echo md5($str);                    //8b1a9953c4611296a827abf8c47804d7
?>
```

md5_file()	Calculates the MD5 hash of a file

Calculate the MD5 hash of the text file "test.txt":

```
<?php
$filename = "test.txt";
$md5file = md5_file($filename);
echo $md5file;
?>
```

metaphone()	Calculates the metaphone key of a string
money_format()	Returns a string formatted as a currency string
nl_langinfo()	Returns specific local information
nl2br()	Inserts HTML line breaks in front of each newline in a string
number_format()	Formats a number with grouped thousands
ord()	Returns the ASCII value of the first character of a string
parse_str()	Parses a query string into variables
print()	Outputs one or more strings
printf()	Outputs a formatted string
quoted_printable_decode()	Converts a quoted-printable string to an 8-bit string

quoted_printable_encode()	Converts an 8-bit string to a quoted printable string
quotemeta()	Quotes meta characters
rtrim()	Removes whitespace or other characters from the right side of a string
setlocale()	Sets locale information
sha1()	Calculates the SHA-1 hash of a string

Calculate the SHA-1 hash of the string "Hello":

```
<?php
$str = "Hello";
echo sha1($str);    //f7ff9e8b7bb2e09b70935a5d785e0cc5d9d0abf0
?>
```

sha1_file()	Calculates the SHA-1 hash of a file
similar_text()	Calculates the similarity between two strings
soundex()	Calculates the soundex key of a string
sprintf()	Writes a formatted string to a variable

sscanf()	Parses input from a string according to a format
str_getcsv()	Parses a CSV string into an array
str_ireplace()	Replaces some characters in a string (case-insensitive)
str_pad()	Pads a string to a new length
str_repeat()	Repeats a string a specified number of times
str_replace()	Replaces some characters in a string (case-sensitive)
str_rot13()	Performs the ROT13 encoding on a string
str_shuffle()	Randomly shuffles all characters in a string
str_split()	Splits a string into an array
str_word_count()	Count the number of words in a string
strcasecmp()	Compares two strings (case-insensitive)

strchr()	Finds the first occurrence of a string inside another string (alias of strstr())
strcmp()	Compares two strings (case-sensitive)
strcoll()	Compares two strings (locale based string comparison)
strcspn()	Returns the number of characters found in a string before any part of some specified characters are found
strip_tags()	Strips HTML and PHP tags from a string
stripcslashes()	Unquotes a string quoted with addcslashes()
stripslashes()	Unquotes a string quoted with addslashes()
stripos()	Returns the position of the first occurrence of a string inside another string (case-insensitive)
stristr()	Finds the first occurrence of a string inside another string (case-insensitive)
strlen()	Returns the length of a string

strnatcasecmp()	Compares two strings using a "natural order" algorithm (case-insensitive)
strnatcmp()	Compares two strings using a "natural order" algorithm (case-sensitive)
strncasecmp()	String comparison of the first n characters (case-insensitive)
strncmp()	String comparison of the first n characters (case-sensitive)
strpbrk()	Searches a string for any of a set of characters
strpos()	Returns the position of the first occurrence of a string inside another string (case-sensitive)
strrchr()	Finds the last occurrence of a string inside another string
strrev()	Reverses a string
strripos()	Finds the position of the last occurrence of a string inside another string (case-insensitive)

strrpos()	Finds the position of the last occurrence of a string inside another string (case-sensitive)
strspn()	Returns the number of characters found in a string that contains only characters from a specified charlist
strstr()	Finds the first occurrence of a string inside another string (case-sensitive)
strtok()	Splits a string into smaller strings
strtolower()	Converts a string to lowercase letters
strtoupper()	Converts a string to uppercase letters
strtr()	Translates certain characters in a string
substr()	Returns a part of a string
substr_compare()	Compares two strings from a specified start position (binary safe and optionally case-sensitive)
substr_count()	Counts the number of times a substring occurs in a

	string
substr_replace()	Replaces a part of a string with another string
trim()	Removes whitespace or other characters from both sides of a string
ucfirst()	Converts the first character of a string to uppercase
ucwords()	Converts the first character of each word in a string to uppercase
vfprintf()	Writes a formatted string to a specified output stream
vprintf()	Outputs a formatted string
vsprintf()	Writes a formatted string to a variable
wordwrap()	Wraps a string to a given number of characters

PHP Numbers

- is_int()
- is_integer()
- is_long()
- is_float()
- is_double()
- is_finite()
- is_infinite()
- is_nan()
- is_numeric()

Type Casting:

(int), (integer), or intval()

PHP Math Functions

Function	Description
abs()	Returns the absolute (positive) value of a number
acos()	Returns the arc cosine of a number
acosh()	Returns the inverse hyperbolic cosine of a number
asin()	Returns the arc sine of a number

asinh()	Returns the inverse hyperbolic sine of a number
atan()	Returns the arc tangent of a number in radians
atan2()	Returns the arc tangent of two variables x and y
atanh()	Returns the inverse hyperbolic tangent of a number
base_convert()	Converts a number from one number base to another
bindec()	Converts a binary number to a decimal number
ceil()	Rounds a number up to the nearest integer
cos()	Returns the cosine of a number
cosh()	Returns the hyperbolic cosine of a number
decbin()	Converts a decimal number to a binary number

dechex()	Converts a decimal number to a hexadecimal number
decoct()	Converts a decimal number to an octal number
deg2rad()	Converts a degree value to a radian value
exp()	Calculates the exponent of e
expm1()	Returns exp(x) - 1
floor()	Rounds a number down to the nearest integer
fmod()	Returns the remainder of x/y
getrandmax()	Returns the largest possible value returned by rand()
hexdec()	Converts a hexadecimal number to a decimal number
hypot()	Calculates the hypotenuse of a right-angle triangle

intdiv()	Performs integer division
is_finite()	Checks whether a value is finite or not
is_infinite()	Checks whether a value is infinite or not
is_nan()	Checks whether a value is 'not-a-number'
lcg_value()	Returns a pseudo random number in a range between 0 and 1
log()	Returns the natural logarithm of a number
log10()	Returns the base-10 logarithm of a number
log1p()	Returns log(1+number)
max()	Returns the highest value in an array, or the highest value of several specified values
min()	Returns the lowest value in an array, or the lowest value of several specified values

mt_getrandmax()	Returns the largest possible value returned by mt_rand()
mt_rand()	Generates a random integer using Mersenne Twister algorithm
mt_srand()	Seeds the Mersenne Twister random number generator
octdec()	Converts an octal number to a decimal number
pi()	Returns the value of PI
pow()	Returns x raised to the power of y
rad2deg()	Converts a radian value to a degree value
rand()	Generates a random integer
round()	Rounds a floating-point number
sin()	Returns the sine of a number

sinh()	Returns the hyperbolic sine of a number
sqrt()	Returns the square root of a number
srand()	Seeds the random number generator
tan()	Returns the tangent of a number
tanh()	Returns the hyperbolic tangent of a number

PHP Constants

To create a constant, use the `define()` function.

PHP Operators

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

===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7.
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

PHP Array Operators

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

?:	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
??	**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*. Introduced in PHP 7

PHP Conditional Statements

In PHP we have the following conditional statements:

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

PHP Loops

In PHP, we have the following loop types:

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

PHP User Defined Functions

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

PHP Arrays

Indexed Array

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

Associative Array

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

Multidimensional Array

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

PHP - Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

- `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

---

PHP Global Variables - Superglobals

Some predefined variables in PHP 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.

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 (also from within functions or methods).

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;
?>
```

PHP $_SERVER

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

The example below shows how to use some of the elements in $_SERVER:

Example

```
<?php
echo $_SERVER['PHP_SELF'];           o/p- /demo/demo_global_server.php
echo "<br>";
echo $_SERVER['SERVER_NAME'];     -  35.194.26.41
echo "<br>";
echo $_SERVER['HTTP_HOST'];         -35.194.26.41
echo "<br>";
echo $_SERVER['HTTP_REFERER'];   -https://tryphp.w3schools.com/showphp.php?
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT']; -  AppleWebKit/537.36 (KHTML, like Gecko)
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];   -/demo/demo_global_server.php
?>
```

Element/Code	Description	Output
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script	/serverglobal.php
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server	CGI/1.1

	is using	
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server	::1
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)	localhost
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)	Apache/2.4.34 (Win32) OpenSSL/1.0.2o PHP/5.6.38
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)	HTTP/1.1
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)	GET
$_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)	Notice: Undefined index: HTTP_ACCEPT_CHARSET in C:\xampp\htdocs\serverglobal.php on line 29
$_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	Notice: Undefined index: HTTPS in C:\xampp\htdocs\serverglobal.php on line 32
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page	::1
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page	**Notice**: Undefined index: REMOTE_HOST

$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server	49169
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script	C:/xampp/htdocs/serverglobal.php
$_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)	postmaster@localhost
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)	80
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages	*Apache/2.4.34 (Win32) OpenSSL/1.0.2o PHP/5.6.38 Server at localhost Port 80*
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script	

$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

What is HTTP?

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server.

Example: A client (browser) sends an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

HTTP Methods

- **GET**
- **POST**
- **PUT**
- **HEAD**
- **DELETE**
- **PATCH**
- **OPTIONS**

The two most common HTTP methods are: GET and POST.

The GET Method

GET is used to request data from a specified resource.

GET is one of the most common HTTP methods.

Note that the query string (name/value pairs) is sent in the URL of a GET request:

```
/test/demo_form.php?name1=value1&name2=value2
```

Some other notes on GET requests:

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests are only used to request data (not modify)

The POST Method

POST is used to send data to a server to create/update a resource.

The data sent to the server with POST is stored in the request body of the HTTP request:

```
POST /test/demo_form.php HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

POST is one of the most common HTTP methods.

Some other notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

Comparison Chart

	GET	POST
Parameters are placed inside	URI	Body
Purpose	Retrieval of documents	Updation of data

	GET	POST
Query results	Capable of being bookmarked.	Cannot be bookmarked.
Security	Vulnerable, as present in plaintext	Safer than GET method
Form data type constraints	Only ASCII characters are permitted.	No constraints, even binary data is permitted.
Form data length	Should be kept as minimum as possible.	Could lie in any range.
Visibility	Can be seen by anyone.	Doesn't display variables in URL.
Variable size	Up to 2000 character.	Up to 8 Mb
Caching	Method data can be cached.	Does not cache the data.

PHP $_REQUEST

PHP $_REQUEST is a PHP super Global variable which 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;
  }
}
?>
```

PHP $_POST

$_POST is an array of variables passed to the current script via the HTTP POST method.

PHP $_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.

```
<html>

<body>

<form method="post" action="collect.php">

  Name: <input type="text" name="fname">

  Middle Name: <input type="text" name="mname">

  Last Name: <input type="text" name="lname">

  <input type="submit">

</form>

</body>
</html>
```

collect.php

```
<?php

if ($_SERVER["REQUEST_METHOD"] == "POST") {

  // collect value of input field

  $name = $_POST['fname'];

  $mname = $_POST['mname'];

  $lname = $_POST['lname'];

  if (empty($name)) {echo "Name is empty"; } else { echo $name;   }

  if (empty($mname)) {echo "Name is empty"; } else { echo $mname;  }

  if (empty($lname)) {echo "Name is empty"; } else { echo $lname;  }

}

?>

</body>

</html>
```

PHP $_GET

PHP $_GET is a PHP super global variable which is used to collect form data after submitting an HTML form with method="get".

$_GET is an array of variables passed to the current script via the URL parameters.

PHP Form Handling

PHP - A Simple HTML Form

```
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

```
<?php
if( $_POST["name"] || $_POST["email"] ) {
if (preg_match("/[^A-Za-z'-]/",$_POST['name'] )) {
die ("invalid name and name should be alpha");
}
echo "Welcome ". $_POST['name']."<br />";
echo "You mail id is". $_POST['email']. "<br />";
exit();
}
?>
```

PHP Form Handling – GET Method

```
<html>
<body>
<form action="welcome1.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```

```
<?php
if( $_GET["name"] || $_GET["email"] ) {
if (preg_match("/[^A-Za-z'-]/",$_GET['name'] )) {
die ("invalid name and name should be alpha");
}
echo "Welcome ". $_GET['name']."<br />";
echo "You mail id is". $_GET['email']. "<br />";
exit();
}
?>
```

GET vs. POST

Both GET and POST create an array (e.g. array(key1 => value1, 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.

Developers prefer POST for sending form data.

For Security

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)
3. Use htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

PHP Regular Expressions

A regular expression is a sequence of characters that forms a search pattern.

```
$exp = "/w3schools/i";
```

Function	Description

preg_match()	Returns 1 if the pattern was found in the string and 0 if not
preg_match_all()	Returns the number of times the pattern was found in the string, which may also be 0
preg_replace()	Returns a new string where matched patterns have been replaced with another string

```php
<?php
$str = "The rain in SPAIN falls mainly on the plains.";
$pattern = "/ain/i";
echo preg_match_all($pattern, $str); // Outputs 4
?>
```

Regular Expression Patterns

Brackets are used to find a range of characters:

Expression	**Description**
[abc]	Find one character from the options between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find one character from the range 0 to 9

Metacharacter	**Description**

|	Find a match for any one of the patterns separated by | as in: cat|dog|fish
.	Find just one instance of any character
^	Finds a match as the beginning of a string as in: ^Hello
$	Finds a match at the end of the string as in: World$
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifier	**Description**
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*

n?	Matches any string that contains zero or one occurrences of *n*
n{x}	Matches any string that contains a sequence of *X n*'s
n{x,y}	Matches any string that contains a sequence of X to Y *n*'s
n{x,}	Matches any string that contains a sequence of at least X *n*'s

Get a date --- Date() Function

```
<?php
Echo(“1”);
echo date("Y/m/d");
echo date("Y.m.d");
echo date("Y-m-d");
?>
```

Output - Sunday 2020/12/06 2020.12.06 2020-12-06

Automatic Copyright Year

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

© 2010-<?php echo date("Y");?>
```

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

Get a Time ------ Date() Function

```
<?php
echo "The time is " . date("h:i:sa");
?>
```

PHP File Handling

Create File - **fopen()**

```
$myfile = fopen("testfile.txt", "w") or die("Unable to open file!");
```

Write to File - fwrite()

```
$txt = "Manikrao Dhre\n";
fwrite($myfile, $txt)
```

readfile() Function

```
echo readfile("webdictionary.txt");
```

or

```
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
```

End-Of-File - feof()

```
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
```

PHP File Upload

n your "php.ini" file, search for the file_uploads directive, and set it to On:

file_uploads = On

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

<form action="upload.php" method="post" enctype="multipart/form-data">

 <input type="file" name="fileToUpload" id="fileToUpload">
 <input type="submit" value="Upload Image" name="submit">
</form>

</body>
</html>
```

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
  $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
  if($check !== false) {
    echo "File is an image - " . $check["mime"] . ".";
    $uploadOk = 1;
  } else {
    echo "File is not an image.";
    $uploadOk = 0;
  }
}
?>
```

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. 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.

A cookie is created with the setcookie() function.

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

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

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

PHP Sessions

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

What is a PHP 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.

Start a PHP Session

A session is started with the **session_start()** function.

Session variables are set with the PHP global variable: $_SESSION.

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

MySQL Database

What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- 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
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

PHP + MySQL Database System

- PHP combined with MySQL are cross-platform - **you can develop in Windows and serve on a Unix platform**

Facts about MySQL Database

- MySQL is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like **Facebook, Twitter, and Wikipedia**).
- Another great thing about MySQL is that it can be scaled down to support **embedded database applications.**

PHP Connect to MySQL

PHP 5 and later can work with a MySQL database using:

- MySQL extension (deprecated in 2012)
- MySQLi extension (the "i" stands for improved)
- PDO (PHP Data Objects)

Should I Use MySQLi or PDO?

- If you need a short answer, it would be "**Whatever you like**".

- **PDO will work on 12 different database systems**, whereas MySQLi will only work with MySQL databases.
- So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.
- Both are **object-oriented, but MySQLi also offers a procedural API.**
- Both support Prepared Statements - Prepared Statements protect from SQL injection, and are very important for web application security.

Example (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";
?>
```

Example (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";
?>
```

Example (PDO)

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

Close the Connection

MySQLi Procedural: **`mysqli_close($conn);`**

MySQLi Object-Oriented: **`$conn->close();`**

PDO: **`$conn = null;`**

Create a MySQL Database Using MySQLi (OOP)

```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);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . $conn->error;
}

?>
```

Create a MySQL Database Using 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());
}

// 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);
?>
```

Create a MySQL Database Using PDO

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

try {
  $conn = new PDO("mysql:host=$servername", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "CREATE DATABASE myDBPDO";
  // use exec() because no results are returned
  $conn->exec($sql);
  echo "Database created successfully<br>";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

Create a MySQL Table Using MySQLi

```
CREATE TABLE Personalinfo (
grno INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
```

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// sql to create table
$sql = "CREATE TABLE Personalinfo (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
  echo "Table Personalinfo created successfully";
} else {
  echo "Error creating table: " . $conn->error;
}

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

Insert Data Into MySQL Using MySQLi

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

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

```
$sql = "INSERT INTO Personalinfo (grno,firstname, lastname, email)
VALUES (123456, 'Disha', 'Dhore', 'dishadhore@sinhgad.edu')";

if ($conn->query($sql) === TRUE) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}
```

Insert Multiple Records Into MySQL Using MySQLi

```
$sql = "INSERT INTO Personalinfo (grno,,firstname, lastname, email)
VALUES (123456'Disha', 'Dhore', 'dishadhore@sinhgad.edu');";
$sql .= "INSERT INTO Personalinfo (grno,firstname, lastname, email)
VALUES (234567'Ruchi', 'Dhore', 'ruchidhore@gmail.com');";
$sql .= "INSERT INTO Personalinfo (grno,,firstname, lastname, email)
VALUES (456789'Manik', 'Dhore', 'manikdhore@gmail.com')";

if ($conn->multi_query($sql) === TRUE) {
  echo "New records created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}
```

Prepared Statements and Bound Parameters

```
// prepare and bind
$stmt = $conn->prepare("INSERT INTO Personalinfo (grno, firstname, lastname, email)
VALUES (?, ?, ?)");
$stmt->bind_param("isss", $grno, $firstname, $lastname, $email);

// set parameters and execute

$grno = 123456
$firstname = "Disha";
$lastname = "Dhore";
$email = "dishadhore@sinhgad.edu";
$stmt->execute();
```

Select Data From a MySQL Database

```
SELECT column_name(s) FROM table_name
```

```
SELECT * FROM table_name
```

```
$sql = "SELECT grno, firstname, lastname, email FROM Personalinfo ";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "Grno: " . $row["grno"]. " - Name: " . $row["firstname"]. " " .
$row["lastname"]. row["email"]."<br>";
  }
} else {
  echo "0 results";
}
```

Select and Filter Data From a MySQL Database

```
SELECT column_name(s) FROM table_name WHERE column_name operator value
```

```
$sql = "SELECT id, firstname, lastname FROM Personalinfo WHERE lastname='Dhore'";
$result = $conn->query($sql);

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

Select and Order Data with MySQLi

The ORDER BY clause is used to sort the result-set in ascending or descending order.

The ORDER BY clause sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC
```

```
$sql = "SELECT grno, firstname, lastname FROM Personalinfo ORDER BY lastname";
$result = $conn->query($sql);
```

Delete Data from a MySQL Table Using MySQLi

```
DELETE FROM table_name
WHERE some_column = some_value
```

```
$sql = "DELETE FROM Personalinfo WHERE id=123456";
```

```
if ($conn->query($sql) === TRUE) {
  echo "Record deleted successfully";
} else {
  echo "Error deleting record: " . $conn->error;
}
```

Update Data In a MySQL Table Using MySQLi

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

```
$sql = "UPDATE Personalinfo SET lastname='More' WHERE grno=123456";

if ($conn->query($sql) === TRUE) {
  echo "Record updated successfully";
} else {
  echo "Error updating record: " . $conn->error;
}
```

Limit Data Selections From a MySQL Database

MySQL provides a LIMIT clause that is used to specify the number of records to return.

```
$sql = "SELECT * FROM Orders LIMIT 30";

$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";

$sql = "SELECT * FROM Orders LIMIT 15, 10";
```