Unit -2: Server Side Scripting with Database Connectivity

Introduction to sever side Scripting Language

A server-side scripting language is a programming language that runs on a web server, rather than on a user's computer. It is used to generate dynamic content on websites, such as web pages that are customized to a user's input, or to interact with databases and other server-side resources.

A server-side script is a program run on the web-server that generates content for a web-client or otherwise responds to some web-client action. The simplest server-side script is one that generates HTML content which then gets sent to the browser on the other end

For server-side scripting, a web server serves as the platform for running it. They create the pages that one would send to the browser.

Server-side or back-end developers create, design, and manage server-side code responsible for data exchange

Server-side Scripting:

- It helps work with the back end.
- It doesn’t depend on the client.
- It runs on the web server.
- It helps provide a response to every request that comes in from the user/client.
- This is not visible to the client side of the application.
- It requires the interaction with the server for the data to be processed.
- It is considered to be a secure way of working with applications.
- It can be used to customize web pages.
- It can also be used to provide dynamic websites.
- Languages: PHP, ASP.Net, Python, Ruby, etc.

Features of Server Side Scripting

Server-side scripting has several important features that make it a powerful tool for web development:

- ☑ **Access to server-side resources:** Server-side scripting allows developers to access server-side resources, such as databases, file systems, and other services. This makes it possible to create dynamic web applications that can store and retrieve data, perform calculations, and interact with other web services.
- ☑ **Customized content generation:** Server-side scripting allows developers to generate customized content based on user input, such as search queries, form submissions, or user preferences. This can be used to create personalized web pages, recommendations, and other customized content.
- ☑ **Security:** Server-side scripting allows developers to implement security measures, such as user authentication, input validation, and encryption, to protect web applications from attacks and unauthorized access.
- ☑ **Scalability**: Server-side scripting makes it possible to scale web applications to handle large volumes of traffic by offloading processing tasks to the server. This can improve performance and reduce the load on the client-side browser.
- ☑ **Maintenance:** Server-side scripting allows developers to maintain web applications more easily by keeping the code and data on the server, rather than on the client-side browser. This can simplify the development process and make it easier to maintain and update web applications over time.

Overall, server-side scripting is a powerful tool that enables developers to create dynamic, secure, and scalable web applications that can deliver customized content to users.

<u>Comparison with client side scripting</u>

- ☑ Client-side scripting refers to the code that runs on the user's web browser, after the web page has been downloaded from the server. This code is written in languages such as JavaScript, and is used to add interactivity to the web page, such as animations, form validation, and dynamic content. Client-side scripting is executed on the user's computer, so it can provide immediate feedback to the user, without requiring a round trip to the server.

- ☑ Server-side scripting, on the other hand, refers to the code that runs on the web server, before the web page is sent to the user's browser. This code is typically written in languages such as PHP, Python, Ruby, or Java, and is used to generate dynamic content, such as web pages that are customized to the user's input, or to interact with databases and other server-side resources. Server-side scripting is executed on the web server, so it can be used to perform complex tasks that require access to server-side resources.
- ☑ In summary, client-side scripting is used to add interactivity to web pages, while server-side scripting is used to generate dynamic content and interact with server-side resources. Both types of scripting are important in web development and are used together to create rich, interactive web applications.

Introduction to PHP

— PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.

- PHP was originally an abbreviation of Personal Home Page

✍ PHP was created by Rasmus Lerdorf in 1994. It’s currently maintained by the PHP Development Team

What can PHP do?

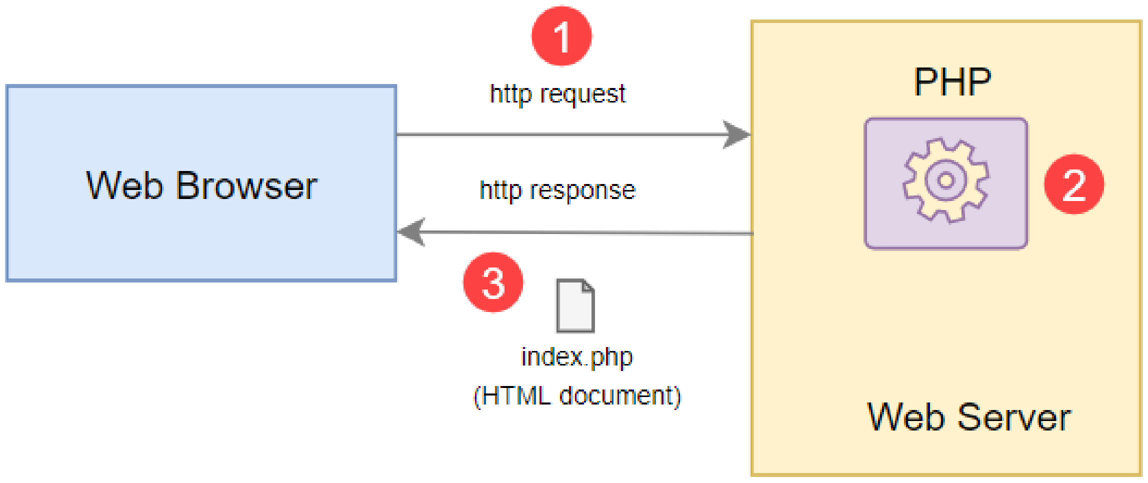
⇒ PHP has two main applications:

- ☞ **Server-side scripting** – PHP is well-suited for developing dynamic websites and web applications.
- ☞ **Command-line scripting** – like Python and Perl, you can run PHP script from the command line to perform administrative tasks like sending emails and generating PDF files.

— PHP is a popular **server-side scripting language** used for web development. Here are some important points about PHP:

- **Open source:** PHP is an open-source language, which means it is free to use and distribute. It is widely used in web development, and has a large community of developers who contribute to its development and maintenance.
- **Cross-platform:** PHP is a cross-platform language, which means it can run on a variety of operating systems, including Windows, Linux, and macOS. This makes it easy to deploy and maintain PHP applications on different platforms.
- **Easy to learn:** PHP is a relatively easy language to learn, especially for developers who are already familiar with programming concepts. It has a simple syntax and can be used to create dynamic, interactive web applications quickly.
- **Integration with databases:** PHP has built-in support for popular databases like MySQL, Oracle, and PostgreSQL. This makes it easy to interact with databases and retrieve data from them.
- **Large community:** PHP has a large community of developers and users, which means it has a wealth of resources, including documentation, tutorials, and libraries. This makes it easy to find solutions to common problems and to get help when needed.
- **Security:** PHP has several security features built-in, such as input validation and encryption. It also has a variety of security libraries and tools that can be used to secure web applications.
- **Scalability:** PHP is highly scalable, which means it can be used to develop web applications that can handle large volumes of traffic. It can be used in conjunction with other technologies like caching, load balancing, and content delivery networks (CDNs) to improve performance and scalability.

How PHP (as server side scripting language) works?



- First, the web browser sends an HTTP request to the web server, e.g., index.php.
- Second, the PHP preprocessor that locates on the web server processes PHP code to generate the HTML document.
- Third, the web server sends the HTML document back to the web browser.

PHP Basic Syntax

PHP files are saved with the "**.php**" extension.PHP scripts can be written anywhere in the document within PHP tags along with normal HTML.

Canonical PHP tags

⇒ When PHP parses a file, it looks for opening and closing tags, which are **<?php** and **?>** which tell PHP to start and stop interpreting the code between them. Parsing in this manner allows PHP to be embedded in all sorts of different documents, as everything outside of a pair of opening and closing tags is ignored by the PHP parser.

⇒ PHP includes a short echo tag **<?=** which is a short-hand to the more verbose **<?php echo**.

```
<?php echo 'if you want to serve PHP code in XHTML or XML
documents, use these tags'; ?>
```

```
You can use the short echo tag to <?= 'print this string' ?>.
It's equivalent to <?php echo 'print this string' ?>.

<? echo 'this code is within short tags, but will only work '.
'if short_open_tag is enabled'; ?>
```

NOTE: Short tags are available by default but can be disabled either via the short_open_tag php.ini configuration file directive, or are disabled by default if PHP is built with the --disable-short-tags configuration.

⇒ **If a file contains only PHP code**, it is **preferable to omit the PHP closing tag at the end** of the file. This prevents accidental whitespace or new lines being added after the PHP closing tag, which may cause unwanted effects because PHP will start output buffering when there is no intention from the programmer to send any output at that point in the script.

```
<?php
echo "Hello world";

// ... more code

echo "Last statement";

// the script ends here with no PHP closing tag
```

PHP statements end with a semicolon (;).

This is used to separate different statements within a PHP code block.

Commenting PHP Code

Single Line Comment: As the name suggests these are single line or short relevant explanations that one can add to their code. To add this, we need to begin the line with (//) or (#).

```
<?php

// This is a single line comment
// These cannot be extended to more lines

echo "hello world!!!";

# This is also a single line comment

?>
```

Multi-line Comment: These are used to accommodate multiple lines with a single tag and can be extended to many lines as required by the user. To add this, we need to begin and end the line with (/*…*/)

```
<?php
/* This is a multi line comment
    In PHP variables are written
    by adding a $ sign at the beginning.*/

$greet = "hello world!";
echo $greet;
?>
```

Whitespace & line breaks

In most cases, whitespace and line breaks don't have special meaning in PHP. Therefore, you can place a statement in one line or span it across multiple lines.

```
 <?php
// PHP code illustrate the whitespace insensitivity
$var1         =     15;
$var2 =
30;
$sum = $var1
+
$var2;

// "\n" for new line
echo $sum, "\n";

$sum1 = $var1 + $var2;
echo $sum1;
?>
```

Case Sensitivity

- ☞ PHP is a SEMI-CASE-SENSITIVE language which means some features are case-sensitive while others are not.
- ☞ Following are case-insensitive
  - ▪ All the keywords (if, else, while, for, foreach, etc.)
  - ▪ Functions
  - ▪ Statements
  - ▪ Classes
- ☞ PHP Variables are case-sensitive.
  - ▪ Only variables with different cases are treated differently.

```
<?php
// Here we can see that all echo
// statements are executed in the same manner

$variable = 25;
echo $variable;
ECHO $variable;
EcHo $variable;

// but this line will show RUNTIME ERROR as
// "Undefined Variable"
echo $VARIABLE
?>
```

NOTE: PHP strings are enclosed in double quotes (" ") or single quotes (' ')

Variables in PHP

- ☞ PHP variables begin with the $ symbol, followed by the variable name

  For example, $name = "John"; assigns the string "John" to the variable $name.

- ☞ The variable name must start with the dollar sign ($).
- ☞ The first character after the dollar sign ($) must be a letter (a-z) or the underscore (_).
- ☞ The remaining characters can be underscores, letters, or numbers.
- ☞ PHP variables are case-sensitive. It means that $message and $Message variables are entirely different.

PHP Data Types

- Variables can store data of different types, and different data types can do different things.
- Some of the data types are listed below:
    - **Integer:** Represents whole numbers without decimal points.
        - Example: $age = 25;
    - **Float:** Represents numbers with decimal points (floating-point numbers).
        - Example: $price = 19.99;
    - **String:** Represents sequences of characters enclosed in quotes.
        - Example: $name = "John Smith";
    - **Boolean:** Represents a value that is either true or false.
        - Example: $isLogged = true;
    - **Array:** Represents an ordered collection of elements.
        - Example: $numbers = [1, 2, 3, 4, 5];
    - **Object:** Represents an instance of a class.
        - Example: $user = new User();
    - **Null:** Represents the absence of a value.
        - Example: $variable = null;
    - **Resource:** Represents a reference to an external resource such as a database connection.

PHP Constants

- ☞ A constant is simply a name that holds a single value. As its name implies, the value of a constant cannot be changed during the execution of the PHP script. From PHP 7, a constant can hold an array.
- ☞ A constant can be accessed from anywhere in the script.
- ☞ Use the **define()** function or **const** keyword to define a constant.
- ☞ Use the **define()** function if you want to define a constant conditionally or using an expression.

```
define(name, value, case-insensitive)
```

name: Specifies the name of the constant; value: Specifies the value of the constant

case-insensitive: Specifies whether the constant name should be case-insensitive. Default is false

```
<?php

// Define a constant

define("PI", 3.14);


// Use the constant

$radius = 5;

$area = PI * $radius * $radius;


echo "The area of a circle with radius $radius is: $area";

?>
```

```
<?php
const MY_CONSTANT                                                    = 5;
echo MY_CONSTANT;
?>
```

Operators in PHP

- ☞ Operators are symbols that you can use to manipulate values and variables by performing an operation on them. You need to use some of these operators to work out the totals and tax on the customer's order.

Arithmetic Operators

Example	Name	Result
+$a	Identity	Conversion of *$a* to int or float as appropriate.
-$a	Negation	Opposite of *$a*.
$a + $b	Addition	Sum of *$a* and *$b*.
$a - $b	Subtraction	Difference of *$a* and *$b*.
$a * $b	Multiplication	Product of *$a* and *$b*.
$a / $b	Division	Quotient of *$a* and *$b*.
$a % $b	Modulo	Remainder of *$a* divided by *$b*.
$a ** $b	Exponentiation	Result of raising *$a* to the *$b*'th power.

- The division operator ("/") returns a float value unless the two operands are integers (or strings that get converted to integers) and the numbers are evenly divisible, in which case an integer value will be returned

Assignment Operator

- The basic assignment operator is "="
- The value of an assignment expression is the value assigned. That is, the value of "$a = 3" is 3. This allows you to do some tricky things:

```
<?php

$a = ($b = 4) + 5; // $a is equal to 9 now, and $b has been set to 4.

?>
```

In addition to the basic assignment operator, there are "combined operators" for all of the binary arithmetic, array union and string operators that allow you to use a value in an expression and then set its value to the result of that expression

```php
<?php
$a = 3;
$a += 5; // sets $a to 8, as if we had said: $a = $a + 5;
$b = "Hello ";
$b .= "There!"; // sets $b to "Hello There!", just like $b = $b .
"There!";
?>
```

Arithmetic Assignment Operators

Example	Equivalent	Operation
$a += $b	$a = $a + $b	Addition
$a -= $b	$a = $a - $b	Subtraction
$a *= $b	$a = $a * $b	Multiplication
$a /= $b	$a = $a / $b	Division
$a %= $b	$a = $a % $b	Modulus
$a **= $b	$a = $a ** $b	Exponentiation

Comparison Operators

Example	Name	Result
$a == $b	Equal	`true` if *$a* is equal to *$b* after type juggling.
$a === $b	Identical	`true` if *$a* is equal to *$b*, and they are of the same type.
$a != $b	Not equal	`true` if *$a* is not equal to *$b* after type juggling.
$a <> $b	Not equal	`true` if *$a* is not equal to *$b* after type juggling.
$a !== $b	Not identical	`true` if *$a* is not equal to *$b*, or they are not of the same type.
$a < $b	Less than	`true` if *$a* is strictly less than *$b*.
$a > $b	Greater than	`true` if *$a* is strictly greater than *$b*.
$a <= $b	Less than or equal to	`true` if *$a* is less than or equal to *$b*.
$a >= $b	Greater than or equal to	`true` if *$a* is greater than or equal to *$b*.

Example	Name	Result
$a <=> $b	Spaceship	An int less than, equal to, or greater than zero when *$a* is less than, equal to, or greater than *$b*, respectively.

Incrementing/Decrementing Operators

Example	Name	Effect
++$a	Pre-increment	Increments *$a* by one, then returns *$a*.
$a++	Post-increment	Returns *$a*, then increments *$a* by one.
--$a	Pre-decrement	Decrements *$a* by one, then returns *$a*.
$a--	Post-decrement	Returns *$a*, then decrements *$a* by one.

Logical Operators

Example	Name	Result
$a and $b	And	`true` if both *$a* and *$b* are `true`.
$a or $b	Or	`true` if either *$a* or *$b* is `true`.
$a xor $b	Xor	`true` if either *$a* or *$b* is `true`, but not both.
! $a	Not	`true` if *$a* is not `true`.
$a && $b	And	`true` if both *$a* and *$b* are `true`.
$a || $b	Or	`true` if either *$a* or *$b* is `true`.

String Operators

There are two string operators. The first is the concatenation operator ('.'), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side.

```
<?php
$a = "Hello ";
```

```
$b = $a . "World!"; // now $b contains "Hello World!"

$a = "Hello ";
$a .= "World!";     // now $a contains "Hello World!"
?>
```

Array Opertors

Example	Name	Result
$a + $b	Union	Union of *$a* and *$b*.
$a == $b	Equality	`true` if *$a* and *$b* have the same key/value pairs.
$a === $b	Identity	`true` if *$a* and *$b* have the same key/value pairs in the same order and of the same types.
$a != $b	Inequality	`true` if *$a* is not equal to *$b*.
$a <> $b	Inequality	`true` if *$a* is not equal to *$b*.
$a !== $b	Non-identity	`true` if *$a* is not identical to *$b*.

The + operator returns the right-hand array appended to the left-hand array; for keys that exist in both arrays, the elements from the left-hand array will be used, and the matching elements from the right-hand array will be ignored.

Bitwise operators

Bitwise operators allow evaluation and manipulation of specific bits within an integer.

Example	Name	Result
`$a & $b`	And	Bits that are set in both *$a* and *$b* are set.
`$a | $b`	Or (inclusive or)	Bits that are set in either *$a* or *$b* are set.
`$a ^ $b`	Xor (exclusive or)	Bits that are set in *$a* or *$b* but not both are set.
`~ $a`	Not	Bits that are set in *$a* are not set, and vice versa.
`$a << $b`	Shift left	Shift the bits of *$a* *$b* steps to the left (each step means "multiply by two")

Example	Name	Result
`$a >> $b`	Shift right	Shift the bits of *$a $b* steps to the right (each step means "divide by two")

Other Operators

- The comma operator (,) separates function arguments and other lists of items. It is normally used incidentally.
- Two special operators, **new** and **->**, are used to instantiate a class and access class members, respectively.
- The **ternary operator (?:)** takes the following form

  ```
  condition ? value_if_true : value_if_false
  ```

  ```
  ($grade >= 50 ? 'Passed' : 'Failed')
  ```

  This expression evaluates student grades to 'Passed' or 'Failed'.
- The **error suppression operator (@)** can be used in front of any expression—that is, anything that generates or has a value. For example,

  ```
      $a = @(57/0);
  ```

  If you are suppressing warnings in this way, you need to write some error handling code to check when a warning has occurred. If you have PHP set up with the track_errors feature enabled in php.ini, the error message will be stored in the global variable $php_errormsg

Control Statements

Control statements are conditional statements that execute a block of statements if the condition is correct. The statement inside the conditional block will not execute until the condition is satisfied.

Control Statements in PHP are divided into three types-

- Conditional/Selection statements
- Iteration/Loop statements
- Jump statements

Conditional statements

Conditional Statements are used to perform actions based on different conditions. Sometimes when we write a program, we want to perform some different actions for different actions. We can solve this by using conditional statements.

In PHP we have these conditional statements:

- **if Statement.**
- **if-else Statement**
- **If-elseif-else Statement**
- **Switch statement**

The if Statement

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

Syntax:

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

Example:

```
<?php

    $age = 20;

    if ($age >= 18) {

        echo "You are old enough to vote.";

    }

?>
```

The if-else Statement

This statement executes the block of code inside the **if** statement if the expression is evaluated as true and executes the block of code inside the else statement if the expression is evaluated as false.

Syntax:

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

```
<?php

    $age = 15;

    if ($age >= 18) {

        echo "You are old enough to vote.";

    } else {
        echo "You are not old enough to vote.";

    }

?>
```

The *if-elseif-else* statement

- The *if-elseif-else* statement is used to execute one block of code if the first condition is true, another block of code if the second condition is true, and so on. If none of the conditions are true, a final block of code is executed. The syntax of the *if-elseif-else* statement is

```
<?php

    $age = 20;

    if ($age < 13) {

        echo "You are a child.";

    } elseif ($age >= 13 && $age < 18) {

        echo "You are a teenager.";

    } else {

        echo "You are an adult.";

    }
```

```
?>
```

Switch Statement

- This statement allows us to execute different blocks of code based on different conditions. Contrary to the if statement, the switch statement is used when the condition contains a particular value rather than a range of values
- Use the switch statement to select one of many blocks of code to be executed.

Syntax:

```
switch (value) {
    case value1:
        // code to execute if value matches value1
        break;
    case value2:
        // code to execute if value matches value2
        break;
    // more case statements as needed
    default:
        // code to execute if none of the cases match value
}
```

Example:

```
<?php
$message = '';
$role = 'author';
```

```
switch ($role) {
       case 'admin':
              $message = 'Welcome, admin!';
              break;
       case 'editor':
       case 'author':
              $message = 'Welcome! Do you want to create a new article?';
              break;
       case 'subscriber':
              $message = 'Welcome! Check out some new articles.';
              break;
       default:
              $message = 'You are not authorized to access this page';
}
echo $message;
```

- PHP also supports the alternative syntax for the switch statement as follows:

```
<?php
switch (expression):
       case value1:
              // code block 1
              break;
       case value2:
```

```
                // code block 2

                break;

        default:

                // default code block

                break;

endswitch;
```

PHP Loops

- Often when you write code, you want the same block of code to run over and over again a certain number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.
- Loops are used to execute the same block of code again and again, as long as a certain condition is true.
- 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

The for loop

— The for loop is used to execute a block of code a specified number of times.

— The syntax for the for loop is:

```
for (initialization; condition; increment) {

    // code to execute
```

```
        }
```

The *initialization* is an expression that is executed once before the *loop* starts. The condition is an expression that is tested at the beginning of each iteration. If the condition is true, the code block is executed. The increment is an expression that is executed at the end of each iteration.

```
<?php
    for ($i = 0; $i < 10; $i++) {
        echo $i . " ";
    }
?>
```

The do-while loop

- The **do-while** loop is similar to the while loop, but the condition is tested at the end of each iteration. This means that the code block is executed at least once, even if the condition is initially false.
- The syntax for the do-while loop is:

```
do {
    // code to execute
} while (condition);
```

The code block is executed at least once, and then the condition is tested. If the condition is true, the code block is executed again. The loop continues until the condition is false.

```
<?php
    $i = 0;
    do {
        echo $i . " ";
        $i++;
    } while ($i < 10);
?>
```

The while loop

- The while loop is used to execute a block of code as long as a condition is true.
- The syntax for the while loop is:

```
while (condition) {
    // code to execute
}
```

```
Example
     <?php
         $i = 0;
         while ($i < 10) {
             echo $i . " ";
             $i++;
         }
```

```
?>
```

The foreach loop

- The **foreach loop** is used to loop through arrays and other **iterable objects**.
- The syntax for the foreach loop is:

```
foreach (array_expression as $value) {
    // code to execute
}
```

The array_expression is an expression that returns an array or other iterable object. The value is a variable that is set to the current value of the array element on each iteration.

Example

```
<?php
    $numbers = array(1, 2, 3, 4, 5);
    foreach ($numbers as $value) {
        echo $value . " ";
    }
?>
```

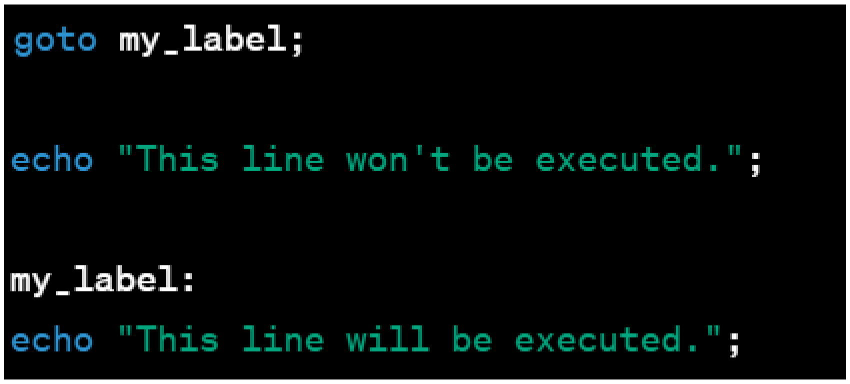
PHP Jump Statements

In PHP, jump statements are used to transfer control from one part of the code to another. The following are the jump statements that can be used in PHP:

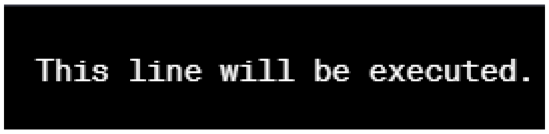
- ☑ **GOTO:** This statement transfers control to a specified label in the program. However, the use of GOTO is generally discouraged in PHP because it can make the code difficult to understand and maintain.

- ☑ **BREAK:** This statement is used to exit a loop or switch statement. When used inside a loop, it terminates the loop and resumes execution at the next statement after the loop. When used inside a switch statement, it terminates the switch and resumes execution at the next statement after the switch.
- ☑ **CONTINUE:** This statement is used to skip the rest of the current iteration of a loop and continue with the next iteration. When used inside a loop, it skips any remaining statements in the loop and resumes execution at the beginning of the loop for the next iteration.
- ☑ **RETURN:** This statement is used to exit a function and return a value to the caller. When used inside a function, it immediately terminates the function and returns the specified value to the code that called the function.

Example: *goto*

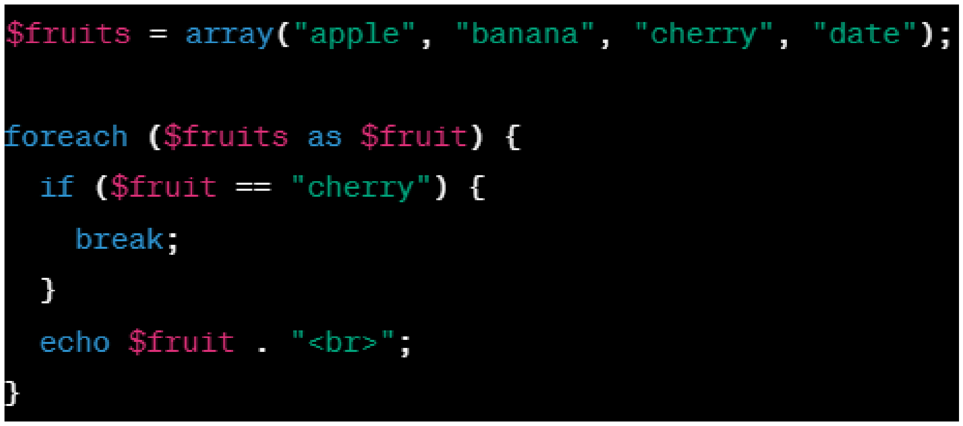


```
goto my_label;

echo "This line won't be executed.";

my_label:
echo "This line will be executed.";
```

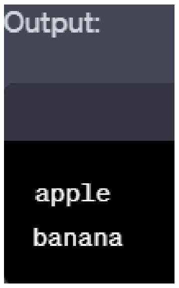
Output: 

```
This line will be executed.
```

Example: *break;*



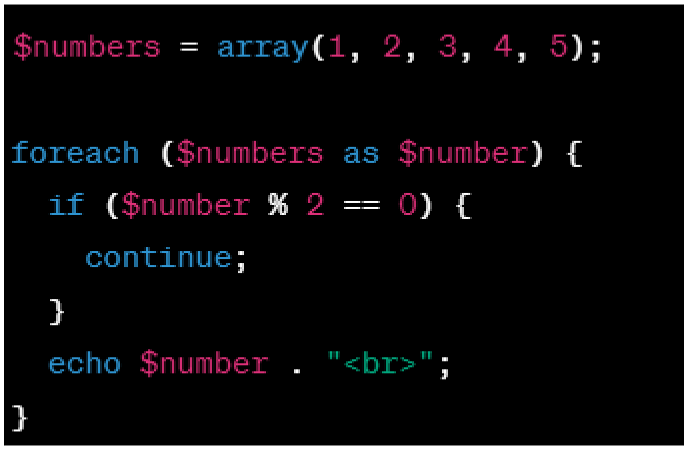
```
$fruits = array("apple", "banana", "cherry", "date");

foreach ($fruits as $fruit) {
  if ($fruit == "cherry") {
    break;
  }
  echo $fruit . "<br>";
}
```



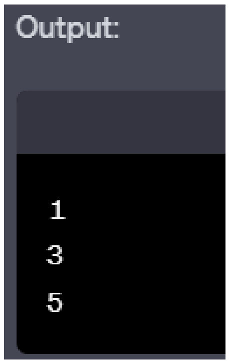
Output:

```
apple
banana
```

Example: *continue*



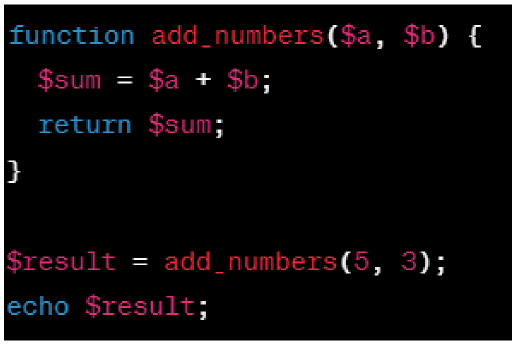
```
$numbers = array(1, 2, 3, 4, 5);

foreach ($numbers as $number) {
  if ($number % 2 == 0) {
    continue;
  }
  echo $number . "<br>";
}
```



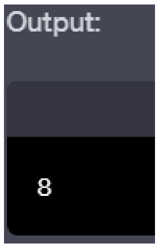
Output:

```
1
3
5
```

Example: *return*



```
function add_numbers($a, $b) {
  $sum = $a + $b;
  return $sum;
}

$result = add_numbers(5, 3);
echo $result;
```



```
Output:

 8
```

PHP Arrays

An array in PHP is actually an ordered map. A map is a type that associates values to keys. This type is optimized for several different uses; it can be treated as an array, list (vector), hash table (an implementation of a map), dictionary, collection, stack, queue, and probably more. As array values can be other arrays, trees and multidimensional arrays are also possible

- ☑ PHP provides you with two types of arrays: **indexed** (numerically indexed) and **associative** (arrays with named keys).
- ☑ The keys of the indexed array are integers that start at 0. Typically, you use indexed arrays when you want to access the elements by their positions.
- ☑ The keys of an associative array are of any types (integer, float, string, boolean, or an object) And you use associative arrays when you want to access elements by named keys.

Creating arrays

- ☑ In PHP, you can use the **array() construct** or **[ ] syntax** to define an array. The [] syntax is shorter and more convenient.

Creating an array using array() construct

- o To define an array, you use the **array()** construct.
- o The following example creates an empty array:

```
<?php
$empty_array = array();
```

- To create an array with some initial elem The following example uses the [] syntax to create a new array that consists of three numbers:ents, you place a comma-separated list of elements within parentheses of the array() construct.
- For example, the following defines an array that has three numbers

```
<?php

$scores = array(1, 2, 3);
```

Creating an array using the [] syntax

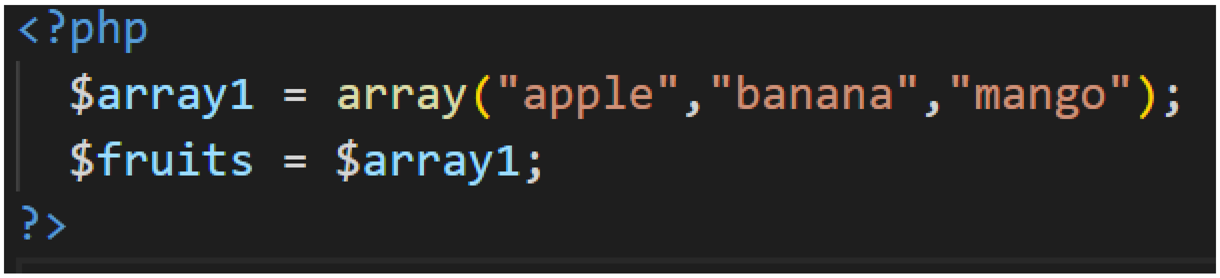
— PHP provides a more convenient way to define arrays with the shorter syntax [], known as JSON notation. The following example uses [] syntax to create a new empty array:

```
<?php
$empty_array = [];
```

— The following example uses the [] syntax to create a new array that consists of three numbers

```
<?php

$scores = [1, 2, 3];
```

- Depending on the contents you need in your array, you might not need to manually initialize them as in the preceding example. If you have the data you need in another array, you can simply copy one array to another using the = operator



```
<?php
  $array1 = array("apple","banana","mango");
  $fruits = $array1;
?>
```

- If you want an ascending sequence of numbers stored in an array, you can use the **range()** function to automatically create the array for you. The following statement creates an array called numbers with elements ranging from 1 to 10:

```
$numbers = range(1,10);
```

- The **range()** function has an optional third parameter that allows you to set the step size between values. For instance, if you want an array of the odd numbers between 1 and 10, you could create it as follows:

```
$odds = range(1, 10, 2);
```

- The **range()** function can also be used with characters, as in this example:

```
$letters = range('a', 'z');
```

Accessing array elements

- To access an element in an array, you specify the index of the element within the square brackets:

$array_name[index]

Example:

```
<?php

$scores = [1, 2, 3];

echo $scores[0];
```

Displaying arrays

- To show the contents of an array, you use the **var_dump()** function.

- The **var_dump()** function dumps information about one or more variables. The information holds type and value of the variable(s).

  var_dump(var1, var2, ...);

For example:

```
<?php

$scores = [1, 2, 3];
var_dump($scores);
```

output

```
array(3) {
  [0]=> int(1)
  [1]=> int(2)
  [2]=> int(3)
}
```

- Or you can use the **print_r()** function

  .

```
<?php

$scores = array(1, 2, 3);
print_r($scores);
```

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
```

Adding an element to the array

- To add an element to an array, you use the following syntax:

```
$array_name[] = new_element;
```

PHP will calculate the highest numerical index plus one each time you assign an element to the array.

The following example shows how to add the number 4 to the `$scores` array:

```
<?php

$scores = [1, 2, 3];

$scores[] = 4;
```

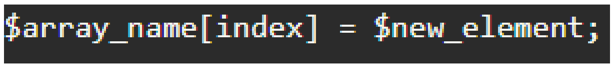
In this example, we defined an array that consists of three numbers initially. Then, we added the number 4 to the array.

It's possible to use an index when you add a new element to the array. For example:

```
$scores = [1, 2, 3];

$scores[3] = 4;
```

Changing array elements

- The following statement changes the element located at the index to the `$new_element`:



```
$array_name[index] = $new_element;
```

For example, to change the first element of the $scores array from 1 to zero, you do it as follows:

```
<?php

$scores = [1, 2, 3];

$scores[0] = 0;
```

Removing array elements

- To remove an element from an array, you use the ***unset()*** function. The following removes the second element of the `$scores` array:

```
<?php

$scores = [1, 2, 3];

unset($scores[1]);
```

Getting the size of an array

- To get the number of elements in an array, you use the count() function. For example:

```
<?php


$scores = [1, 2, 3, 4, 5];


echo count($scores);//1
```

- The **sizeof()** function is an alias of the count() function. We can also use sizeof() to get the size of an array.

Array Operators

- One set of special operators applies only to arrays.

Operator	Name	Example	Result
+	Union	$a + $b	Union of $a and $b. The array $b is appended to $a, but any key clashes are not added.
==	Equality	$a == $b	True if $a and $b contain the same elements.
===	Identity	$a === $b	True if $a and $b contain the same elements, with the same types, in the same order.
!=	Inequality	$a != $b	True if $a and $b do not contain the same elements.
<>	Inequality	$a <> $b	Same as !=.
!==	Non-identity	$a !== $b	True if $a and $b do not contain the same elements, with the same types, in the same order.

Associative Arrays

- An associative array is an array that uses key-value pairs to store data, unlike numeric arrays which use numeric indices
- Associative arrays are arrays that use named keys that you assign to them.
- Associative arrays are useful when you need to store data that is not easily indexed by numbers, such as names or other strings.
- The keys of an associative array are of any types (integer, float, string, boolean, or an object)
- Note that when you use non-string keys, PHP will automatically convert the key to a string
- In PHP, there are two main ways to define an associative array:
  - ❖ **Using the array() function**:
    - — You can define an associative array using the **array()** function and providing the key-value pairs as arguments. Here's an example:

```
$person = array(
    "name" => "John",
    "age" => 30,
    "city" => "Kathmandu"
);
```

In the example above, `$person` is an associative array with three key-value pairs. The keys are "name", "age", and "city", and the values are "John", 30, and "Kathmandu", respectively.

- ❖ **Using square brackets:**
  - — You can also define an associative array using square brackets and assigning key-value pairs to it. Here's an example:

```
$colors = [

    "red" => "#FF0000",

    "green" => "#00FF00",
    "blue" => "#0000FF"

];
```

In the example above, $colors is an associative array with three key-value pairs. The keys are "red", "green", and "blue", and the values are "#FF0000", "#00FF00", and "#0000FF", respectively.

For each loop in array

❖ In PHP, **foreach** loop is used to iterate over arrays, objects, or other iterable data structures. The syntax of the foreach loop is as follows:.

```
foreach ($array as $value) {
    // Code to be executed for each element in the array
}
```

In this syntax, ***$array*** is the array that you want to loop through, and ***$value*** is a variable that is assigned the value of each element in the array in each iteration of the loop

```
$fruits = array("apple", "banana", "orange", "mango");

foreach ($fruits as $fruit) {

    echo $fruit . "<br>";

}
```

- You can also use the **foreach** loop to iterate over **associative arrays**, in which case you need to specify both the key and the value variables:

```
$person = array(
    "name" => "John",
    "age" => 30,
    "city" => "New York"
);
```

```
foreach ($person as $key => $value) {
    echo "$key: $value <br>";
}
```

- The foreach loop is a powerful tool for iterating over arrays and other iterable data structures in PHP.

Multidimensional Arrays

- A multidimensional array is an array that has more than one dimension.
- A multidimensional array is an array containing one or more arrays.
- PHP supports multidimensional arrays that are two, three, four, five, or more levels deep.

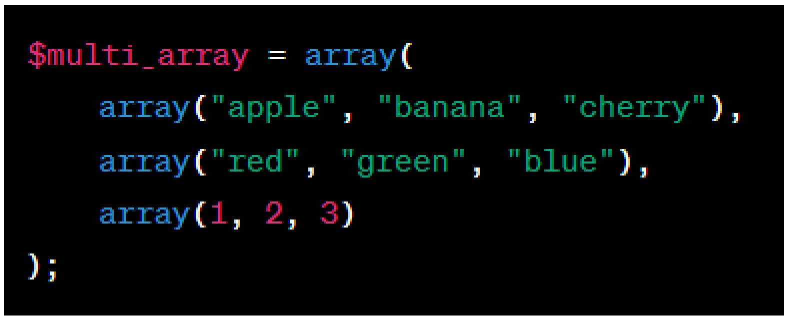
NOTES:

The dimension of an array indicates the number of indices you need to select an element.

For a two-dimensional array you need two indices to select an element

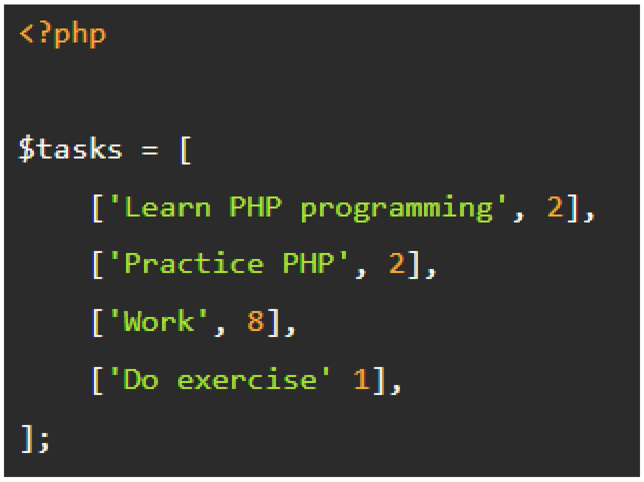
For a three-dimensional array you need three indices to select an element

To create a multidimensional array in PHP, you can use the array() function and specify the elements of the array as nested arrays. Here's an example:



```
$multi_array = array(
    array("apple", "banana", "cherry"),
    array("red", "green", "blue"),
    array(1, 2, 3)
);
```

The following example uses an array of arrays to define a two-dimensional array:



```
<?php

$tasks = [
    ['Learn PHP programming', 2],
    ['Practice PHP', 2],
    ['Work', 8],
    ['Do exercise' 1],
];
```

In the **$tasks array**, the first dimension represents the tasks and the second dimension specifies hour spent for each.

Adding elements to a PHP multidimensional array

- To add an element to a multidimensional array, you use the the following syntax:

  ```
  <?php
  $array[] = [element1, element2, ...];
  ```

  For example, to add an element at the end of the $tasks array, you use the following:

```php
<?php

$tasks = [
        ['Learn PHP programming', 2],
        ['Practice PHP', 2],
        ['Work', 8],
        ['Do exercise', 1],
];

$tasks[] = ['Build something matter in PHP', 2];

print_r($tasks );
```

Removing elements from a PHP multidimensional array

To remove an element from a multidimensional array, you can use the unset() function.

The following example uses the **unset()** function to remove the third element of the $tasks array:

```php
unset($tasks[2]);
```

Note that the **unset()** function doesn't change the array's keys. To **reindex** the key, you can use the **array_splice()** function.

Iterating over elements of a multidimensional array using foreach

To iterate a multidimensional array, you use a nested **foreach** loop like this:

```
<?php
$tasks = [
    ['Learn PHP programming', 2],
    ['Practice PHP', 2],
    ['Work', 8],
    ['Do exercise', 1],
];
foreach ($tasks as $task) {
    foreach ($task as $task_detail) {
        echo $task_detail . '<br>';
    }
}
```

Accessing elements of a multidimensional array

To access an element in an multidimensional array, you use the square brackets ([]):

```
<?php
$array[key][key][key]...
```

For example, to access the number of hour spent for the "Learn PHP programming" task, you use the following code:

```
<?php
$tasks = [
    ['Learn PHP programming', 2],
    ['Practice PHP', 2],
    ['Work', 8],
    ['Do excercise', 1],
];
echo $tasks[0][1];
```

Multi Dimensional Associative Array

In PHP, a multidimensional associative array is an array that contains one or more associative arrays as its elements. Each element of the parent array can be another associative array, which can have its own set of key-value pairs.

To create a multidimensional associative array in PHP, you can use the array() function and specify the key-value pairs of the array as nested associative arrays. Here's an example:

```
$multi_assoc_array = array(
    "fruits" => array("apple" => 1, "banana" => 2, "cherry" => 3),
    "colors" => array("red" => 4, "green" => 5, "blue" => 6),
    "numbers" => array("one" => 7, "two" => 8, "three" => 9)
);
```

In this example, ***$multi_assoc_array*** is a multidimensional associative array that contains three nested associative arrays. The first nested associative array is indexed by the keys "apple", "banana", and "cherry", with corresponding values 1, 2, and 3. The second nested associative array is indexed by the keys "red", "green", and "blue", with corresponding values 4, 5, and 6. The third nested associative array is indexed by the keys "one", "two", and "three", with corresponding values 7, 8, and 9.

You can access elements of a multidimensional associative array by specifying the keys of the element at each level of the array. For example, to access the element with key "banana" in the "fruits" array of the above example, you would use the following code:

```
     echo $multi_assoc_array["fruits"]["banana"];
```

This code accesses the value associated with the key "banana" in the "fruits" array, which is 2.

PHP Array Functions

Function	**Description**
array()	Creates an array
array_change_key_case()	Changes all keys in an array to lowercase or uppercase
array_chunk()	Splits an array into chunks of arrays
array_column()	Returns the values from a single column in the input array
array_combine()	Creates an array by using the elements from one "keys" array and one "values" array
array_count_values()	Counts all the values of an array
array_diff()	Compare arrays, and returns the differences (compare values only)
array_diff_assoc()	Compare arrays, and returns the differences (compare keys and values)
array_diff_key()	Compare arrays, and returns the differences (compare keys only)
array_diff_uassoc()	Compare arrays, and returns the differences (compare keys and values, using a user-defined key comparison function)
array_diff_ukey()	Compare arrays, and returns the differences (compare keys only, using a user-defined key comparison function)
array_fill()	Fills an array with values
array_fill_keys()	Fills an array with values, specifying keys
array_filter()	Filters the values of an array using a callback function
array_flip()	Flips/Exchanges all keys with their associated values in an array
array_intersect()	Compare arrays, and returns the matches (compare values only)
array_intersect_assoc()	Compare arrays and returns the matches (compare keys and values)
array_intersect_key()	Compare arrays, and returns the matches (compare keys only)
array_intersect_uassoc()	Compare arrays, and returns the matches (compare keys and values, using a user-defined key comparison function)
array_intersect_ukey()	Compare arrays, and returns the matches (compare keys only, using a user-defined key comparison function)
array_key_exists()	Checks if the specified key exists in the array

array_keys()	Returns all the keys of an array
array_map()	Sends each value of an array to a user-made function, which returns new values
array_merge()	Merges one or more arrays into one array
array_merge_recursive()	Merges one or more arrays into one array recursively
array_multisort()	Sorts multiple or multi-dimensional arrays
array_pad()	Inserts a specified number of items, with a specified value, to an array
array_pop()	Deletes the last element of an array
array_product()	Calculates the product of the values in an array
array_push()	Inserts one or more elements to the end of an array
array_rand()	Returns one or more random keys from an array
array_reduce()	Returns an array as a string, using a user-defined function
array_replace()	Replaces the values of the first array with the values from following arrays
array_replace_recursive()	Replaces the values of the first array with the values from following arrays recursively
array_reverse()	Returns an array in the reverse order
array_search()	Searches an array for a given value and returns the key
array_shift()	Removes the first element from an array, and returns the value of the removed element
array_slice()	Returns selected parts of an array
array_splice()	Removes and replaces specified elements of an array
array_sum()	Returns the sum of the values in an array
array_udiff()	Compare arrays, and returns the differences (compare values only, using a user-defined key comparison function)
array_udiff_assoc()	Compare arrays, and returns the differences (compare keys and values, using a built-in function to compare the keys and a user-defined function to compare the values)
array_udiff_uassoc()	Compare arrays, and returns the differences (compare keys and values, using two user-defined key comparison functions)
array_uintersect()	Compare arrays, and returns the matches (compare values only, using a user-defined key comparison function)
array_uintersect_assoc()	Compare arrays, and returns the matches (compare keys and values, using a built-in function to compare the keys and a user-defined function to compare the values)

array_uintersect_uassoc()	Compare arrays, and returns the matches (compare keys and values, using two user-defined key comparison functions)
array_unique()	Removes duplicate values from an array
array_unshift()	Adds one or more elements to the beginning of an array
array_values()	Returns all the values of an array
array_walk()	Applies a user function to every member of an array
array_walk_recursive()	Applies a user function recursively to every member of an array
arsort()	Sorts an associative array in descending order, according to the value
asort()	Sorts an associative array in ascending order, according to the value
compact()	Create array containing variables and their values
count()	Returns the number of elements in an array
current()	Returns the current element in an array
each()	Deprecated from PHP 7.2. Returns the current key and value pair from an array
end()	Sets the internal pointer of an array to its last element
extract()	Imports variables into the current symbol table from an array
in_array()	Checks if a specified value exists in an array
key()	Fetches a key from an array
krsort()	Sorts an associative array in descending order, according to the key
ksort()	Sorts an associative array in ascending order, according to the key
list()	Assigns variables as if they were an array
natcasesort()	Sorts an array using a case insensitive "natural order" algorithm
natsort()	Sorts an array using a "natural order" algorithm
next()	Advance the internal array pointer of an array
pos()	Alias of current()
prev()	Rewinds the internal array pointer
range()	Creates an array containing a range of elements
reset()	Sets the internal pointer of an array to its first element
rsort()	Sorts an indexed array in descending order
shuffle()	Shuffles an array
sizeof()	Alias of count()
sort()	Sorts an indexed array in ascending order

uasort()	Sorts an array by values using a user-defined comparison function and maintains the index association
uksort()	Sorts an array by keys using a user-defined comparison function
usort()	Sorts an array by values using a user-defined comparison function

PHP array_push() Function

- The **array_push()** function inserts one or more elements to the end of an array.
- Returns the new number of elements in the array
    - You can add one value, or as many as you like.
    - Even if your array has string keys, your added elements will always have numeric keys

Syntax:

```
array_push(array, value1, value2, ...)
```

```
<?php

$a=array("red","green");

array_push($a,"blue","yellow");

print_r($a); // Array ( [0] => red [1] => green [2] => blue [3] => yellow )

?>
```

```
<?php
$a=array("a"=>"red","b"=>"green");
array_push($a,"blue","yellow");
print_r($a);// Array ( [a] => red [b] => green [0] => blue [1] => yellow )
?>
```

PHP array_pop() Function

- The array_pop() function deletes the last element of an array.
- Returns the last value of array. If array is empty, or is not an array, NULL will be returned.
  Syntax:

```
array_pop(array)
```

```
<?php
$a=array("red","green","blue");
array_pop($a);
print_r($a);// Array ( [0] => red [1] => green )
?>
```

PHP array_shift() Function

- The **array_shift()** function removes the first element from an array, and returns the value of the removed element.
- Returns the value of the removed element from an array, or NULL if the array is empty
- If the keys are numeric, all elements will get new keys, starting from 0 and increases by 1
- Syntax:

```
array_shift(array)
```

```
<?php
$a=array("a"=>"red","b"=>"green","c"=>"blue");
echo array_shift($a);
print_r ($a);// red  Array ( [b] => green [c] => blue )
?>

<?php
$a=array(0=>"red",1=>"green",2=>"blue");
echo array_shift($a);
print_r ($a);// red  Array ( [0] => green [1] => blue )
?>
```

PHP array_search() Function

- The array_search() function search an array for a value and returns the key.
- Returns the key of a value if it is found in the array, and FALSE otherwise. If the value is found in the array more than once, the first matching key is returned.

Syntax:

```
array_search(value, array, strict)
```

Parameter Values

Parameter	**Description**
value	Required. Specifies the value to search for
array	Required. Specifies the array to search in
strict	Optional. If this parameter is set to TRUE, then this function will search for identical elements in the array. Possible values: true false - Default When set to true, the number 5 is not the same as the string 5

```
<?php
$a=array("a"=>"red","b"=>"green","c"=>"blue");
echo array_search("red",$a);// a
?>
```

```
<?php
$a=array("a"=>"5","b"=>5,"c"=>"5");
echo array_search(5,$a,true);// b
?>
```

PHP array_unshift() Function

- The array_unshift() function inserts new elements to an array. The new array values will be inserted in the beginning of the array.
- Returns the new number of elements in the array
    - You can add one value, or as many as you like.
    - Numeric keys will start at 0 and increase by 1. String keys will remain the same.

Syntax:

```
array_unshift(array, value1, value2, value3, ...)
```

```
<?php
$a=array("a"=>"red","b"=>"green");
print_r(array_unshift($a,"blue"));//3
?>
```

```
<?php
$a=array(0=>"red",1=>"green");
array_unshift($a,"blue");
print_r($a);// Array ( [0] => blue [1] => red [2] => green )
?>
```

Sorting Array

Using sort()

- The sort() function sorts an indexed array in ascending order.

  Tip: Use the rsort() function to sort an indexed array in descending order.

  Syntax

  `sort(array, sorttype)`

Parameter	Description
array	Required. Specifies the array to sort
sorttype	Optional. Specifies how to compare the array elements/items. Possible values: 0 = SORT_REGULAR - Default. Compare items normally (don't change types) 1 = SORT_NUMERIC - Compare items numerically 2 = SORT_STRING - Compare items as strings 3 = SORT_LOCALE_STRING - Compare items as strings, based on current locale 4 = SORT_NATURAL - Compare items as strings using natural ordering 5 = SORT_FLAG_CASE - Can be combined with SORT_STRING or SORT_NATURAL to sort strings case-insensitively

Using the PHP sort() function to sort an array of numbers

The following example uses the PHP **sort()** function to sort an array of three numbers:

```
<?php

$numbers = [2, 1, 30,45,12,5];

sort($numbers);
```

Using the PHP **sort()** function to sort an array of strings

The following example uses the **sort()** function to sort an array of strings alphabetically:

```
<?php

$names = ['Bob', 'John', 'Alice','Bipin'];

sort($names, SORT_STRING);

print_r($names);
```

Using the PHP sort() function to sort an array of strings using “natural ordering”

To sort an array of strings in the “natural ordering”, you combine the SORT_STRING and SORT_NATURAL flags. For example:

```
<?php

$ranks = ['A-1', 'A-2', 'A-12', 'A-11'];

sort($ranks, SORT_STRING | SORT_NATURAL);


print_r($ranks);
```

Using asort() and ksort() to Sort Arrays

- The **asort()** function sorts an associative array in ascending order, according to the value.
- The **ksort()** function sorts an associative array in ascending order, according to the key.

Tips:

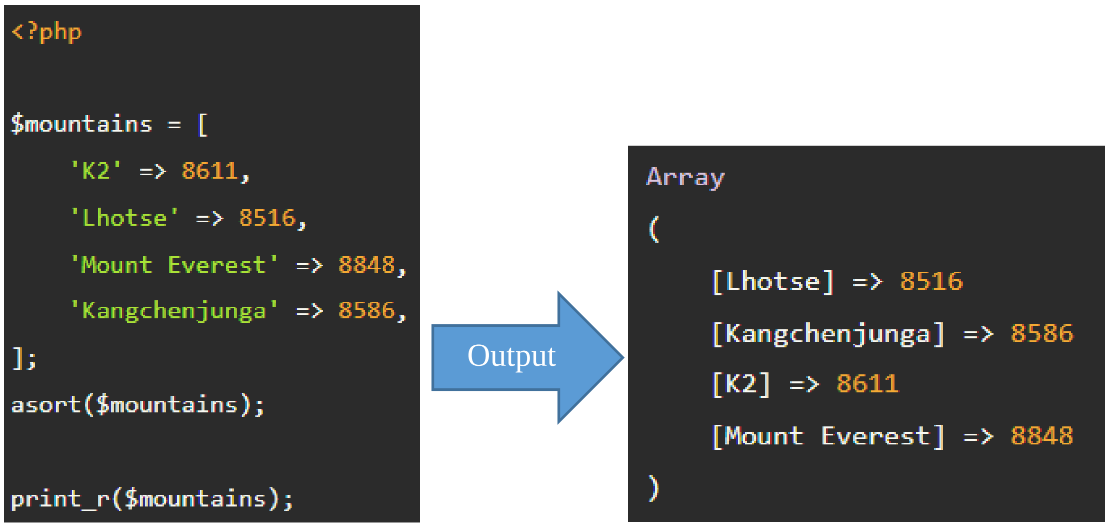
— Use the **arsort()** function to sort an associative array in descending order, according to the value.

— Use the **krsort()** function to sort an associative array in descending order, according to the key.

Syntaxes:

```
asort(array, sorttype)

ksort(array, sorttype)
```

[description of parameters are same as in sort()]


```
<?php

$mountains = [
    'K2' => 8611,
    'Lhotse' => 8516,
    'Mount Everest' => 8848,
    'Kangchenjunga' => 8586,
];
asort($mountains);

print_r($mountains);
```

```
Array
(
    [Lhotse] => 8516
    [Kangchenjunga] => 8586
    [K2] => 8611
    [Mount Everest] => 8848
)
```

```
<?php

$employees = [

    'john' => [
        'age' => 24,

        'title' => 'Front-end Developer'

    ],

    'alice' => [

        'age' => 28,

        'title' => 'Web Designer'

    ],

    'bob' => [

        'age' => 25,

        'title' => 'MySQL DBA'

    ]

];

ksort($employees, SORT_STRING);

print_r($employees);
```

output of this :

```
Array

(

    [alice] => Array
```

```
        (
            [age] => 28
            [title] => Web Designer
        )

    [bob] => Array
        (
            [age] => 25
            [title] => MySQL DBA
        )

    [john] => Array
        (
            [age] => 24
            [title] => Front-end Developer
        )
)
```

Sorting in Reverse

The three different sorting functions—**sort(), asort(),** and **ksort()—**sort an array into ascending order. Each function has a matching reverse sort function to sort an array into descending order. The reverse versions are called **rsort(), arsort(),** and **krsort().**

Reordering Arrays

- For some applications, you might want to manipulate the order of the array in other ways than a sort. The function **shuffle()** randomly reorders the elements of your array. The function **array_reverse()** gives you a copy of your array with all the elements in reverse order.

PHP shuffle() Function

- The shuffle() function randomizes the order of the elements in the array.
- This function assigns new keys for the elements in the array. Existing keys will be removed

```
shuffle(array)
```

Example:

```
<?php

$my_array = array("red","green","blue","yellow","purple");

shuffle($my_array);

print_r($my_array);

?>

<p>Refresh the page to see how shuffle() randomizes the order of
the elements in the array.</p>
```

Another Example

```
<?php

$my_array                                                    =
array("a"=>"red","b"=>"green","c"=>"blue","d"=>"yellow","e"=>"pur
ple");
```

```
shuffle($my_array);

print_r($my_array);

?>

<p>Refresh the page to see how shuffle() randomizes the order of
the elements in the array.</p>
```

Navigating Within an Array: each(), current(), reset(), end(), next(), pos(), and prev()

- The **each()** function returns the current element key and value, and moves the internal pointer forward.

  Syntax:

  ```
  each(array)
  ```

  — This element key and value is returned in an array with four elements. Two elements (1 and Value) for the element value, and two elements (0 and Key) for the element key.

```
<?php

$people = array("Prem", "Jiten", "Gopal", "Chandra");

print_r (each($people));

?>
```

Note: The each() function is deprecated in PHP 7.2.

- The **current()** function returns the value of the current element in an array.
  — Every array has an internal pointer to its "current" element, which is initialized to the first element inserted into the array.
  — If you create a new array, the current pointer is initialized to point to the first element in the array. Calling *current($array_name)* returns the first element

— This function does not move the arrays internal pointer.

Syntax: **current(*array*)**

```
<?php
$people = array("Prem", "Jiten", "Gopal", "Chandra");

echo current($people) . "<br>";//Prem
?>
```

- The **next()** function moves the internal pointer to, and outputs, the next element in the array.

— Calling next($array_name) advances the pointer and then returns the new current element.

Syntax: **next(*array*)**

```
<?php
$people = array("Prem", "Jiten", "Gopal", "Chandra");

echo current($people) . "<br>";// Prem

echo next($people) . "<br>";// Jiten

?>
```

- The **reset()** function moves the internal pointer to the first element of the array.

Syntax: **reset(*array*)**

```
<?php
$people = array("Prem", "Jiten", "Gopal", "Chandra");

echo current($people) . "<br>";// Prem

echo next($people) . "<br>";// Jiten

echo next($people) . "<br>";// Gopal

echo reset($people) . "<br>";// Prem

?>
```

- The **end()** function moves the internal pointer to, and outputs, the last element in the array.

Syntax: **end(*array*)**

```php
<?php
$people = array("Prem", "Jiten", "Gopal", "Chandra");

echo current($people) . "<br>";//Prem

echo end($people) . "<br>";//Chandra
?>
```

- The **prev()** function moves the internal pointer to, and outputs, the previous element in the array.

  Syntax: **prev(*array*)**

```php
<?php
$people = array("Prem", "Jiten", "Gopal", "Chandra");

echo current($people) . "<br>";//Prem

echo end($people) . "<br>";//Chandra

echo prev($people) . "<br>";//Gopal
?>
```

Functions

- A function is a named block of code that performs a specific task.
- Function is a block of reusable code that performs a specific task.
- Function can take input parameters, perform operations, and return a value.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.
- Functions can be used to organize code, make code reusable, and improve code readability.
- PHP has more than 1000 built-in functions, and in addition you can create your own custom functions.
- Calls to functions are not case sensitive, so calls to function_name(), Function_Name(), or FUNCTION_NAME() are all valid and all have the same result.

Creating a User Defined Function in PHP

- Functions in PHP are defined using the "function" keyword, followed by the function name, parentheses for parameters (if any), and curly braces for the function body.

```
function function_name() {
    statement;
}
```

- A function may have zero or more parameters.

```
<?php

function function_name(parameter1, parameter2, ...) {

}
```

Example:

```
function welcome_user($username)

{

     echo 'Welcome ' . $username;

}
```

- Once a function is defined, it can be called using its name.

```
Example: welcome_user('Admin'); //  Welcome Admin

         welcome_user('Customer');// Welcome Customer
```

Default Parameter

In PHP, you can define default parameter values for function parameters. Default parameters allow you to specify a default value that will be used if an argument is not provided when calling the function.

```
     function greet($name = "Guest") {

         echo "Hello, $name!";

     }

     greet(); // Output: Hello, Guest!

     greet("John"); // Output: Hello, John!
```

Functions returning value

A function can return a value. To return a value from a function, you use the return statement.

Examples :

```
function welcome_user($username)

{

    return 'Welcome '. $username . '!';

}


$welcome_message = welcome_user('Admin');

-------------------------------------------------

     function get_sum($x, $y) {

       $sum = $x + $y;

       return $sum;

     }
```

```
        $result = get_sum(10, 20);

        echo $result; // outputs 30
```

- We can also return multiple values from a function by using an array. For example, the following function returns the length and width of a rectangle:

  ```
  function get_rectangle_dimensions($width, $height) {
    $dimensions = array($width, $height);
    return $dimensions;
  }
  ```

  This function can be called like this:

  ```
  $dimensions = get_rectangle_dimensions(10, 20);

  echo $dimensions[0]; // outputs 10

  echo $dimensions[1]; // outputs 20
  ```

PHP Return Type Declarations

- PHP 7 introduced support for return type declarations, allowing you to specify the expected return type of a function. Return type declarations help improve code clarity and can also provide some level of type safety.
- To declare a type for the function return, add a colon (:) and the type right before the opening curly ({)bracket when declaring the function.

  Example

  ```
        <?php declare(strict_types=1); // strict requirement
        function addNumbers(float $a, float $b) : float {
          return $a + $b;
        }
        echo addNumbers(1.2, 5.2);
        ?>
  ```

You can specify a different return type, than the argument types, but make sure the return is the correct type:

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : int {
  return (int)($a + $b);
}
echo addNumbers(1.2, 5.2);
?>
```

Passing Arguments by Reference

- In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed.
- When a function argument is passed by reference, changes to the argument also change the variable that was passed in. To turn a function argument into a reference, the & operator is used.
- In PHP, you can pass a variable by reference to a function, allowing the function to modify the original value of the variable directly. This is known as "pass by reference."

Passing variables by reference can be useful when you need to modify the original value of a variable inside a function and have the changes reflected outside of the function.

// Example: Call by value

```
<?php

function add_five($value) {

  $value += 5;

}

$num = 2;

add_five($num);

echo $num;//2
```

```
//Example: Call by reference

<?php

function add_five(&$value) {

  $value += 5;

}

$num = 2;

add_five($num);

echo $num;//7

?>
```

Form Handling in PHP

- The PHP superglobals $_GET and $_POST are used to collect form-data.

> NOTE: 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.
>
> Some PHP superglobal variables are:
>
> $GLOBALS, $_SERVER, $_REQUEST, $_POST, $_GET, $_FILES, $_ENV, $_COOKIE, $_SESSION

Example uisng 'post' method

```
<html>
<body>

<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>

</body>
</html>
```

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.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>

</body>
</html>
```

Example uisng 'get' method

The same result could also be achieved using the HTTP GET method:

```
<form action="welcome_get.php" method="get">
```

```
//welcome_get.php
Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>
```

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.

GET	POST
▪ Data is appended to the URL as query parameters.	▪ Data is sent in the body of the HTTP request.
▪ Limited data size: The data is limited in size due to URL length restrictions.	▪ No data size limit: POST method doesn't have inherent size limitations, allowing larger data payloads.
▪ Data is visible in the URL: Since the data is part of the URL, it is visible to users, saved in browser history, and can be bookmarked.	▪ Data is not visible in the URL: The data is not directly visible in the URL and is not saved in browser history or bookmarks.
▪ Data is cached: GET requests can be cached by browsers and intermediary servers.	▪ Data is not cached: POST requests are typically not cached by browsers or intermediary servers.
▪ Used for retrieving data: GET is commonly used for retrieving data from a server, such as fetching search results or accessing pages.	▪ Used for submitting data: POST is commonly used for submitting sensitive data, such as login credentials or form data, to a server for processing or storage.

- It's important to note that both GET and POST data can be manipulated by users or malicious scripts, so proper validation and sanitization of user input is essential to prevent security vulnerabilities.

PHP $_GET

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

Example:

Assume we have an HTML page that contains a hyperlink with parameters:

```
<html>
<body>

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

</body>
</html>
```

When a user clicks on the link "Test $GET", the parameters "subject" and "web" are sent to "test_get.php", and you can then access their values in "test_get.php" with $_GET.

The example below shows the code in "test_get.php":

```
<html>
<body>

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

</body>
</html>
```

PHP $_POST

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

- The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the superglobal variable $_POST to collect the value of the input field:

```
<html>
<body>
<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;
  }
}
?>

</body>
</html>
```

PHP $_REQUEST

- PHP $_REQUEST is a PHP superglobal variable which is used to collect data after submitting an HTML form.
- The $_REQUEST superglobal variable contains all the data that is sent to the server, regardless of the HTTP method that is used. This means that the $_REQUEST variable contains data from both the GET and POST methods.
- The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the superglobal variable $_REQUEST to collect the value of the input field:

```
<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;
  }
}
?>
```

The following table shows the differences between the $_GET, $_POST, and $_REQUEST superglobal variables:

Variable	**Data**	**Visible in URL**	**Bookmarkable**
$_GET	Data sent using the GET method	Yes	Yes
$_POST	Data sent using the POST method	No	No
$_REQUEST	All data sent to the server	No	No

You should use the $_REQUEST variable when you need to access all the data that is sent to the server, regardless of the HTTP method that is used. For example, you might use the $_REQUEST variable to validate a user's input, regardless of whether the input was sent using the GET or POST method

☑ $_REQUEST is a superglobal variable in PHP that contains the values of both $_GET and $_POST arrays, as well as the values of cookies if present. It can be used to retrieve user input from HTML forms or URL parameters, regardless of the HTTP method (GET or POST) used to submit the data. However, it's generally recommended to use $_GET or $_POST specifically depending on the intended data source to ensure more secure and predictable behavior.

PHP Form validation

- Form validation is an important step in web development to ensure that user-submitted data meets the required criteria and is safe to process
- Form validation is the process of checking the data entered in a form to make sure it is valid. This helps to prevent errors and ensures that the data is in the correct format.
- We can perform server side validation using PHP.
- Server-side form validation refers to the process of validating user-submitted form data on the server-side, typically using a server-side programming language
- Server-side form validation is crucial because client-side validation using JavaScript can be bypassed or manipulated by users. It is important to validate and sanitize all user-submitted data to ensure data integrity, prevent security vulnerabilities, and provide a better user experience.

- There are several advantages to server-side form validation over client-side form validation.
    - First, server-side form validation is more secure. The data entered in a form is not sent to the client until it has been validated on the server. This means that it is less likely that the data will be intercepted by a malicious user.
    - Second, server-side form validation can be more comprehensive. Client-side form validation is limited by the capabilities of the browser. Server-side form validation, on the other hand, can be as comprehensive as the developer wants it to be.
    - Third, server-side form validation can be more flexible. Client-side form validation is typically implemented using JavaScript. This means that it is not possible to use client-side form validation for forms that are submitted using a non-JavaScript browser (JS disabled). Server-side form validation, on the other hand, can be used for any type of form.

To perform form validation in PHP, you can follow these steps:

- **Retrieve form data:** Use $_POST or $_GET (depending on the form method) to retrieve the submitted form data. For example:

```
$name = $_POST['name'];

$email = $_POST['email'];

// ...
```

- **Define validation rules:** Specify the validation rules for each form field. This may include checking for required fields, validating email addresses, enforcing minimum/maximum length, and more.
- **Perform validation:** Use conditional statements and functions to validate the form data based on the defined rules. If any validation rule fails, store error messages in an array or variable.
- **Display error messages:** If there are any error messages, display them to the user next to the corresponding form fields.

Important functions used in form validation in php

- **empty():** Checks if a variable is empty or not.

  Syntax: *empty($variable)*

  Example:

```
$name = $_POST['name'];

if (empty($name)) {

    echo "Name is required";

}
```

- **isset():** Checks if a variable is set and not null.

  Syntax: *isset($variable)*

  Example:

```
$email = $_POST['email'];

if (!isset($email)) {

    echo "Email is missing";

}
```

- **filter_var():** Validates and filters a specific value using a specified filter.

  Syntax: *filter_var($value, $filter, $options)*

  Example:

```
$email = $_POST['email'];

if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {

    echo "Invalid email format";

}
```

- **preg_match():** Performs a regular expression match on a string.

  Syntax: *preg_match($pattern, $subject)*

  Example

```
$password = $_POST['password'];

if (!preg_match("/^[a-zA-Z0-9]{6,}$/", $password)) {

    echo "Password must be at least 6 characters long and
contain alphanumeric characters only";

}
```

- **strlen():** Gets the length of a string.

  Syntax*: strlen($str)*

  Example:

```
$username = $_POST['username'];
if (strlen($username) < 5) {
    echo "Username must be at least 5 characters long";
}
```

- **htmlspecialchars():** Converts special characters to HTML entities to prevent XSS attacks.

  Syntax: *htmlspecialchars($string, $flags, $encoding)*

  Example:

$comment = $_POST['comment'];

$safeComment = htmlspecialchars($comment, ENT_QUOTES, 'UTF-8');

// Use $safeComment in your code to prevent XSS attacks

- **strip_tags():** Strips HTML and PHP tags from a string.

  Syntax: *strip_tags($string, $allowable_tags)*

  Example:

```
$description = $_POST['description'];
$cleanDescription = strip_tags($description,
'<p><a><strong>');
// Use $cleanDescription in your code to remove unwanted
//tags
```

- **trim():** Removes whitespace or specified characters from the beginning and end of a string.

```
Syntax: trim($string, $characters)

Example:

$zipcode = $_POST['zipcode'];
$cleanZipcode = trim($zipcode);
// Use $cleanZipcode in your code without leading/trailing
whitespace
```

- **ctype_digit():** Checks if a string contains only digits.

  Syntax: *ctype_digit($string)*

  Example:

```
$age = $_POST['age'];
if (!ctype_digit($age)) {
    echo "Age must be a positive integer";
}
```

- **ctype_alpha():** Checks if a string contains only alphabetic characters.

Syntax: *ctype_alpha($string)*

Example:

```
$name = $_POST['name'];
if (!ctype_alpha($name)) {
    echo "Name must contain only letters";
}
```

- **ctype_alnum():** Checks if a string contains only alphanumeric characters.

  Syntax: *ctype_alnum($string)*

  Example:

```
$username = $_POST['username'];

if (!ctype_alnum($username)) {

    echo "Username must contain only letters and numbers";

}
```

- **ctype_upper():** Checks if a string contains only uppercase letters.

  Syntax: ctype_upper($string)

  Example:

```
$input = $_POST['input'];

if (!ctype_upper($input)) {

    echo "Input must be in uppercase";

}
```

- **ctype_lower():** Checks if a string contains only lowercase letters.

  Syntax: ctype_lower($string)

  Example:

```
$input = $_POST['input'];
if (!ctype_lower($input)) {
    echo "Input must be in lowercase";
}
```

- **ctype_space():** Checks if a string contains only whitespace characters.

  Syntax: *ctype_space($string)*

  Example:

```
$input = $_POST['input'];
if (ctype_space($input)) {
    echo "Input must not be empty or contain only whitespace";
}
```

  The ctype_space() function returns true if the input string contains only whitespace characters (spaces, tabs, newlines), and false otherwise. It can be used to validate that an input is not empty or does not consist solely of whitespace.

- Custom validation functions: You can create your own validation functions to perform specific checks on input values, such as checking for a certain format or comparing values.

Forms can be submitted to the web page itself using PHP.

```
<form name="form1" method="post" action="<?php echo
htmlspecialchars($_SERVER['PHP_SELF']); ?>" >
```

- The main purpose of submitting forms to self is for data validation. Data validation means checking for the required data to be entered in the form fields**PHP_SELF** is a variable that returns the current script being executed. You can use this variable in the action field of the form. The action field of the form instructs where to submit the form data when the user presses the submit button. Most PHP pages maintain data validation on the same page as the form itself.
- An advantage of doing this is in case of a change in the website structure, the data validation code for the form, and the form remain together.
- **$_SERVER['PHP_SELF']:** The *$_SERVER["PHP_SELF"]* is a super global variable that returns the filename of the currently executing script. It sends the submitted form data to the same page, instead of jumping on a different page.
- **htmlspecialcharacters():** The **htmlspecialchars()** function converts special characters to HTML entities. It will replace HTML characters like with < and >. This prevents scripting attacks by attackers who exploit the code by inserting HTML or JavaScript code in the form fields.

The isset() function in PHP

It is used to check if a variable is declared and is not NULL. It returns TRUE if the variable exists and is not NULL; else, it returns FALSE.

For example, the following code will print "The variable is set" if the $name variable is set and is not NULL:

```
$name = "John Doe";
if (isset($name)) {
  echo "The variable is set";
} else {
  echo "The variable is not set";
}
```

The isset() function can also be used to check multiple variables at the same time. For example, the following code will print "All variables are set" if all of the $name, $age, and $gender variables are set and are not NULL:

```
$name = "John Doe";
$age = 25;
$gender = "male";
if (isset($name, $age, $gender)) {
  echo "All variables are set";
} else {
  echo "One or more variables are not set";
}
```

The isset() function is a useful tool for ensuring that variables are set before you use them. This can help to prevent errors and unexpected behavior in your code.

PHP $_SERVER

- $_SERVER is a PHP superglobal variable which holds information about headers, paths, and script locations.
- The example below shows how to use some of the elements in $_SERVER:

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

The most important elements that can go inside $_SERVER:

- SCRIPT_NAME: The name of the script that is currently being executed.
- SCRIPT_FILENAME: The full path to the script that is currently being executed.
- SERVER_NAME: The name of the server that is hosting the script.
- SERVER_PORT: The port on which the server is listening.
- REMOTE_ADDR: The IP address of the client that is making the request.
- REQUEST_METHOD: The HTTP method that was used to make the request.
- QUERY_STRING: The query string that was passed in the URL.
- REQUEST_URI: The full URI of the request.

PHP Regular Expression

- A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.
- A regular expression can be a single character, or a more complicated pattern.
- Regular expressions can be used to perform all types of text search and text replace operations.
- In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers.

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

In the example above, / is the delimiter, *Nepal* is the pattern that is being searched for, and *i* is a modifier that makes the search case-insensitive.

The delimiter can be any character that is not a letter, number, backslash or space. The most common delimiter is the forward slash (/), but when your pattern contains forward slashes it is convenient to choose other delimiters such as # or ~.

Regular Expression Functions in PHP

- PHP provides a variety of functions that allow you to use regular expressions. The **preg_match(), preg_match_all()** and **preg_replace()** functions are some of the most commonly used ones:

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

- The **preg_match()** function will tell you whether a string contains matches of a pattern

```
<?php
$str = "Visit cdcsit";
$pattern = "/cdcsit/i";
echo preg_match($pattern, $str); // Outputs 1
?>
```

- The **preg_match_all()** function will tell you how many matches were found for a pattern in a string.

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

- The **preg_replace()** function will replace all of the matches of the pattern in a string with another string**.**

```
<?php
$str = "Visit Microsoft!";
$pattern = "/microsoft/i";
//Outputs "Visit TU!"
echo preg_replace($pattern, "TU", $str); ?>
```

(See about other different concepts about regular expression in unit1 notes)

```
<?php
if($_SERVER['REQUEST_METHOD']=='POST'){
    $name= $age = $success_message="";
    $errors = [];
    if(empty($_POST['name'])){
        $errors['name_error']="Name is required!";
    }else{
        $name = $_POST['name'];
    }
    if(empty($_POST['age'])){
        $errors['age_error']="Age is required!";
    }elseif($_POST['age'] <0||$_POST['age']>140){
        $errors['age_error']="Age must be from 0 t0 140";
    }else{
        $age = $_POST['age'];
    }
    if(empty($errors))
     $success_message = "Form is submitted successfully!";
}
?>
<html>
 <style>
        .error{
            color:red;
        }
 </style>
<form action = <?php echo htmlspecialchars($_SERVER['PHP_SELF'])?> method="post">
    Name :<input type = "text" name = "name">
    <span class = "error" >
        <?php
        if(isset($errors['name_error']))
          echo $errors['name_error'];
        ?>
    </span>
    <br><br>
    Age : <input type = "number" name = "age">
    <span class = "error" >
        <?php
        if(isset($errors['age_error']))
          echo $errors['age_error'];
        ?>
    </span>
    <br><br>
    <input type="submit" value="GO">
</form>
```

```
<?php
//display the submitted values
 if(empty($errors)&& isset($name,$age)){
    echo'<h6>'.$success_message.'</h6>';
    echo'<h4>The submitted values are : </h4>';
    echo'<h5> Name = '.$name.'</h5>';
    echo'<h5> Age = '.$age.'</h5>';
 }
?>
```

PHP Date and Time

- PHP has a built-in function called date() that can be used to format dates and times. The syntax for the date() function is:

```
date(format, timestamp)
```

The *format* parameter is a string that specifies the format of the output date and time. The *timestamp* parameter is an optional parameter that specifies the timestamp of the date and time to be formatted. If no timestamp is specified, the current date and time will be used.

The optional *timestamp* parameter is an int Unix timestamp that defaults to the current local time if timestamp is omitted or null. In other words, it defaults to the value of *time()*.

Returns a formatted date string

```
<?php
  echo "Today's date is :";
  $today = date("d/m/Y");
  echo $today;
?>
```

Formatting options available in date() function for getting date

- The format parameter of the date() function is a string that can contain multiple characters allowing to generate the dates in various formats. Date-related formatting characters that are commonly used in the format string:
  - ☞ **d:** Represents day of the month; two digits with leading zeros (01 or 31).
  - ☞ **D:** Represents day of the week in the text as an abbreviation (Mon to Sun).
  - ☞ **m:** Represents month in numbers with leading zeros (01 or 12).
  - ☞ **M:** Represents month in text, abbreviated (Jan to Dec).
  - ☞ **y:** Represents year in two digits (08 or 23).
  - ☞ **Y:** Represents year in four digits (2008 or 2023).
  - ☞ l (lowercase 'L') - Represents the day of the week
- The parts of the date can be separated by inserting other characters, like hyphens (-), dots (.), slashes (/), or spaces to add additional visual formatting.

The example below formats today's date in three different ways:

```
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
?>
```

Example: Automatic Copyright Year

```
&copy; 2010-<?php echo date("Y");?>
```

Formatting options available in date() function for getting time

☞ The following characters can be used along with the date() function to format the time string:

- ○ **h:** Represents hour in 12-hour format with leading zeros (01 to 12).
- ○ **H:** Represents hour in 24-hour format with leading zeros (00 to 23).
- ○ **i:** Represents minutes with leading zeros (00 to 59).

- **s:** Represents seconds with leading zeros (00 to 59).
- **a:** Represents lowercase antemeridian and post meridian (am or pm).
- **A:** Represents uppercase antemeridian and post meridian (AM or PM).

The example below outputs the current time in the specified format:

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

NOTE: The time() function is used to get the current time as a Unix timestamp (the number of seconds since the beginning of the Unix epoch: January 1, 1970, 00:00:00 GMT).

The date_default_timezone_set() function

- The date_default_timezone_set() function sets the default timezone used by all date/time functions in a script. The syntax for the date_default_timezone_set() function is:

```
date_default_timezone_set(timezone)
```

- The timezone parameter is a string that specifies the timezone to be used. The timezone can be specified in a variety of formats.
- Here are some examples of how to use the date_default_timezone_set() function:

```
// Set the default timezone to UTC
date_default_timezone_set("UTC");

// Set the default timezone to America/New_York
date_default_timezone_set("America/New_York");

// Set the default timezone to Europe/London
date_default_timezone_set("Europe/London");
```

⇒ The date_default_timezone_set() function is a useful function for ensuring that all date/time functions in a script use the same timezone. This can be important for ensuring that dates and times are displayed and interpreted correctly.

⇒ Here are some of the benefits of using the date_default_timezone_set() function:

— Ensures that all date/time functions in a script use the same timezone.

— Prevents errors caused by using different timezones in different date/time functions.

— Makes it easier to debug date/time related errors.

PHP mktime() Function:

— The **mktime()** function is used to create the timestamp for a specific date and time. If no date and time are provided, the timestamp for the current date and time is returned.

```
mktime(hour, minute, second, month, day, year)
```

```
<?php
   echo mktime(23, 21, 50, 11, 25, 2017);// 1511652110
?>
```

The above code creates a time stamp for 25th Nov 2017,23 hrs 21mins 50secs.

Create a Date From a String With strtotime()

— The PHP strtotime() function is used to convert a human readable date string into a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).

```
strtotime(time, now)
```

```
<?php
$d=strtotime("10:30pm April 15 2016");
```

//Created date is 2016-04-15 10:30:00pm

```
echo "Created date is " . date("Y-m-d h:i:sa", $d);?>
```

PHP is quite clever about converting a string to a date, so you can put in various values:

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";
$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

However, strtotime() is not perfect, so remember to check the strings you put in there.

Including Files in PHP

- In PHP, the **require** and **include** statements are used to include external files into your PHP script. These statements allow you to reuse code and organize your project by separating it into multiple 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'; or include('filename');
  require 'filename'; or require('filename');
  ```

  Here, *filename* is name of file with file path and it can be enclosed within single or double quotes

- The **require** statement includes the specified file and throws a fatal error if the file cannot be found or included. This means that if the file is not found, the script execution will stop immediately.

- The **include** statement includes the specified file and produces a warning if the file cannot be found or included. Unlike **require**, the script execution continues even if the file is not found.

  ```
  //vars.php
  <?php

  $color = 'green';
  $fruit = 'apple';

  ?>

  //test.php
  <?php

  echo "A $color $fruit"; // A

  include 'vars.php';

  echo "A $color $fruit"; // A green apple

  ?>
  ```

The **only difference** between the two statements is that if the file cannot be found, the include statement will generate a warning, while the require statement will generate a fatal error.

It is **recommended to** use the *require()* statement if you're including the library files or files containing the functions and configuration variables that are essential for running your application, such as database configuration file.

The include_once and require_once Statements

If you accidentally include the same file (typically functions or classes files) more than one time within your code using the include or require statements, it may cause conflicts. To prevent this situation, PHP provides **include_once** and **require_once** statements. These statements behave in the same way as include and require statements with one exception.

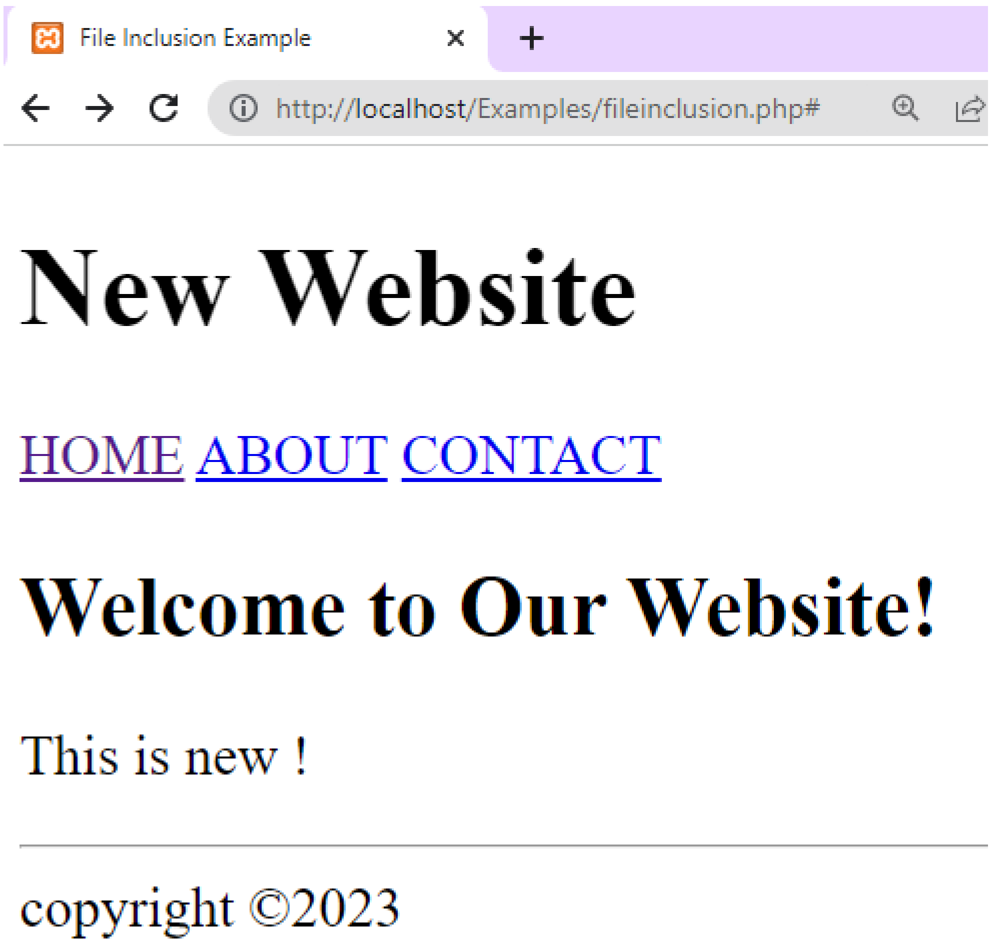
The **include_once** and **require_once** statements will only include the file once even if asked to include it a second time i.e. if the specified file has already been included in a previous statement, the file is not included again.

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>File Inclusion Example</title>
</head>
<body>
<?php include "header.php"; ?>
<?php include "menu.php"; ?>
    <h2>Welcome to Our Website!</h2>
    <p>This is new ! </p>
<?php include "footer.php"; ?>
</body>
</html>
```

//header.php

```
<h1> New Website</h1>

//menu.php
<a href="#">HOME</a>
<a href="about.php">ABOUT</a>
<a href="contact.php">CONTACT</a>

//footer.php
      <hr>
      copyright &copy;<?php echo  date('Y')?>
```



New Website

HOME ABOUT CONTACT

Welcome to Our Website!

This is new !

copyright ©2023

Working with MySQL database

- MySQL and PHP are commonly used together to create dynamic web applications.
- MySQL is a popular open-source relational database management system (RDBMS) that is widely used for storing and managing data. It provides a powerful and scalable solution for handling structured data and supports various features such as data integrity, transaction support, and SQL querying capabilities.
- PHP, on the other hand, is a server-side scripting language that is often used in conjunction with MySQL to create dynamic web pages and applications. PHP can interact with MySQL databases to retrieve, manipulate, and store data.
- The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

- 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

How to connect PHP with MySQL Database?

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

  ⇒ **MySQLi extension**

    ▪ **MySQLi Procedural**

    ▪ **MySQLi Object oriented**

  ⇒ **PDO (PHP Data Objects)**

- **MySQLi extension**
    - MySQLi stands for "MySQL Improved" and is an extension of PHP that provides enhanced features for working with MySQL databases.
    - To use MySQLi in PHP, you need to have the MySQLi extension enabled in your PHP configuration.
    - The MySQLi extension provides two different approaches for interacting with MySQL databases: procedural style and object-oriented style
    - You can then use the MySQLi functions (**MySQLi Procedural)** or instantiate the MySQLi class to interact with the MySQL database (**MySQLi Object Oriented)**
- **PDO (PHP Data Objects)**
    - PDO stands for PHP Data Objects. It is a PHP extension that provides a consistent interface for accessing databases, including MySQL, PostgreSQL, SQLite, and more. PDO allows developers to write database-agnostic code by providing a unified API to interact with different database systems.
    - It supports only object-oriented style

MySQLi vs PDO

- PDO works on 12 different database systems, whereas MySQLi works 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 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.

Some important functions in MySQLi (Procedural) Approach

- ☑ **mysqli_connect():** Establishes a connection to a MySQL database.
  Syntax:

```
$connection = mysqli_connect($host, $username, $password, $database);
```

- ☑ **mysqli_query():** Executes a SQL query on the database.

  Syntax:

  ```
  $result = mysqli_query($connection, $query);
  ```

- ☑ **mysqli_fetch_assoc():** Fetches a row from the result set as an associative array.

  Syntax:

  ```
  $row = mysqli_fetch_assoc($result);
  ```

- ☑ **mysqli_affected_rows():** Returns the number of affected rows by the previous query.

  Syntax:

  ```
  $numRows = mysqli_affected_rows($connection);
  ```

- ☑ **mysqli_num_rows():** Returns the number of rows in the result set.

  Syntax:

  ```
  $numRows = mysqli_num_rows($result);
  ```

- ☑ **mysqli_error():** Returns the last error message.

  Syntax:

  ```
  $errorMessage = mysqli_error($connection);
  ```

(NOTE: Similar function names are there for Object Oriented approach also)

Some important PDO (PHP Data Objects) syntaxes for performing database-related tasks:

- ☑ **Connecting to a Database:**

  Syntax:

```
$pdo = new PDO("mysql:host=$host;dbname=$dbname", $username, $password);
```

☑ **Executing a Query**

Syntax:

```
$statement = $pdo->query($sql);
```

☑ **Preparing and Executing a Query with Parameters:**

Syntax:

```
  $statement = $pdo->prepare($sql);

  $statement->bindValue(':param', $value);

  $statement->execute();
```

☑ **Fetching Results**

Syntax:

```
$result = $statement->fetchAll(PDO::FETCH_ASSOC);
```

☑ **Inserting Data**

Syntax (without prepared statement)

```
  $pdo->exec($sql);
```

Syntax (with prepared statement)

```
        $statement = $pdo->prepare($sql);

        $statement->bindValue(':param1', $value1);

        $statement->bindValue(':param2', $value2);
```

```
$statement->execute();
```

☑ **Updating Data**

Syntax (without prepared statement)

```
$pdo->exec($sql);
```

Syntax (with prepared statement)

```
$statement = $pdo->prepare($sql);

$statement->bindValue(':param1', $value1);

$statement->bindValue(':param2', $value2);

$statement->execute();
```

☑ Deleting Data

Syntax (without prepared statement)

```
$pdo->exec($sql);
```

Syntax (with prepared statement)

```
$statement = $pdo->prepare($sql);

$statement->bindValue(':param', $value);

$statement->execute();
```

Opening Connection to MySQL Database

There are 3 ways in which we can connect to MySQl from PHP as listed above and described below:

❖ **Connection using MySQLi procedural procedure**

- In MySQLi procedural approach instead of creating an instance we can use the mysqli_connect() function available in PHP to establish a connection. This function takes the information as arguments such as host, username , password , database name etc. This function returns MySQL link identifier on successful connection or FALSE when failed to establish a connection

```
<?php
// Database configuration

$servername = "localhost";
$username = "username";
$password = "password";
// Creating connection

$conn = mysqli_connect($servername, $username, $password);

// Checking connection

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

- **Connection using MySQLi object-oriented procedure**

  We can use the MySQLi object-oriented procedure to establish a connection to MySQL database from a PHP script.

  We can create an instance of the **mysqli** class providing all the necessary details required to establish the connection such as host, username, password etc. If the instance is created successfully then the connection is successful otherwise there is some error in establishing connection.

```
<?php
// Database configuration

$servername = "localhost";
$username = "username";
$password = "password";
// Creating connection

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

// Checking connection

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

❖ **Connection using PDO procedure:**

PDO stands for PHP Data Objects. That is, in this method we connect to the database by creating object of PDO class.

```
<?php
// Database configuration

$servername = "localhost";
$username = "username";
$password = "password";


try {
    $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
    // setting 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();
    }
?>
```

The exception class in PDO is used to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

Closing a Connection

☞ When we establish a connection to MySQL database from a PHP script , we should also disconnect or close the connection when our work is finished. Here we have described the syntax of closing the connection to a MySQL database in all 3 methods described above. We have assumed that the reference to the connection is stored in $conn variable.

- MySQLi Procedural: **`mysqli_close($conn);`**
- MySQLi Object-Oriented: **`$conn->close();`**
- PDO: **`$conn = null;`**

Select (Read) Data from a MySQL Database

- SQL SELECT statement can be used to read data from database.
- Let's assume we have a table MyGuests (id,firstname,lastname)

<u>Using MySQLi (procedural)approach</u>

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

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

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

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

mysqli_close($conn);
?>
```

- First, we set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table. The next line of code runs the query and puts the resulting data into a variable called $result.
- Then, the function mysqli_num_rows() checks if there are more than zero rows returned.
- If there are more than zero rows returned, the function mysqli_fetch_assoc() puts all the results into an associative array that we can loop through. The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.

Using MySQLi (object oriented) approach

```
<?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 = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  echo "<table><tr><th>ID</th><th>Name</th></tr>";
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "<tr><td>".$row["id"]."</td><td>".$row["firstname"]."
".$row["lastname"]."</td></tr>";
  }
  echo "</table>";
} else {
  echo "0 results";
}
$conn->close();
?>
```

Using PDO approach

```
<?php
try {
    $pdo = new PDO("mysql:host = localhost;
                      dbname=myDB", "username", "password");
    $pdo->setAttribute(PDO::ATTR_ERRMODE,
                        PDO::ERRMODE_EXCEPTION);
}
catch (PDOException $e) {
    die("ERROR: Could not connect. ".$e->getMessage());
}
try {
    $sql = "SELECT * FROM MyGuests";
    $res = $pdo->query($sql);
    if ($res->rowCount() > 0) {
        echo "<table>";
        echo "<tr>";
        echo "<th>ID</th>";
        echo "<th>Firstname</th>";
        echo "<th>Lastname</th>";

        echo "</tr>";
        while ($row = $res->fetch()) {
            echo "<tr>";
            echo "<td>".$row['id']."</td>";
            echo "<td>".$row['firstname']."</td>";
            echo "<td>".$row['lastname']."</td>";
            echo "</tr>";
        }
        echo "</table>";
        unset($res);
    }
    else {
        echo "No matching records are found.";
    }
}
catch (PDOException $e) {
    die("ERROR: Could not able to execute $sql. "
                                .$e->getMessage());
}
unset($pdo);
?>
```

Insert Data into MySQL Database

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

The following examples add a new record to the "MyGuests" table

MySQLi (Procedural approach)

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

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

$sql = "INSERT INTO MyGuests (id, firstname, lastname)
VALUES (102, 'John', 'Doe')";

//result of the mysqli_query returns Boolean value
if (mysqli_query($conn, $sql)) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

MySQLi (Object Oriented approach)

```php
<?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 = "INSERT INTO MyGuests (id, firstname, lastname)
VALUES (102,'John', 'Doe')";

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

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

<u>PDO Approach</u>

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

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname",
$username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (id, firstname, lastname)
  VALUES (102, 'John', 'Doe', 'john@example.com')";
  // use exec() because no results are returned
  $conn->exec($sql);
  echo "New record created successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}
```

```
$conn = null;
?>
```

Update Data In a MySQL Table

- The UPDATE statement is used to update existing records in a table.

Example using MySQLi procedural approach

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

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

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if (mysqli_query($conn, $sql)) {
  echo "Record updated successfully";
} else {
  echo "Error updating record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Using MySQLi Object Oriented approach

```php
<?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 = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

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

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

<u>Using PDO approach</u>

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

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname",
$username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

  // Prepare statement
  $stmt = $conn->prepare($sql);

  // execute the query
  $stmt->execute();
  // echo a message to say the UPDATE succeeded
  echo $stmt->rowCount() . " records UPDATED successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

Delete Data From a MySQL Table

The DELETE statement is used to delete records from a table

Example: Using MySQLi (procedural) approach

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

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

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if (mysqli_query($conn, $sql)) {
  echo "Record deleted successfully";
} else {
  echo "Error deleting record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example: Using MySQLi (object oriented) approach

```
<?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 delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

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

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

Example: PDO approach

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

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname",
$username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // sql to delete a record
  $sql = "DELETE FROM MyGuests WHERE id=3";

  // use exec() because no results are returned
  $conn->exec($sql);
  echo "Record deleted successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

File Handling in PHP

- ❖ File handling in PHP is the process of reading, writing, creating, and deleting files. PHP provides a variety of functions for file handling, including:
  - **fopen():** Opens a file for reading, writing, or appending.
  - **fread():** Reads data from a file.
  - **fwrite():** Writes data to a file.
  - **fclose():** Closes a file.
  - **unlink():** Deletes a file.
- ❖ File handling is an important part of any web application. You often need to open and process a file for different tasks.

Opening and Closing Files

- The PHP **fopen()** function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate. (This function also has other optional parameters. See the syntax.)
- A file can be opened in following modes:
  - ➢ **r** - Open the file for reading.
  - ➢ **w** - Open the file for writing. If the file already exists, its contents are erased.
  - ➢ **a** - Open the file for appending. If the file does not exist, it is created.
  - ➢ **r+** - Open the file for reading and writing. The file pointer is positioned at the beginning of the file.
  - ➢ **w+** - Open the file for reading and writing. If the file already exists, its contents are erased. The file pointer is positioned at the beginning of the file.
  - ➢ **a+** - Open the file for reading and appending. If the file does not exist, it is created. The file pointer is positioned at the end of the file.

- If an attempt to open a file fails then fopen returns a value of false otherwise it returns a file pointer which is used for further reading or writing to that file.
- After making a changes to the opened file it is important to close it with the **fclose()** function. The **fclose()** function requires a file pointer as its argument and then returns true when the closure succeeds or false if it fails.
- It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

Detail Syntax of fopen() function

```
fopen(filename, mode, include_path, context)
```

Parameter	Description
filename	Required. Specifies the file or URL to open
mode	Required. Specifies the type of access you require to the file/stream. Possible values: ▪ "r" - Read only. Starts at the beginning of the file ▪ "r+" - Read/Write. Starts at the beginning of the file ▪ "w" - Write only. Opens and truncates the file; or creates a new file if it doesn't exist. Place file pointer at the beginning of the file ▪ "w+" - Read/Write. Opens and truncates the file; or creates a new file if it doesn't exist. Place file pointer at the beginning of the file ▪ "a" - Write only. Opens and writes to the end of the file or creates a new file if it doesn't exist ▪ "a+" - Read/Write. Preserves file content by writing to the end of the file ▪ "x" - Write only. Creates a new file. Returns FALSE and an error if file already exists ▪ "x+" - Read/Write. Creates a new file. Returns FALSE and an error if file already exists ▪ "c" - Write only. Opens the file; or creates a new file if it doesn't exist. Place file pointer at the beginning of the file ▪ "c+" - Read/Write. Opens the file; or creates a new file if it doesn't exist. Place file pointer at the beginning of the file ▪ "e" - Only available in PHP compiled on POSIX.1-2008 conform systems.

include_path	Optional. Set this parameter to '1' or 'true' if you want to search for the file in the include_path (in php.ini) as well
context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream

Reading File in PHP

Reading file using fread()

- The fread() function reads from an open file.

  `Syntax: fread(file, length)`
- The first parameter of **fread()** contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.
- The following PHP code reads the "myfile.txt" file to the end

  ```
  fread($myfile,filesize("myfile.txt"));
  ```

Reading single line using fgets()

- The **fgets()** function is used to read a single line from a file.

  `Syntax: fgets(file, length)`
- The example below outputs the first line of the "myfile.txt" file:

  ```
  <?php
  $myfile = fopen("mytext.txt", "r") or die("Unable to open file!");
  echo fgets($myfile);
  fclose($myfile);
  ?>
  ```
- After a call to the **fgets()** function, the file pointer has moved to the next line.

Read single character using fgetc()

- The **fgetc()** function is used to read a single character from a file.

  `Syntax: fgetc(file)`

- The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open
file!");
// Output one character until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

NOTE: The **feof()** function checks if the "end-of-file" **(EOF)** has been reached. The **feof()** function is useful for looping through data of unknown length.

- After a call to the fgetc() function, the file pointer moves to the next character.

Reading file using file_get_contents():

  - The file_get_contents() function provides a simple way to read the entire contents of a file into a string. It takes the file path as a parameter and returns the file contents.
  - The file_get_contents() reads a file into a string.
    Syntax:

```
file_get_contents(path, include_path, context, start, max_length)
```

Parameter	Description
path	Required. Specifies the path to the file to read
include_path	Optional. Set this parameter to '1' if you want to search for the file in the include_path (in php.ini) as well

context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream. Can be skipped by using NULL.
start	Optional. Specifies where in the file to start reading. Negative values count from the end of the file
max_length	Optional. Specifies the maximum length of data read. Default is read to EOF

- This function is the preferred way to read the contents of a file into a string. It will use memory mapping techniques, if this is supported by the server, to enhance performance.

```
$filePath = 'path/to/file.txt';

$content = file_get_contents($filePath);
```

This method is suitable for small to medium-sized files. Keep in mind that if the file is large, loading the entire content into memory may not be optimal.

Writing to File in PHP

Writing into file using fwrite()

- The fopen() function is used to open a file, and fwrite() is used to write content to the opened file. This method provides more control over file operations, such as appending to an existing file or setting file permissions.
- The function will stop at the end of the file (EOF) or when it reaches the specified length, whichever comes first.
- Syntax: `fwrite(`*`file, string, length`*`)`
- Example:

```
$filePath = 'path/to/file.txt';

$content = 'Hello, World!';
```

```
$file = fopen($filePath, 'w'); // Open the file in write mode

fwrite($file, $content); // Write the content to the file

fclose($file); // Close the file
```

In the example above, the 'w' parameter in fopen() specifies that the file should be opened in write mode. If the file already exists, it will be truncated (all existing content will be removed). If you want to append to an existing file instead, use 'a' as the mode.

NOTE: fputs() function can be used interchangeably with fwrite()

Writing to file Using file_put_contents():

- The file_put_contents() function provides a simple way to write data to a file. It takes two parameters: the file path and the content to be written. If the file doesn't exist, file_put_contents() will create it; if it does exist, it will overwrite the existing content.
- `Syntax: file_put_contents(`*`filename, data, mode, context`*`)`

Parameter	Description
filename	Required. Specifies the path to the file to write to. If the file does not exist, this function will create one
data	Required. The data to write to the file. Can be a string, array, or a data stream
mode	Optional. Specifies how to open/write to the file. Possible values: FILE_USE_INCLUDE_PATH - search for filename in the include directory FILE_APPEND - if file already exists, append the data to it - instead of overwriting it LOCK_EX - Put an exclusive lock on the file while writing to it
context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream.

```
$filePath = 'path/to/file.txt';

$content = 'Hello, World!';

file_put_contents($filePath, $content);
```

Random Accessing of File in PHP

- Random accessing of a file in PHP refers to the ability to read or write data at specific positions within a file, rather than sequentially from the beginning to the end. PHP provides functions that allow you to perform random access on files using the file pointer.
- fseek() and ftell() functions are helpful in random accessing of files.
- The fseek() function seeks in an open file. This function moves the file pointer from its current position to a new position, forward or backward, specified by the number of bytes.
- `Syntax: fseek(`*`file, offset, whence`*`)`

Parameter	Description
file	Required. Specifies the open file to seek in
offset	Required. Specifies the new position (measured in bytes from the beginning of the file)
whence	Optional. Possible values: SEEK_SET - Set position equal to offset. Default SEEK_CUR - Set position to current location plus offset SEEK_END - Set position to EOF plus offset (to move to a position before EOF, the offset must be a negative value)

Example: Random Accessing of file

```
$filePath = 'path/to/file.txt';


// Open the file in read/write mode

$file = fopen($filePath, 'r+');


// Move the file pointer to a specific position

fseek($file, 10, SEEK_SET);


// Write data at the current position of the file pointer

fwrite($file, 'Hello');


// Move the file pointer to another position

fseek($file, 0, SEEK_END);


// Read data from the current position of the file pointer

$data = fread($file, 5);


// Print the data

echo $data; // Output: World


// Close the file

fclose($file);
```

File uploading in PHP

- ☑ A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.
- ☑ Information in the *phpinfo.php* page describes the temporary directory that is used for file uploads as *upload_tmp_dir* and the maximum permitted size of files that can be uploaded is stated as *upload_max_filesize*. These parameters are set into PHP configuration file *php.ini*

The process of uploading a file follows these steps −

- The user opens the page containing a HTML form featuring a text files, a browse button and a submit button.
- The user clicks the browse button and selects a file to upload from the local PC.
- The full path to the selected file appears in the text filed then the user clicks the submit button.
- The selected file is sent to the temporary directory on the server.
- The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
- The PHP script confirms the success to the user.

As usual when writing files it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only then process will fail.

An uploaded file could be a text file or image file or any document.

Creating an upload form

Some rules to follow for the HTML form above:

- Make sure that the form uses method="post"

- The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form Without the requirements above, the file upload will not work.

Other things to notice:

- The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

The following HTM code below creates an uploader form. This form is having method attribute set to post and enctype attribute is set to multipart/form-data

```
<html>
  <body>

    <form action = "" method = "POST" enctype = "multipart/form-data">
      <input type = "file" name = "fileToUpload" />
      <input type = "submit"/>

      <ul>
        <li>Sent file: <?php echo $_FILES[' fileToUpload '][ 'name'];  ?>
        <li>File size: <?php echo $_FILES[' fileToUpload '][ 'size'];  ?>
        <li>File type: <?php echo $_FILES[' fileToUpload '][ 'type'] ?>
      </ul>

    </form>

  </body>
</html>
```

```
<?php
   if(isset($_FILES['fileToUpload'])){
      $errors= array();
      $file_name = $_FILES['fileToUpload']['name'];
      $file_size =$_FILES['fileToUpload ']['size'];
      $file_tmp =$_FILES['fileToUpload']['tmp_name'];
      $file_type=$_FILES['fileToUpload']['type'];
$file_ext=strtolower(end(explode('.',$_FILES['fileToUpload']['name'])));

      $extensions= array("jpeg","jpg","png");

      if(in_array($file_ext,$extensions)=== false){
         $errors['extensionError']="extension not allowed, please choose a
JPEG or PNG file.";
      }

      if($file_size > 2*1024*1024){
         $errors['sizeError']='Max File size is 2 MB';
      }

      if(empty($errors)==true){
         move_uploaded_file($file_tmp,"images/".$file_name);
         echo "Success";
      }else{
         print_r($errors);
      }
   }
?>
```

Creating an upload script

- There is one global PHP variable called $_FILES. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was my_file, then PHP would create following five variables –

  $_FILES['my_file']['tmp_name']

  – the uploaded file in the temporary directory on the web server.

$_FILES['my_file']['name']

– the actual name of the uploaded file.

$_FILES['my_file']['size']

– the size in bytes of the uploaded file.

$_FILES['my_file']['type']

– the MIME type of the uploaded file.

$_FILES['my_file']['error']

– the error code associated with this file upload.

PHP move_uploaded_file() Function

- ❖ The move_uploaded_file() function moves an uploaded file to a new destination.
- ❖ This function only works on files uploaded via PHP's HTTP POST upload mechanism.
- ❖ If the destination file already exists, it will be overwritten.

Syntax

move_uploaded_file(file, dest)

Parameter	Description
file	Required. Specifies the filename of the uploaded file
dest	Required. Specifies the new location for the file

Returns true on success and false in failure

//File: uploadform.html

```
<form action="uploader.php" method="post" enctype="multipart/form-
data">
    Select File:
    <input type="file" name="fileToUpload"/>
    <input type="submit" value="Upload Image" name="submit"/>
```

```
</form>
    // File: uploader.php
<?php
$target_path = "/uploads";
$target_path = $target_path.basename( $_FILES['fileToUpload']['name']);

if(move_uploaded_file($_FILES['fileToUpload']['tmp_name'], $target_path))
{
    echo "File uploaded successfully!";
} else{
    echo "Sorry, file not uploaded, please try again!";
}
?>
```

PHP basename() Function

⇒ The *basename()* function returns the filename from a path

Syntax

```
basename(path, suffix)
```

Parameter	Description
path	Required. Specifies a file path
suffix	Optional. A file extension. If the filename has this file extension, the file extension will be cut off

⇒ Returns The filename of the specified path

```
<?php
$path = "/testweb/home.php";

//Show filename
echo basename($path) ."<br/>";
```

```
//Show filename, but cut off file extension for ".php" files
echo basename($path,".php");
?>
```

The output of the code above will be:

```
home.php
home
```

PHP pathinfo() Function

⇒ The **pathinfo()** function returns information about a file path.

Syntax

pathinfo(path, options)

Parameter	Description
path	Required. Specifies the path to be checked
options	Optional. Specifies which array element to return. If not specified, it returns all elements. Possible values: PATHINFO_DIRNAME - return only dirname PATHINFO_BASENAME - return only basename PATHINFO_EXTENSION - return only extension PATHINFO_FILENAME - return only filename

⇒ If the option parameter is omitted, it returns an associative array with dirname, basename, extension, and filename. If the option parameter is specified, it returns a string with the requested element. FALSE on failure

Example:

```
<?php
print_r(pathinfo("/testweb/test.txt",PATHINFO_BASENAME));
?>
```

The output of the code above will be:

```
test.txt
```

File uploading compelled example:

— At first, define HTML form elements to allow the user to select a file they want to upload. Make sure <form> tag contains the following attributes.

- method="post"
- enctype="multipart/form-data"
- Also, make sure <input> tag contains type="file" attribute.

//fileuploadform.html

```
<form action="fileUpload.php" method="post" enctype="multipart/form-data">
       Select File to Upload:
       <input type="file" name="file">
       <input type="submit" name="submit" value="Upload">
</form>
```

The above file upload form will be submitted to the server-side script (upload.php) for uploading the selected file to the server.

The following **upload.php** file contains server-side code to handle the file upload process using PHP.

- ☞ PHP provides a built-in function named **move_uploaded_file()** that moves an uploaded file to a new location. Using **move_uploaded_file()** function we can upload a file in PHP.
- ☞ In the **$targetDir** variable, specify the desire upload folder path where the uploaded file will be stored on the server.
- ☞ In the **$allowedTypes** variable, specify the types of the file in array format that you want to allow to upload.
- ☞ Use PHP **move_uploaded_file()** function to upload file to the server.

```
<?php
//file upload path
$targetDir = "uploads/";
$status = [];
$fileName = basename($_FILES["fileToUpload"]["name"]);
$targetFilePath = $targetDir . $fileName;
$fileType = pathinfo($targetFilePath,PATHINFO_EXTENSION);
$fileSize = $_FILES["fileToUpload"]["size"];
if(isset($_POST["submit"]) && !empty($_FILES["fileToUpload"]["name"])) {
    //allow certain file formats
    $allowedTypes = array('jpg','png','jpeg','gif');
    $isAllowed= in_array($fileType, $allowedTypes);
    if(!$isAllowed){
      $status['type_error'] ="Only JPG, JPEG, PNG, and GIF files are
allowed " ;
    }
    //max file size is 2 MB
    $maxSize = 2*1024*1024;
    $isSizeOk = true;
    if($fileSize>$maxSize){
        $isSizeOk  = false;
        $status['size_error']="Max File Size is 2 MB";
    }
    if($isAllowed && $isSizeOk){
        //upload file to server
        $res= move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],
$targetFilePath);
        if($res){
            $status['success_msg'] = "The file ".$fileName. " has been
uploaded.";
        }else{
            $status['unsuccess_msg'] = "Sorry, there was an error uploading
your file.";
        }
    }
    if(isset($status['success_msg'])){
        echo $status['success_msg'];
    }elseif(isset($status['unsuccess_msg'])){
        echo $status['unsuccess_msg'];
    }else{
        //display error messages
        foreach($status as $err_key =>$err_msg){
            echo $err_key.": ".$err_msg;
            echo "<br>";
        }
    }
}
?>
```

Working with Session

- A session refers to a period of interaction between a user and a computer system or application. It represents a logical connection between the user and the system, allowing the system to track and maintain information about the user's activities and state across multiple requests.
- HTTP is a "stateless" protocol which means each time a client retrieves a Webpage, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.
- Session variable is one of the ways to make HTTP "statefull". Session variable does this by storing user information to be used across multiple pages (e.g. username). By default, session variables last until the user closes the browser.
- Session variables hold information about one single user, and are available to all pages in one application.

  Session variables are a crucial component in web applications for maintaining user-specific information across multiple requests and pages. They allow the server to remember data associated with a particular user throughout their session, which typically spans multiple HTTP requests.

- The idea of session control is to be able to track a user during a single session on a website. If you can do this, you can easily support logging in a user and showing content according to her authorization level or personal preferences. Additionally, you can track the user’s behavior, and you can implement shopping carts, among many other actions.
- **PHP includes a rich set of native session control functions, as well as a single $_SESSION superglobal available for your use.**
- Here's an example to illustrate the need for session variables:
  - Let's consider a scenario where you have developed an e-commerce website that allows users to add items to their shopping cart and proceed to checkout. To provide a personalized experience, you need to remember the

items added by a specific user while they navigate through different pages of the website.

1. User visits the website and adds some items to their shopping cart.
2. The web application creates a session for the user and assigns a unique session identifier (usually stored in a cookie or passed through URL rewriting).
3. The server stores the user's shopping cart information in the session, associating it with the user's session identifier.
4. The user proceeds to browse other pages on the website, such as product categories, reviews, or account settings.
5. During this process, the server can access the session identifier from the user's subsequent requests and retrieve their shopping cart data from the corresponding session.
6. The server dynamically generates each page according to the user's session data, displaying their selected items, cart total, and any other personalized information.
7. If the user decides to remove an item or update the quantity in the shopping cart, the server modifies the session data accordingly.
8. Finally, when the user proceeds to checkout, the server retrieves the complete shopping cart information from the session to process the payment and complete the order.

- In this example, session variables play a crucial role in maintaining the state of the user's shopping cart throughout their session on the website. Without session variables, the server would not have a way to associate the user's shopping cart with their subsequent requests and provide a personalized experience.

Why sessions are useful?

Sessions play a crucial role in web applications for several reasons:

- **User identification and authentication**: Sessions enable web applications to identify and authenticate users. When a user logs in, a session is created and associated with their credentials. Subsequent requests from the user contain the session ID, allowing the server to recognize and verify the user's identity without requiring them to authenticate with each request.
- **Personalization and customization:** Sessions allow web applications to personalize the user experience by remembering user-specific preferences, settings, and customization options. For example, a session can store a user's language preference, theme selection, or saved filters. This enables the application to provide a tailored experience to individual users.
- **Shopping carts and e-commerce**: Sessions are widely used in e-commerce applications to maintain the state of a user's shopping cart. As users browse and add items to their cart, the session keeps track of the selected products, quantities, and other relevant information. This ensures that the user's cart contents persist across pages and allows for seamless checkout.
- **Multi-step processes and form submissions:** In web applications that involve multi-step processes or form submissions, sessions are used to retain the entered data throughout the steps. For instance, when filling out a multi-page form, the session allows the application to collect and store the entered information until the final submission.
- **Security and authorization:** Sessions contribute to the security of web applications by allowing for centralized control over user access and permissions. Session data can include authorization details, roles, and permissions associated with the user. This information can be used to validate user actions and enforce access restrictions to sensitive resources.
- **Stateful communication:** Web applications often require maintaining state information across multiple requests to provide a seamless and interactive user experience. Sessions enable the storage of this state data on the server side, allowing

the application to maintain context, remember user interactions, and provide continuity between different pages or actions.

Implementing Simple Sessions

The basic steps of using sessions in PHP are

1. Starting a session
2. Registering session variables
3. Using session variables
4. Deregistering variables and destroying the session

Note that these steps don't necessarily all happen in the same script, and some of them happen in multiple scripts.

Starting a session and Registering session variables

⇒ A session is started with the **session_start()** function. (This step is also known as **Session Creation)**
It initializes a new session or resumes an existing one if available.

⇒ Session variables are set with the PHP superglobal variable: $_SESSION.
(Creating keys and values for associative array $_SESSION") (This step is also known as **Setting Session Variables)**

— You can set session variables by assigning values to the $_SESSION super-global array. These variables can hold any data you need to store.

— Example:

```
<?php

$_SESSION['username'] = 'JohnDoe';

$_SESSION['role'] = 'admin';

?>
```

Next Example: Let's create a new page called "session_example1.php". In this page, we start a new PHP session and set some session variables:

```
<?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>
```

NOTE : The **session_start()** function must be the very first thing in your document. Before any HTML tags

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

The location of the temporary file is determined by a setting in the **php.ini** file called **session.save_path**. Before using any session variable make sure you have setup this path.

When a session is started following things happen −

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as *3c7foj34c3jj973hjkop2fc937e3443*.
- A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie *sess_3c7foj34c3jj973hjkop2fc937e3443*.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the **PHPSESSID** cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Using session variables

You can retrieve the values stored in session variables by accessing the $_SESSION superglobal array.

```
<?php

session_start();

if (isset($_SESSION['username'])) {

    $username = $_SESSION['username'];

    echo "Welcome back, $username!";

} else {

    echo "Welcome, guest!";

}

?>
```

Next Example:

Now we create another page called "session_example2.php". From this page, we will access the session information we set on the first page ("session_example1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Also notice that all session variable values are stored in the global $_SESSION variable:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
```

```
?>

</body>
</html>
```

Modify a PHP Session Variable

To change a session variable, just overwrite it.

Example:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>
```

Deregistering variables and destroying the session

Although the web server will terminate the session by default upon closing the browser, you can also destroy it manually. Following functions can help you achieve this.

- **unset()** is used to remove a specific session variable (or any variable in general).
- **session_unset()** clears all session variables but does not destroy the session itself.
- **session_destroy()** completely destroys the session, including all session variables, and removes the session cookie.

The **unset()** function in PHP is used to destroy a specific variable or variables, including session variables. It allows you to remove the value and existence of a variable. However, using **unset()** on session variables does not destroy the entire session or clear other session data.

Example:

```
<?php

session_start();

// Unset a specific session variable

unset($_SESSION['username']);

?>
```

The **session_unset()** function is used to remove all session variables. It clears the values of all variables stored in the $_SESSION superglobal array, but it does not destroy the session itself or remove the session cookie.

Example:

```
<?php

session_start();

// Unset all session variables

session_unset();

?>
```

The **session_destroy()** function is used to completely destroy a session. It removes all session data, including variables, and invalidates the session ID. It also removes the session cookie from the user's browser. After calling **session_destroy(),** a new session can be started.

```
<?php

session_start();

// Clear all session variables

// Destroy the session
```

```
            session_destroy();

            ?>
```

In most cases, when you want to end a session and remove all session data, it's recommended to use **session_destroy()**. It ensures that the session is properly destroyed and can be followed by a new session if needed.

Login/Logout example

Here we use three files.

mylogin.php -- for login form, if login is valid session is created

welcome.php -- this page will open only if login is valid

logout.php -- this destroys the session and redirects to login page

```
//myLogin.php

<?php
session_start();
if(isset($_POST['loginform'])){
  if($_POST['username']=="user1" && $_POST['password']=="pass123" ){
    $_SESSION['user_name']= $_POST['username'];
    header('Location:/Examples/welcome.php');
  }else{
    echo'invalid user';
  }
}
?>
<form action="" method="post">
    Username: <input type="text" name ="username">
    <br><br>
    Password: <input type="password" name ="password">
    <br><br>
    <input type="submit" value="GO" name = "loginform">

</form>

//welcome.php
```

```
<?php
  session_start();
  if(isset($_SESSION['user_name'])){
    echo '<a href= "/Examples/logout.php">Logout</a><br><br>';

    echo'welcome '. $_SESSION['user_name']. 'Its your place';
    print_r($_SESSION);
  }
  else{
    echo 'login first';
    echo '<a href="/Examples/mylogin.php">Login</a>';
  }
?>
```

//Logout.php

```
<?php
  session_start();
  session_destroy();
  header('Location:/Examples/mylogin.php');
?>
```

Turning on Auto Session

You don't need to call start_session() function to start a session when a user visits your site if you can set session.auto_start variable to 1 in **php.ini** file.

Sessions without cookies

There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

```
session_start();

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

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

   echo ( $msg );
?>

<p>
   To continue  click following link <br />

   <a href = "nextpage.php? <?php echo htmlspecialchars(SID);
?>">click</a>
</p>
```

Cookies in PHP

- A cookie is a small piece of information that scripts can store on a client-side machine.
- Cookies are text files stored on the client computer and they are kept for user tracking purpose. PHP transparently supports HTTP cookies.
- Cookies are usually set in an HTTP header but JavaScript can also set a cookie directly on a browser.
- 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.

  All of the data in the cookie is automatically sent to the server each time the browser requests a page from the server.

- There are two main types of cookies: **Session Cookies** and **Persistent Cookies**

Session Cookies: Session cookies, also known as transient cookies, are temporary cookies that are stored in the user's browser for the duration of their session on a website. These cookies are typically used to store session identifiers or other session-specific information. Session cookies are automatically deleted when the user closes the browser or ends the session, and they do not persist beyond that session. They are commonly used for session management, user authentication, and maintaining user state during a browsing session.

Persistent Cookies: Persistent cookies, also referred to as permanent cookies or tracking cookies, are cookies that are stored on the user's browser for an extended period, even after the browser is closed. These cookies have an expiration date set in the future and remain on the user's machine until that date or until the user manually deletes them. Persistent cookies are often used for purposes such as remembering user preferences, personalization, targeted advertising, and user tracking. They can store information like login details, language preferences, user demographics, or previous browsing activity.

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

Why cookies are used?

⇒ Cookies are used for various purposes, including:

Session Management: Cookies are commonly used to manage user sessions on websites. When a user logs in to a website, a session cookie is often created to keep track of their authentication status. This allows the user to navigate different pages of the website without needing to authenticate again for each request.

Personalization: Cookies can be used to personalize the user experience on a website. By storing user preferences or past interactions in cookies, websites can customize content, layouts, and settings to match the user's preferences.

Tracking and Analytics: Cookies are widely used for tracking user behavior and gathering analytics data. Website owners can set cookies to track user interactions, monitor session duration, analyze traffic patterns, and gather information for marketing or optimization purposes.

Shopping Carts and E-commerce: Cookies are often used in e-commerce websites to store information about a user's shopping cart. By storing the selected items and their quantities in a cookie, the website can retain the cart contents even if the user navigates away from the page or closes the browser.

Advertising and Remarketing: Cookies play a significant role in targeted advertising and remarketing. Ad networks and advertisers use cookies to track user behavior across websites, allowing them to display personalized ads based on the user's interests or previous interactions.

Cross-Site Communication: Cookies can be used for communication between different websites or domains. Websites can set cookies with shared values to exchange information or authenticate users across different domains.

Setting Cookie in PHP

To set a cookie in PHP, the **setcookie()** function is used. The **setcookie()** function needs to be called prior to any output generated by the script otherwise the cookie will not be set.

Syntax:

setcookie($name, $value, $expire, $path, $domain, $security,$httponly);

- The **setcookie()** function returns true on success or false on failure.

The parameters for the setcookie() function are as follows:

$name (required): The name of the cookie.

$value (optional): The value of the cookie. By default, it is an empty string.

$expires (optional): The expiration time of the cookie, specified as a Unix timestamp. If set to 0, or omitted, the cookie will expire at the end of the session.

$path (optional): The path on the server for which the cookie will be available. If set to "/", the cookie will be available for the entire domain. If set to a specific path, the cookie will only be available within that path and its subdirectories.

$domain (optional): The domain for which the cookie will be available. By default, the cookie will be available for the current domain.

$secure (optional): Indicates whether the cookie should only be transmitted over a secure HTTPS connection. By default, it is set to false.

$httponly (optional): Indicates whether the cookie should be accessible only through the HTTP protocol and not through client-side scripts. By default, it is set to false.

```
setcookie("mycookie", "example value", time() + 3600, "/path", "example.com", true, true);
```

This example sets a cookie named "mycookie" with the value "example value". It will expire in one hour (3600 seconds) and will be available for the path "/path" on the domain "example.com". The cookie will only be transmitted over a secure connection (HTTPS) and will be accessible only through the HTTP protocol.

Another Example :

```
<?php
```

```
// Set the cookie name and value

$cookie_name = "user";

$cookie_value = "John Doe";

// Set the cookie expiry

$cookie_expiry = time() + (30*24*60*60); // 30 days

// Set the cookie

setcookie($cookie_name, $cookie_value, $cookie_expiry);

?>
```

Accessing Cookie Values

For accessing a cookie value, the PHP **$_COOKIE** superglobal variable is used. It is an associative array that contains a record of all the cookies values sent by the browser in the current request. The records are stored as a list where the cookie name is used as the key

Example:

```
      <?php

      // Get the cookie value

      $user = $_COOKIE["user"];

      // Print the cookie value

      echo $user;

      ?>
```

Checking Whether a Cookie Is Set Or Not:

It is always advisable to check whether a cookie is set or not before accessing its value. Therefore to check whether a cookie is set or not, the PHP **isset()**

```
<?php
```

```
// Check if the cookie is set

if (isset($_COOKIE["user"])) {

  // The cookie is set

      echo $user;


} else {

  // The cookie is not set

      echo 'no user!';

?>
```

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

Deleting Cookies:

- The setcookie() function can be used to delete a cookie. To delete a cookie, use the setcookie() function with an expiration date in the past

  Example

  ```
  <?php
  // set the expiration date to one hour ago
  setcookie("user", "", time() - 3600);
  ?>
  ```
- If the expiration time of the cookie is set to 0 or omitted, the cookie will expire at the end of the session i.e. when the browser closes.
- The same path, domain, and other arguments should be passed that were used to create the cookie in order to ensure that the correct cookie is deleted.

Sending Emails using PHP

- There are different options to send emails in PHP. Here we are focusing on built-in mail() function
- The mail() function allows you to send emails directly from a script.
- Syntax

  `mail(to,subject,message,headers,parameters);`

Parameter	Description
to	Required. Specifies the receiver / receivers of the email
subject	Required. Specifies the subject of the email. Note: This parameter cannot contain any newline characters

message	Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters.
	
	Windows note: If a full stop is found on the beginning of a line in the message, it might be removed. To solve this problem, replace the full stop with a double dot:
	`<?php`
	`$txt = str_replace("\n.", "\n..", $txt);`
	`?>`

headers	Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n). Note: When sending an email, it must contain a From header. This can be set with this parameter or in the php.ini file.
parameters	Optional. Specifies an additional parameter to the sendmail program (the one defined in the sendmail_path configuration setting). (i.e. this can be used to set the envelope sender address when using sendmail with the -f sendmail option)

- Note that the mail() function uses the underlying mail configuration on your server to send the email. Make sure your server is properly configured to send emails. Additionally, the function may return true even if the email is not actually delivered to the recipient, as it only indicates that the email was accepted for delivery by the server.

  It's also worth mentioning that the mail() function has some limitations and may not be suitable for all use cases. Consider using a dedicated email library like PHPMailer or SwiftMailer if you need more advanced email features.

  Example:

```
<?php
$to_email = "receipient@gmail.com";
$subject = "Simple Email Test via PHP";
$body = "Hi, This is test email send by PHP Script";
$headers = "From: Sender Name";

if (mail($to_email, $subject, $body, $headers)) {
    echo "Email successfully sent to $to_email...";
} else {
    echo "Email sending failed...";
}
```

INSTRUCTIONS TO ENABLE MAILING FEATURE IN LOCALHOST (XAMPP)

Go to C:\xampp\php and open the php.ini file.

Find [mail function] by pressing ctrl + f.

Search and pass the following values:

```
SMTP=smtp.gmail.com
smtp_port=587
sendmail_from = YourGmailId@gmail.com
sendmail_path = "\"C:\xampp\sendmail\sendmail.exe\" -t"
```

Now, go to C:\xampp\sendmail and open the sendmail.ini file.

Find [sendmail] by pressing ctrl + f.

Search and pass the following values

```
smtp_server=smtp.gmail.com
smtp_port=587
error_logfile=error.log
debug_logfile=debug.log
auth_username=YourGmailId@gmail.com
auth_password=Your-Gmail-Password //generate app password for sender mail  and use
```

TO GENERATE app password in gmail

Click "manage your google account " then "Security"

Enable 2 Step verification

Click on "2 Step verification"

Click on "App passwords" and generate by selecting - other app (give any name) and click on "GENERATE" then you will get app password.