



# UNIT-2

## PHP

J Vishnu Priyanka  
Assistant Professor©  
Dept of CSE  
RGUKT-SRIKAKULAM.

# **Contents:**

- Server Programming: PHP basics: PHP Syntax,
- Variables
- Constants
- Data Types
- Strings
- Conditional and Control Structures.
- PHP GET and POST.
- PHP Advanced: include files, File system
- Parsing directories
- File upload and download
- Sessions
- Form handling
- JSON Parsing



# Server Programming: PHP basics

- The PHP Hypertext Pre-processor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases.
- PHP is basically used for developing web based software applications.
- Rasmus Lerdorf unleashed the first version of PHP way back in 1994
- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML.
- It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server
- PHP supports a large number of major protocols such as POP3, IMAP



## COMMON USES OF PHP:

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



A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

## Basic PHP Syntax

- A PHP script can be placed anywhere in the document.

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

- **<?php**  
    **// PHP code goes here**  
    **?>**
- The default file extension for PHP files is ".php".
- A PHP file normally contains HTML tags, and some PHP scripting code.
- Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:



## Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>My first PHP page</h1>
```

```
<?php
```

```
echo "Hello World!";
```

```
?>
```

```
</body>
```

```
</html>
```

**Note: PHP statements end with a semicolon (;).**



# PHP CASE SENSITIVITY

- In PHP, keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are not case-sensitive.
- In the example below, all three echo statements below are equal and legal:

## Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php
```

```
ECHO "Hello World!<br>";
```

```
echo "Hello World!<br>";
```

```
EcHo "Hello World!<br>";
```

```
?>
```

```
</body>
```

```
</html>
```



However; all variable names are case-sensitive!

Look at the example below;

only the first statement will display the value of the \$color variable! This is because \$color, \$COLOR, and \$coLOR are treated as three different variables:

Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php
```

```
$color = "red";
```

```
echo "My car is " . $color . "<br>";
```

```
echo "My house is " . $COLOR . "<br>";
```

```
echo "My boat is " . $coLOR . "<br>";
```

```
?>
```

```
</body>
```

```
</html>
```





# Comments in PHP

PHP supports several ways of commenting:

## Example

- Syntax for single-line comments:
- `<!DOCTYPE html>`  
`<html>`  
`<body>`  
  
`<?php`  
`// This is a single-line comment`  
  
`# This is also a single-line comment`  
`?>`  
  
`</body>`  
`</html>`



## Example

- Syntax for multiple-line comments:

- `<!DOCTYPE html>`

`<html>`

`<body>`

`<?php`

`/*`

**This is a multiple-lines comment block  
that spans over multiple  
lines**

`*/`

`?>`

`</body>`

`</html>`



## Example

- Using comments to leave out parts of the code:

- `<!DOCTYPE html>`

`<html>`

`<body>`

`<?php`

**// You can also use comments to leave out parts of a code line**

`$x = 5 /* + 15 */ + 5;`

`echo $x;`

`?>`

`</body>`

`</html>`



# PHP VARIABLES

## Creating (Declaring) PHP Variables

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

### Example

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



# PHP Variables

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume).

## Rules for PHP variables:

- A variable starts with the \$ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
- Variable names are case-sensitive (\$age and \$AGE are two different variables)



## Output Variables

- The PHP echo statement is often used to output data to the screen.
- The following example will show how to output text and a variable:

### Example

```
<?php  
$txt = "Srikakulam";  
echo "RGUKT $txt!";  
?>
```

The following example will output the sum of two variables:

### Example

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



## PHP Constants

- A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
- A valid constant name starts with a letter or underscore (no \$ sign before the constant name).

## Create a PHP Constant

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

### Syntax

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

### Parameters:

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



## Example

Create a constant with a **case-sensitive** name:

```
<?php  
define("GREETING", "Welcome to RGUKT Srikakulam!");  
echo GREETING;  
?>
```

## Example

Create a constant with a **case-insensitive** name:

- ```
<?php  
define("GREETING", "Welcome to RGUKT Srikakulam!", true);  
echo greeting;  
?>
```





## PHP Constant Arrays

In PHP7, you can create an Array constant using the define() function.

### Example

Create an Array constant:

```
<?php  
define("cars", [  
    "Alfa Romeo",  
    "BMW",  
    "Toyota"  
]);  
echo cars[0];  
?>
```



## Constants are Global

- Constants are automatically global and can be used across the entire script.

### Example

- This example uses a constant inside a function, even if it is defined outside the function:

```
<?php
define("GREETING", "Welcome toRGUKT Srikakulam!");

function myTest() {
    echo GREETING;
}

myTest();
?>
```



# PHP Data Types

Variables can store data of different types, and different data types can do different things. PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource



## PHP String

- A string is a sequence of characters, like "Hello world!".
- A string can be any text inside quotes. You can use single or double quotes:

### Example

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

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



# PHP Integer

Rules for integers:

- An integer must have at least one digit
- An integer can be either positive or negative
- Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example \$x is an integer. The PHP var\_dump() function returns the data type and value:

## Example

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



## PHP Float

- A float (floating point number) is a number with a decimal point or a number in exponential form.
- In the following example \$x is a float. The PHP var\_dump() function returns the data type and value:

### Example

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

## PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

- \$x = true;  
\$y = false;
- Booleans are often used in conditional testing.



## PHP Array

- An array stores multiple values in one single variable.
- In the following example \$cars is an array. The PHP var\_dump() function returns the data type and value:

### Example

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php
```

```
$depts = array("Cse","civil","Mech");
```

```
var_dump($depts);
```

```
?>
```

```
</body>
```

```
</html>
```

---

```
array(3) { [0]=> string(3) "Cse" [1]=> string(5) "civil" [2]=> string(4) "Mech" }
```



## PHP Objects:

When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

- Let's assume we have a class named Car. A Car can have properties like model, color, etc. We can define variables like \$model, \$color, and so on, to hold the values of these properties.
- When the individual objects (Volvo, BMW, Toyota, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.
- If you create a `__construct()` function, PHP will automatically call this function when you create an object from a class.





## Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php
```

```
class Car {
```

```
    public $color;
```

```
    public $model;
```

```
    public function __construct($color, $model) {
```

```
        $this->color = $color;
```

```
        $this->model = $model;
```

```
    }
```

```
    public function message() {
```

```
        return "My car is a " . $this->color . " " . $this->model . "!";
```

```
    }
```

```
}
```



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

```
</body>
```

```
</html>
```

---

My car is a black Volvo!  
My car is a red Toyota!



## PHP NULL Value

- Null is a special data type which can have only one value: NULL.
- A variable of data type NULL is a variable that has no value assigned to it.

### Example:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php
```

```
$x = "Hello world!";
```

```
$x = null;
```

```
var_dump($x);
```

```
?>
```

```
</body>
```

```
</html>
```



## PHP Resource

- The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.
- A common example of using the resource data type is a database call.

## PHP String Functions

`strlen()` - Return the Length of a String

The PHP `strlen()` function returns the length of a string.

### Example

Return the length of the string "Hello world!":

```
<?php  
echo strlen("Hello world!"); // outputs 12  
?>
```



## **str\_word\_count() - Count Words in a String**

The PHP str\_word\_count() function counts the number of words in a string.

### Example

Count the number of word in the string "Hello world!":

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

## **strrev() - Reverse a String**

The PHP strrev() function reverses a string.

### Example

Reverse the string "Hello world!":

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



## strpos() - Search For a Text Within a String

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

### Example

Search for the text "world" in the string "Hello world!":

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

## str\_replace() - Replace Text Within a String

The PHP str\_replace() function replaces some characters with some other characters in a string.

### Example

Replace the text "world" with "Dolly":

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



# PHP Conditional Statements

In PHP we have the following conditional statements:

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

## PHP - 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
$t = date("H");

if ($t < "20") {
    echo "Have a good day!";
}
?>
```

## PHP - The if...else Statement

The if...else statement executes some code if a condition is true and another code if that condition is false.

### Syntax

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





### Example:

```
<?php
$t = date("H");


if ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

### PHP - The if...elseif...else Statement

The if...elseif...else statement executes different codes for more than two conditions.

#### Syntax

```
if (condition) {
    code to be executed if this condition is true;
} elseif (condition) {
    code to be executed if first condition is false and this condition is true;
} else {
    code to be executed if all conditions are false;
}
```



```
<?php
$t = date("H");

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

---

The hour (of the server) is 10, and will give the following message:

Have a good day!



# The PHP switch Statement

Use the switch statement to **select one of many blocks of code to be executed.**

## Syntax

```
switch (n) {  
    case label1:  
        code to be executed if n=label1;  
        break;  
    case label2:  
        code to be executed if n=label2;  
        break;  
    case label3:  
        code to be executed if n=label3;  
        break;  
    ...  
    default:  
        code to be executed if n is different from all labels;  
}
```



```
<?php
$favcolor = "red";

switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}
?>
```



# PHP 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 PHP while Loop

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

### Syntax

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



## Example:

The example below displays the numbers from 1 to 5:

```
<?php
$x = 1;

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

## Example Explained

- **\$x = 1;** - Initialize the loop counter (\$x), and set the start value to 1
- **\$x <= 5** - Continue the loop as long as \$x is less than or equal to 5
- **\$x++;** - Increase the loop counter value by 1 for each iteration



## The PHP do...while Loop

- The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

### Syntax

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

### Example:

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



## The PHP for Loop

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

### Syntax

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

### Parameters:

***init counter***: Initialize the loop counter value

***test counter***: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.

***increment counter***: Increases the loop counter value

### Example:

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





## The PHP foreach Loop

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

### Syntax

```
foreach ($array as $value) {  
    code to be executed;  
}
```

For every loop iteration, the value of the current array element is assigned to \$value and the array pointer is moved by one, until it reaches the last array element.

### Example

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



# PHP GET and POST

There are two ways the browser client can send information to the web server.


- The GET Method
- The POST Method
- Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

**name1=value1&name2=value2&name3=value3**

## The GET Method

- The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character.

**http://www.test.com/index.htm?name1=value1&name2=value2**



- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send upto 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY\_STRING environment variable.
- The PHP provides \$\_GET associative array to access all the sent information using GET method.



## Example:

```
<?php
    if( $_GET["name"] || $_GET["age"] ) {
        echo "Welcome ". $_GET['name']. "<br />";
        echo "You are ". $_GET['age']. " years old.";

        exit();
    }
?>

<html>
    <body>

        <form action = "<?php $_PHP_SELF ?>" method = "GET">
            Name: <input type = "text" name = "name" />
            Age: <input type = "text" name = "age" />
            <input type = "submit" />
        </form>

    </body>
</html>
```

It will produce the following result -

A screenshot of a web browser displaying a form. The form is enclosed in a light gray border and contains three elements: a label 'Name:' followed by a text input field, a label 'Age:' followed by another text input field, and a 'Submit' button with a gray gradient and rounded corners. The input fields are empty.

Name:  Age:



## The POST Method

- The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY\_STRING.
- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **\$\_POST** associative array to access all the sent information using POST method.

### Example:

```
<?php
if( $_POST["name"] || $_POST["age"] ) {
    if(preg_match("/[^A-Za-z'-]/",$_POST['name'])) {
        die ("invalid name and name should be alpha");
    }
}
```



```
echo "Welcome ". $_POST['name']. "<br />";
    echo "You are ". $_POST['age']. " years old.";

    exit();
}
?>
<html>
    <body>

        <form action = "<?php $_PHP_SELF ?>" method = "POST">
            Name: <input type = "text" name = "name" />
            Age: <input type = "text" name = "age" />
            <input type = "submit" />
        </form>

    </body>
</html>
```



## The \$\_REQUEST variable

- The PHP \$\_REQUEST variable contains the contents of both \$\_GET, \$\_POST, and \$\_COOKIE
- The PHP \$\_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

### Example:

```
<?php
```

```
    if( $_REQUEST["name"] || $_REQUEST["age"] ) {  
        echo "Welcome ". $_REQUEST['name']. "<br />";  
        echo "You are ". $_REQUEST['age']. " years old.";  
        exit();  
    }
```

```
?>
```

```
<html>
```

```
<body>
```

```
<form action = "<?php $_PHP_SELF ?>" method = "POST">
```



```
Name: <input type = "text" name = "name" />  
    Age: <input type = "text" name = "age" />  
    <input type = "submit" />  
</form>  
</body>  
</html>
```





## PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

### Example

```
<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>
```

The output could be something like this:

```
Welcome John  
Your email address is john.doe@example.com
```

[https://tryphp.w3schools.com/showphp.php?filename=demo\\_form\\_post](https://tryphp.w3schools.com/showphp.php?filename=demo_form_post)



## PHP - File Inclusion

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to included one PHP file into another PHP file.

- The include() Function
- The require() Function

### The include() Function

- The include() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **include()** function generates a warning but the script will continue execution.

### Syntax

- `include 'filename';`

or

`require 'filename';`



## Example

Assume we have a standard footer file called "footer.php", that looks like this:

- ```
<?php
    echo "<p>Copyright &copy; 1999-" . date("Y") . " Rgukt.com</p>";
?>
```

To include the footer file in a page, use the include statement:

```
<html>
```

```
<body>
```

```
<h1>Welcome to my home page!</h1>
```

```
<p>Some text.</p>
```

```
<p>Some more text.</p>
```

```
<?php include 'footer.php';?>
```

```
</body>
```

```
</html>
```



## Example

Assume we have a standard menu file called "menu.php":

- `<?php`  
    `echo '<a href="/default.asp">Home</a> -`  
    `<a href="/html/default.asp">HTML Tutorial</a> -`  
    `<a href="/css/default.asp">CSS Tutorial</a> -`  
    `<a href="/js/default.asp">JavaScript Tutorial</a> -`  
    `<a href="default.asp">PHP Tutorial</a>';`  
    `?>`

`<html>`

`<body>`

`<div class="menu">`

`<?php include 'menu.php';?>`

`</div>`

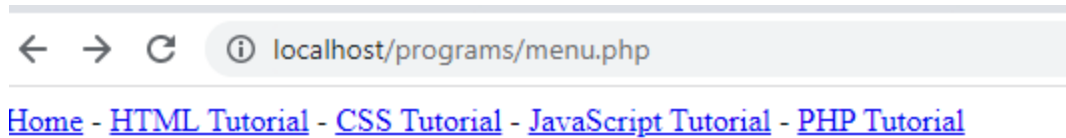
`<h1>Welcome to my home page!</h1>`

`<p>Some text.</p>`

`<p>Some more text.</p>`

`</body>`

`</html>`



## **Example**

Assume we have a file called "cars.php", with some variables defined:

```
<?php  
$color='red';  
$car='BMW';  
?>
```

Then, if we include the "vars.php" file, the variables can be used in the calling file:

## **Example**

```
<html>  
<body>  
  
<h1>Welcome to my home page!</h1>  
<?php include 'vars.php';  
echo "I have a $color $car.";  
?>  
  
</body>  
</html>
```



**PHP require() Function**: The **require()** function in PHP is basically used to include the contents/code/data of one PHP file to another PHP file. During this process if there are any kind of errors then this **require()** function will pop up a warning along with a fatal error and it will immediately stop the execution of the script.

## **PHP include vs. require**

- The require statement is also used to include a file into the PHP code.
- However, there is one big difference between include and require; when a file is included with the include statement and PHP cannot find it, the script will continue to execute:

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

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

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

**Welcome to my home page!**

I have a .

Using the require statement, the echo statement will not be executed because the script execution dies after the require statement returned a fatal error:

```
<html>
```

```
<body>
```

```
<h1>Welcome to my home page!</h1>
```

```
<?php require 'noFileExists.php';
```

```
echo "I have a $color $car.";
```

```
?>
```

```
</body>
```

```
</html>
```

---

**Welcome to my home page!**





## PHP File Handling

PHP has several functions for creating, reading, uploading, and editing files.

### PHP readfile() Function

- The readfile() function reads a file and writes it to the output buffer.
- Assume we have a text file called "**webdictionary.txt**", stored on the server, that looks like this:
- AJAX = Asynchronous JavaScript and XML  
CSS = Cascading Style Sheets  
HTML = Hyper Text Markup Language  
PHP = PHP Hypertext Preprocessor  
SQL = Structured Query Language  
SVG = Scalable Vector Graphics  
XML = EXtensible Markup Language



- The PHP code to read the file and write it to the output buffer is as follows (the `readfile()` function returns the number of bytes read on success):

### Example

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

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

### 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.
- Files modes can be specified as one of the six options in this table.

| Sr.No | Mode & Purpose   |
|-------|--|
| 1     | <b>r</b><br>Opens the file for reading only.<br>Places the file pointer at the beginning of the file.  |
| 2     | <b>r+</b><br>Opens the file for reading and writing.<br>Places the file pointer at the beginning of the file.  |
| 3     | <b>w</b><br>Opens the file for writing only.<br>Places the file pointer at the beginning of the file.<br>and truncates the file to zero length. If files does not exist then it attempts to create a file. |



|   |   |
|---|---|
| 4 | <b>w+</b><br>Opens the file for reading and writing only.<br>Places the file pointer at the beginning of the file.<br>and truncates the file to zero length. If files does not exist then it attempts to create a file. |
| 5 | <b>a</b><br>Opens the file for writing only.<br>Places the file pointer at the end of the file.<br>If files does not exist then it attempts to create a file.   |
| 6 | <b>a+</b><br>Opens the file for reading and writing only.<br>Places the file pointer at the end of the file.<br>If files does not exist then it attempts to create a file.  |



## Reading a file

- Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.
- The files length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes.

So here are the steps required to read a file with PHP.

- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.
- Read the file's content using **fread()** function.
- Close the file with **fclose()** function.



```
<html>
<head>
  <title>Reading a file using PHP</title>
</head>
<body>
  <?php
    $filename = "tmp.txt";
    $file = fopen( $filename, "r" );
    if( $file == false ) {
      echo ( "Error in opening file" );
      exit();
    }
    $filesize = filesize( $filename );
    $filetext = fread( $file, $filesize );
    fclose( $file );
    echo ( "File size : $filesize bytes" );
    echo ( "<pre>$filetext</pre>" ); </body></html>
```



---

File size : 278 bytes

The PHP Hypertext Preprocessor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases.

PHP is basically used for developing web based software applications. This tutorial helps you to build your base with PHP.

---



## Writing a file

- A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written

```
<?php
$filename = "/home/user/guest/newfile.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
echo ( "Error in opening new file" );
exit();
}
fwrite( $file, "This is a simple test\n" );
fclose( $file );?>
```





```
<html>
<head>
<title>Writing a file using PHP</title>
</head>
<body>
<?php
    $filename = "newfile.txt";
    $file = fopen( $filename, "r" );

    if( $file == false ) {
        echo ( "Error in opening file" );
        exit();
    }

    $filesize = filesize( $filename );
    $filetext = fread( $file, $filesize );

    fclose( $file );

    echo ( "File size : $filesize bytes" );
    echo ( "$filetext" );
    echo("file name: $filename");
?>
</body></html>
```

It will produce the following result -

File size : 23 bytes

This is a simple test

file name: newfile.txt



# Parsing directories

## PHP scandir() Function

- The **scandir()** function in PHP is an inbuilt function that is used to return an array of files and directories of the specified directory.
- The **scandir()** function lists the files and directories which are present inside a specified path.
- The directory, stream behavior, and **sorting\_order** of the files and directories are passed as a parameter to the **scandir()** function and it returns an array of filenames on success, or false on failure.

### Syntax:

**scandir(directory, sorting\_order, context);**

**Parameters:** The **scandir()** function in PHP accepts 3 parameters that are listed below:

- **directory:** It is a mandatory parameter that specifies the path.



- **sorting\_order:** It is an optional parameter that specifies the sorting order. Alphabetically ascending order (0) is the default sort order. It can be set to `SCANDIR_SORT_DESCENDING` or 1 to sort in alphabetically descending order, or `SCANDIR_SORT_NONE` to return the result unsorted.
- **context:** It is an optional parameter that specifies the behavior of the stream.

**Return Value:** It returns an array of filenames on success, or false on failure.

### **Errors And Exceptions:**

- The `scandir()` function throws an error of level `E_WARNING` if the directory specified is not a directory.
- Doing a recursive `scandir` will on a directory that has many files will likely either slow down your application or cause a high rise in RAM consumption due to the large size of the generated array.



```
<?php
$dir = "/images/";

// Sort in ascending order - this is default
$a = scandir($dir);

// Sort in descending order
$b = scandir($dir,1);

print_r($a);
print_r($b);
?>
```

```
Array
(
    [0] => .
    [1] => ..
    [2] => cat.gif
    [3] => dog.gif
    [4] => horse.gif
    [5] => myimages
)
Array
(
    [0] => myimages
    [1] => horse.gif
    [2] => dog.gif
    [3] => cat.gif
    [4] => ..
    [5] => .
)
```



## Create The HTML Form

- Next, create an HTML form that allow users to choose the image file they want to upload:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

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

Select image to upload:

```
<input type="file" name="fileToUpload" id="fileToUpload">
```

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

```
</form>
```

```
</body>
```

```
</html>
```



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 form above sends data to a file called **"upload.php"**, which we will create next.

## Create The Upload File PHP Script

The "upload.php" file contains the code for uploading a file:



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



## PHP script explained:

- `$target_dir = "uploads/"` - specifies the directory where the file is going to be placed
- `$target_file` specifies the path of the file to be uploaded
- `$uploadOk=1` is not used yet (will be used later)
- `$imageFileType` holds the file extension of the file (in lower case)
- Next, check if the image file is an actual image or a fake image

You will need to create a new directory called "uploads" in the directory where "upload.php" file resides. The uploaded files will be saved there.

### Check if File Already Exists

First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and `$uploadOk` is set to 0:

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





## Limit File Size

- The file input field in our HTML form above is named "fileToUpload".
- Now, we want to check the size of the file. If the file is larger than 500KB, an error message is displayed, and \$uploadOk is set to 0:
- ```
// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
}
```

## Limit File Type

- The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting \$uploadOk to 0:
- ```
// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png" &&
$imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";
    $uploadOk = 0;
}
```



## Complete Upload File PHP Script

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType
= strtolower(pathinfo($target_file,PATHINFO_EXTENSION));

// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
    if($check !== false) {
        echo "File is an image - " . $check["mime"] . ".";
        $uploadOk = 1;
    } else {
        echo "File is not an image.";
        $uploadOk = 0;
    }
}
```



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

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

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



```
// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
    echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
    if (move_uploaded_file($_FILES["fileToUpload"]["tmp_name"],
    $target_file)) {
        echo "The file ". htmlspecialchars(
basename( $_FILES["fileToUpload"]["name"])). " has been uploaded.";
    } else {
        echo "Sorry, there was an error uploading your file.";
    }
}
?>
```



## File Download

Image-gallery.php

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<title>Simple Image Gallery</title>
```

```
<style type="text/css">
```

```
    .img-box{  
        display: inline-block;  
        text-align: center;  
        margin: 0 15px;  
    }
```

```
</style>
```

```
</head>
```

```
<body>
```



```
<?php
// Array containing sample image file names
    $images=array("kites.jpg","balloons.jpg");
// Loop through array to create image gallery
    foreach($images as $image){
        echo '<div class="img-box">';
            echo '';
            echo '<p><a href="download.php?file=' .
urlencode($image) . '">Download</a></p>';
        echo '<div>';
    }
?>
</body>
</html>
```



## Download.php

```
<?php
```

```
if(isset($_REQUEST["file"])){
```

```
// Get parameters
```

```
    $file = urldecode($_REQUEST["file"]); //Decode URL-encoded string
```

```
    $filepath = "images/" . $file;
```

```
// Process download
```

```
    if(file_exists($filepath)) {
```

```
        header('Content-Description: File Transfer');
```

```
        header('Content-Type: application/octet-stream');
```

```
        header('Content-Disposition: File Transfer');
```

```
        header('Expires: 0');
```

```
        header('Cache-Control: must-revalidate');
```

```
        header('Pragma: public');
```

```
        header('content-Length: ' . filesize($filepath));
```

```
        flush(); //flush system output buffer
```

```
        readfile($filepath);
```

```
        exit;
```

```
    }
```

```
}
```

```
?>
```





[Download](#)



[Download](#)





# PHP Sessions

Session variables hold information about one single user, and are available to all pages in

## Start a PHP Session

- A session is started with the `session_start()` function.
- Session variables are set with the PHP global variable: `$_SESSION`.
- Now, let's create a new page called "demo\_session1.php". In this page, we start a new PHP session and set some session variables:  
one application.

```
<?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>
```

## Get PHP Session Variable Values

- Next, we create another page called "demo\_session2.php". And from this page it access the session information we set on the first page ("demo\_session1.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>
```

A screenshot of a web browser's address bar. It contains navigation icons (back, forward, refresh) and an information icon. The address bar text is "localhost/programs/s1.php".

localhost/programs/s1.php

Favorite color is green.

Favorite animal is cat.



## 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>
```



## Destroy a PHP Session

To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`:

### Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// remove all session variables
session_unset();
// destroy the session
session_destroy();
?>
</body>
</html>
```



# JSON

- JSON stands for JavaScript Object Notation, and is a syntax for storing and exchanging data.
- Since the JSON format is a text-based format, it can easily be sent to and from a server, and used as a data format by any programming language.

## PHP and JSON

PHP has some built-in functions to handle JSON.

First, we will look at the following two functions:

- `json_encode()`
- `json_decode()`

## PHP - json\_encode()

- The `json_encode()` function is used to encode a value to JSON format.

### Example

- This example shows how to encode an associative array into a JSON object:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php
```

```
$age = array("Peter"=>35, "Ben"=>37, "Joe"=>43);
```

```
echo json_encode($age);
```

```
?>
```

```
</body>
```

```
</html>
```

## Example

- This example shows how to encode an indexed array into a JSON array:



```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php
```

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

```
echo json_encode($cars);
```

```
?>
```

```
</body>
```

```
</html>
```

## **PHP - json\_decode()**

- The `json_decode()` function is used to decode a JSON object into a PHP object or an associative array.

### **Example**

- This example decodes JSON data into a PHP object:





```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<?php
```

```
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';
```

```
var_dump(json_decode($jsonobj));
```

```
?>
```

```
</body>
```

```
</html>
```

```
object(stdClass)#1 (3) { ["Peter"]=> int(35) ["Ben"]=> int(37) ["Joe"]=> int(43) }
```



