PHP Fundamentals

Types of Data / Data Types:

- Integer
- Double/Float
- String
- Array
- Boolean (True/False only)
- Null
- Object
- Resource

Operators:

PHP Operators:

Operators are used to perform operations on variables and values.
PHP divides the operators in the following groups:

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

Arithmetic Operators:

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result	Try it
+	Addition	$x + $y	Sum of $x and $y	Try it »
-	Subtraction	$x - $y	Difference of $x and $y	Try it »
*	Multiplication	$x * $y	Product of $x and $y	Try it »

Operator	Name	Example	Result	Try it
/	Division	$x / $y	Quotient of $x and $y	Try it »
%	Modulus	$x % $y	Remainder of $x divided by $y	Try it »
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power	Try it »

Assignment Operators:

The PHP assignment operators are used with numeric values to write a value to a variable. The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description	Try it
x = y	x = y	The left operand gets set to the value of the expression on the right	Try it »
x += y	x = x + y	Addition	Try it »
x -= y	x = x - y	Subtraction	Try it »
x *= y	x = x * y	Multiplication	Try it »
x /= y	x = x / y	Division	Try it »
x %= y	x = x % y	Modulus	Try it »

Comparison Operators:

The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result	Try it
==	Equal	$x == $y	Returns true if $x is equal to $y	Try it »
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type	Try it »
!=	Not equal	$x != $y	Returns true if $x is not equal to $y	Try it »
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y	Try it »

Operator	Name	Example	Result	Try it
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type	Try it »
>	Greater than	$x > $y	Returns true if $x is greater than $y	Try it »
<	Less than	$x < $y	Returns true if $x is less than $y	Try it »
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y	Try it »
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y	Try it »
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7.	Try it »

Increment / Decrement Operators:

The PHP increment operators are used to increment a variable's value.
The PHP decrement operators are used to decrement a variable's value.

Operator	Same as...	Description	Try it
++$x	Pre-increment	Increments $x by one, then returns $x	Try it »
$x++	Post-increment	Returns $x, then increments $x by one	Try it »
--$x	Pre-decrement	Decrements $x by one, then returns $x	Try it »
$x--	Post-decrement	Returns $x, then decrements $x by one	Try it »

Logical Operators:

The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result	Try it
and	And	$x and $y	True if both $x and $y are true	Try it »
or	Or	$x or $y	True if either $x or $y is true	Try it »
xor	Xor	$x xor $y	True if either $x or $y is true, but not both	Try it »
&&	And	$x && $y	True if both $x and $y are true	Try it »
|	Or	$x | $y	True if either $x or $y is true	Try it »
!	Not	!$x	True if $x is not true	Try it »

String Operators:

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result	Try it
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2	Try it »
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1	Try it »

Array Operators:

The PHP array operators are used to compare arrays.

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

Conditional Assignment Operators:

The PHP conditional assignment operators are used to set a value depending on conditions:

Operator	Name	Example	Result	Try it
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE	Try it »
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x	Try it »

Operator	Name	Example	Result	Try it
			is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7	

Variables, Constants and Comments:

Variables:

```php
<?php

// variable declaring styles (camelCase, TitleCase, snake_case etc.)
$camelCase = "John"; // Used most of the times for php
$TitleCase = "John";
$snake_case = "John";

// declared a variable
$word = "age";
$age = 16;
// will show output = 16, because $$word means -> $$word -> $(age) -> $age -> 16

echo $$word;



Output:

16
```

Constants:

```php
<?php

// constant variable declare: use define() function -> variable name should be written in UPPERCASE -> define( "VARIABLENAME", "value" );
define( "PI", 3.14 );

// while printing -> don't use any $ keyword in front of the variable name, just use the variable name directly
echo "Value of PI " . PI;
```

```
echo "\n";

//another method to run with constant() function
echo constant( "PI" );
// keep that in mind, variables are mutable, can be overridden, but constant variables cannot be overridden or immutable. So, once declared means cannot be changed.



Output:

Value of PI 3.14
3.14
```

Comments:

```
// -> single line comment -> mostly used than #
# -> another single line comment


/*
multi line comment
multi line comment
multi line comment
multi line comment
*/
```

Printing methods:

- echo
- var_dump() -> additional info about the variable with output
- printf()

With echo:

```
<?php

$name = "John";
$age = 22;
$question = "How are you?";


// a method to show output
```

```php
echo "Hello " . $name; // with dot operator
echo "\n"; // newline

// another method to show output
echo "Hello {$name}, your age is {$age} \n"; // in a single string
echo "Hello {$name}, {$question} \n";
```

```
Output:

Hello John
Hello John, your age is 22.
Hello John, How are you?
```

With var_dump():

```php
<?php

$name = "John";
$age = 16;
$result = True;

// with output, if we want to know additional info about the variable, we
use var_dump() ->
var_dump( $name ); // it will show the type (string(with length)) and value
var_dump( $age ); // it will show the type (int(value))
var_dump( $result ); // it will show the type (bool(value)) -> value is true
or false
```

```
Output:

string(4) "John"
int(16)
bool(true)
```

With printf() -> we can use %s, %d etc:

```php
<?php

$name = "Earth";
printf( "We're living on %s \n", strtoupper( $name ) ); // -> we can
directly use a function like strtoupper() to manipulate the variable with
printf()

$firstName = "Isaac";
```

```
$lastName = "Newton";

printf( "My name is %s %s \n", $firstName, $lastName );
printf( "My %s Name is %s %s \n", "Full", $firstName, $lastName );


// method to show how much decimal number we want after the decimal point
$float = 2.5565654645;

printf( "The number is %f \n", $float );
printf( "The number is %.2f \n", $float ); // -> .2f -> think like this, how
much decimal number we want to show after the decimal point


Output:

We're living on EARTH
My name is Isaac Newton
My Full Name is Isaac Newton
The number is 2.556565
The number is 2.56
```

With sprintf():

```
<?php

$planet1 = "Mercury";
$planet2 = "Neptune";

// to keep the print statement in a variable, we use sprintf()
$output = sprintf( "The smallest planet is %s and the largest planet is %s!
\n", $planet1, $planet2 );
// now just simply echo that variable
echo strtoupper( $output );



Output:

THE SMALLEST PLANET IS MERCURY AND THE LARGEST PLANET IS NEPTUNE!
```

All printing methods in a single place:

```
<?php

$planet1 = "Mercury";
$planet2 = "Neptune";
```

```php
echo "The smallest planet is {$planet1} and the largest planet is
{$planet2}! \n";
echo "The smallest planet is " . $planet1 . " and the largest planet is " .
$planet2 . "!" . "\n";
printf( "The smallest planet is %s and the largest planet is %s! \n",
$planet1, $planet2 );
var_dump( $planet1, $planet2 );

$output = sprintf( "The smallest planet is %s and the largest planet is %s!
\n", $planet1, $planet2 );
echo $output;
```

```
Output:

The smallest planet is Mercury and the largest planet is Neptune!
The smallest planet is Mercury and the largest planet is Neptune!
The smallest planet is Mercury and the largest planet is Neptune!
string(7) "Mercury"
string(7) "Neptune"
The smallest planet is Mercury and the largest planet is Neptune!
```

Operand and Operators, Summation, Subtraction, Multiplication, Division, Remainder:

Operand and Operators:

```php
<?php

// Operand and Operators
$number = 12 + 3;
// Variable = Operand(12) + Operator(+) + Operand(3)
```

Summation(+):

```php
<?php

$number = 10;

$number = $number + 5;
echo $number;
echo "\n";

$number += 5;
```

```
echo $number;


Output:

15
20
```

Summation with (++):

```
<?php

$number = 4;
$number++; // will increase only 1

echo $number;


Output:

5
```

Subtraction(-):

```
<?php

$number = 10;

$number = $number - 5;
echo $number;
echo "\n";

$number -= 5;
echo $number;


Output:

5
0
```

Subtraction with (--):

```
<?php

$number = 5;
$number--; // will decrease only 1
```

```
echo $number;


Output:

4
```

Multiplication(*):

```
<?php

$number = 10;

$number = $number * 5;
echo $number;
echo "\n";

$number *= 5;
echo $number;



Output:

50
250
```

Division(/):

```
<?php

$number = 12;

$number = $number / 4;
echo $number;
echo "\n";

$number /= 3;
echo $number;



Output:

3
1
```

Remainder(%):

```php
<?php

$number1 = 12;
$number2 = 12;

$number1 = $number1 % 5;
echo $number1;
echo "\n";


$number2 %= 3;
echo $number2;



Output:

2
0
```

Number Conversion Systems:

- How to declare a number in a variable in decimal, octal and hexadecimal
- How to convert in each other

Declaring in variables:

```php
<?php

// to declare a number in decimal, simply write the number
$decimal = 12;
// to declare a number in octal, put one zero(0) before the number, with no space
$octal = 012;
// to declare a number in hexadecimal, put one zero(0) with x before the number, with no space
$hexadecimal = 0x12;
// to declare a number in binary, put one zero(0) with b before the number, with no space
$binary = 0b1100;
```

Conversions:

```php
<?php

$decimal = 12;
$octal = 012;
$hexadecimal = 0x12;
$binary = 0b1100;

// conversions of ($decimal, $octal, $hexadecimal, $binary) to decimals
printf( "The number is %d, %d, %d and %d \n", $decimal, $octal,
$hexadecimal, $binary );

// conversions to binaries
printf( "The binary of %d is %b \n", $decimal, $decimal ); // -> with %d, it
will first convert the decimal to decimal and then with %b, convert it to
binary
printf( "The binary of %d is %b \n", $octal, $octal ); // -> with %d, it
will first convert the octal to decimal and then with %b, convert it to
binary
printf( "The binary of %d is %b \n", $hexadecimal, $hexadecimal ); // ->
with %d, it will first convert the hexadecimal to decimal and then with %b,
convert it to binary
printf( "The binary of %d is %b \n", $binary, $binary ); // -> with %d, it
will first convert the binary to decimal and then with %b, convert it to
binary

// conversions to hexadecimals (%x -> for lowercase / %X -> for uppercase)
printf( "The hexadecimal of %d is %X \n", $decimal, $decimal ); // -> with
%d, it will first convert the decimal to decimal and then with %X/%x,
convert it to hexadecimal
printf( "The hexadecimal of %d is %X \n", $octal, $octal ); // -> with %d,
it will first convert the octal to decimal and then with %X/%x, convert it
to hexadecimal
printf( "The hexadecimal of %d is %X \n", $hexadecimal, $hexadecimal ); // -
> with %d, it will first convert the hexadecimal to decimal and then with
%X/%x, convert it to hexadecimal
printf( "The hexadecimal of %d is %X \n", $binary, $binary ); // -> with %d,
it will first convert the binary to decimal and then with %X/%x, convert it
to hexadecimal
printf( "The hexadecimal is %X \n", $decimal );

// conversions to octal
printf( "The octal of %d is %o \n", $decimal, $decimal ); // -> with %d, it
will first convert the decimal to decimal and then with %o, convert it to
octal
printf( "The octal of %d is %o \n", $octal, $octal ); // -> with %d, it will
first convert the octal to decimal and then with %o, convert it to octal
printf( "The octal of %d is %o \n", $hexadecimal, $hexadecimal ); // -> with
%d, it will first convert the hexadecimal to decimal and then with %o,
convert it to octal
```

```
printf( "The octal of %d is %o \n", $binary, $binary ); // -> with %d, it
will first convert the binary to decimal and then with %o, convert it to
octal
printf( "The octal is %o \n", $decimal );

// conversions of hexadecimal to octal
printf( "The hexadecimal = %x number is equivalent to octal = %o \n",
$hexadecimal, $hexadecimal );
```

Conditions (Logics):

```
<?php

/*
If statement ->
If (condition) {
code to be executed if condition is true
}else {
code to be executed if condition is false
}
 */

// Example - 1
$n = 12;

if ( $n % 2 == 0 ) {
    echo "{$n} is Even \n";
} else {
    echo "{$n} is Odd \n";
}


// Example - 2
if ( $n > 10 ) {
    echo "{$n} is greater than 10 \n";
} else {
    echo "{$n} is less than 10 \n";
}


// Example - 3 with else if
/*
If (condition) {
code to be executed if condition is true
}elseif (condition) {
code to be executed if condition is true
```

```
}elseif (condition) {
code to be executed if condition is true
}else {
code to be executed if all conditions are false
}
 */

$alam = 100;
$rahim = 200;

if ( $alam == $rahim ) {
    echo "Alam has equal amount of money as Rahim \n";
} elseif ( $alam > $rahim ) {
    echo "Alam has more amount of money than Rahim \n";
} elseif ( $alam < $rahim ) {
    echo "Alam has less amount of money than Rahim \n";
} elseif ( $alam >= $rahim ) {
    echo "Alam has more or equal amount of money than Rahim \n";
}



// Example - 4
$age = 14;

if ( $age >= 13 && $age <= 19 ) {
    echo "This person is a teenager \n";
} else {
    echo "This person is not a teenager \n";
}



// Example - 5
$food = "apple";

// for any static data, it's always a good practice to use that static data
first in the condition check, so that, unwillingly if the declared operator
is '=', will show error / won't work
if ( "tuna" == $food ) {
    echo "It has Vitamin D \n";
} else {
    echo "Not matching \n";
}



// Example - 6
$food = "tuna";
```

```
// for any static data, it's always a good practice to use that static data
first in the condition check, so that, unwillingly if the declared operator
is '=', will show error / won't work
if ( "tuna" == $food || "apple" == $food ) {
    echo "It's a food \n";
} else {
    echo "Not a food \n";
}



Output:

12 is Even
12 is greater than 10
Alam has less amount of money than Rahim
This person is a teenager
Not matching
It's a food
```

Readability of the code (if-else):

```php
<?php

$year = 2020;

/*
1. Check if year is divisible by 4
2. Check if year is divisible by 100
3. Check if year is divisible by 400
4. If all above conditions are true, then it's a leap year
5. Otherwise, it's not a leap year
6. Print the result
7. If the year is not divisible by 4, then it's not a leap year
8. If the year is not divisible by 100, then it's a leap year
9. If the year is not divisible by 400, then it's not a leap year
10. Otherwise, it's a leap year
11. Print the result
 */

// Though it works, but not that readable
if ( $year % 4 == 0 ) {
    if ( $year % 100 == 0 ) {
        if ( $year % 400 == 0 ) {
            echo "It's a leap year \n";
        } else {
            echo "It's not a leap year \n";
        }
```

```
    } else {
        echo "It's a leap year \n";
    }

} else {
    echo "It's not a leap year \n";
}



// Best way:
if ( $year % 4 == 0 && ( $year % 100 != 0 || $year % 400 == 0 ) ) {
    echo "It's a leap year \n";
} else {
    echo "It's not a leap year \n";
}



Output:

It's a leap year
It's a leap year
```

Ternary Operator (? :) -> (Make if-else code shorter and simpler):

```
<?php

// normal if-else
$number = 10;

if ( 10 == $number ) {
    echo "Ten \n";
} else {
    echo "A number \n";
}

// cleaning up with ternary operator
$numberInWord = ( 10 == $number ) ? "Ten" : "A number";
// variable = (condition) ? (take this if true) : (take this if false)
echo $numberInWord;
echo "\n";


// Another Example:
$n = 10;

if ( $n % 2 == 0 ) {
    echo "Even \n";
```

```php
} else {
    echo "Odd \n";
}

// cleaning up with ternary operator
$evenOrOdd = ( $n % 2 == 0 ) ? "Even" : "Odd";
echo $evenOrOdd;
echo "\n";


// Another Example:
$num = 10;

if ( $num == 12 ) {
    echo "Twelve \n";
} elseif ( $num == 10 ) {
    echo "Ten \n";
} else {
    echo "A number \n";
}

// cleaning up with ternary operator
$result = ( 12 == $num ) ? "Twelve" : ( ( 10 == $num ) ? "Ten" : "A number"
);
echo $result;


Output:

Ten
Ten
Even
Even
Ten
Ten
```

Functions:

PHP built-in functions:

Built in by default for direct use. A lot of built in functions are already created in PHP. To learn them visit :
https://www.php.net/manual/en/functions.internal.php
https://www.php.net/manual/en/indexes.functions.php

User-defined functions:

Functions declared/written by the programmer him or herself.

```php
<?php

// User-defined function
function sum() {

    $num1 = 10;
    $num2 = 10;

    $result = $num1 + $num2;

    echo $result;
}

// calling the function
sum();


Output:

20
```

Parameters and Arguments:

```php
<?php

/**
 * Parameter and Argument
 * after the function name, inside the parenthesis, we write the parameters
 * arguments are the values that are passed to the function while calling
the function inside the parenthesis, these values will be passed in the
order of the parameters
 */

/*
method:

function functionName(parameter, parameter, moreParametersAsManyAsWeWant) {
    //code
}
 */

function sum( $x, $y ) {

    $num1 = $x;
    $num2 = $y;
```

```
    $result = $num1 + $num2;
    echo $result;
}

sum( 10, 40 );
```

Default Parameters:

```
<?php

// $y = 40; declared by default, not passing as argument
function sum( $x, $y = 40 ) {

    $num1 = $x;
    $num2 = $y;

    $result = $num1 + $num2;

    echo $result;
}

sum( 20 );
```

Parameter Type-Hinting:

```
<?php

// to ensure the right type of argument passing in the function, we can
strictly declare the type(int, float, bool, string, array, null) of the
argument in the parenthesis after the function name. Passing any other type
of data as argument, will show error
function sum( int $x, int $y ) {

    $num1 = $x;
    $num2 = $y;

    $result = $num1 + $num2;

    echo $result;
}

sum( 30, 20 );
```

Parameter Multiple Type-Hinting:

```
<?php
```

```php
// multiple type hinting
function sum( int | float $x, int $y ) {

    $num1 = $x;
    $num2 = $y;

    $result = $num1 + $num2;

    echo $result;
}

sum( 30.5, 20 );
```

Parameter Null Type-Hinting:

```php
<?php

// to ensure that an argument is passed as NULL to the parameter and no
error occur, we use "?" before the type
function sum( ?string $age ) {

    echo $age;
}

sum( NULL );
```

Variadic Function:

If it happens that, we don't know about how many parameters we need to take, we can use Variadic Function which is declared by using 'spread operator(...)'. It will take as many arguments as we want in the parameters.

```php
<?php

function worldCountryList( ...$country ) {

    print_r( $country );
}

worldCountryList( "Bangladesh", "England", "USA", "Canada", "Iceland",
"India", "Nepal", "Sri Lanka" );


Output:

Array
(
```

```
    [0] => Bangladesh
    [1] => England
    [2] => USA
    [3] => Canada
    [4] => Iceland
    [5] => India
    [6] => Nepal
    [7] => Sri Lanka
)
```

Anonymous Function:

```php
<?php

// anonymous function, it will be executed automatically without being called particularly
( function () {
    echo "Hello World \n";
} )();
```

Return:

```php
<?php

function addTwoNumber( $num1, $num2 ) {

    $result = $num1 + $num2;

    return $result;
}

echo addTwoNumber( 10, 20 );
```

Return Type:

```php
<?php

// to make sure to return a specific type, we use ':type', we use ':void' if the function return nothing
// multiple return type / Union Return Type :
function addTwoNumber( $parameter ): int | float | String | bool | null {

    return $parameter;
}

echo addTwoNumber( "A" );
```

By Reference:

```php
<?php

$name = "John Doe";

function name( &$person ) {
    $person = "Jane Doe";
    $greet = "Hello {$person} \n";

    echo $greet;
}

name( $name );

echo $name;
// after using '&' before the parameter -> the changed value in the function
will also change the original variable, which is $name here


Output:

Hello Jane Doe
Jane Doe
```

Working with long function:

While working on long function, the codes in the function will be so many and it will reduce the readability of the codes. The best practice is to break the tasks of that function into pieces and make different functions with those and then call those in the main function. For example -
Let's say, we defined a function named completeOrder(). In the function, we have to work with - code for Database Connection, code for reducing product stock, code for changing in order status, code for calculating the total price, code for generating pdf invoice, and code for sending email. Now all of these small tasks can be defined in a single function, but these so many lines of codes will reduce readability of that function and will be harder to understand. So we will break these tasks in other functions and will call them in the functions according to our needs and orders.

```php
<?php

function completeOrder( $orderId ) {

    // code for Database Connection
    $databaseConnection = databaseConnection( $orderId );

    // code for reducing product stock
```

```php
    $reduceProductStock = reduceProductStock( $orderId );

    // code for changing in order status
    $changeOrderStatus = changeOrderStatus( $orderId );

    // code for calculating the total price
    $calculateTotalPrice = calculateTotalPrice( $orderId );

    // code for generating pdf invoice
    $generatePdfInvoice = generatePdfInvoice( $orderId );

    // code for sending email
    $sendEmail = sendEmail( $orderId );

    return true;
}



// functions of small tasks
function databaseConnection( $orderId ) {

    // code for Database Connection
}

function reduceProductStock( $orderId ) {

    // code for reducing product stock
}

function changeOrderStatus( $orderId ) {

    // code for changing in order status
}

function calculateTotalPrice( $orderId ) {

    // code for calculating the total price
}

function generatePdfInvoice( $orderId ) {

    // code for generating pdf invoice
}

function sendEmail( $orderId ) {

    // code for sending email
}
```

Variable Scope:

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has different variable scopes:

- local
- global
- super global
- static

Local:

```php
<?php

// $num1 and $num2 both variables are with same name in both functions below, but they have different values. Though with same name, they are different variables.Because they are declared inside of those different functions. So the values won't be overwritten. When a variable works in a single function, it is called local scope variable.
function sum1() {
    $num1 = 10;
    $num2 = 10;

    $result = $num1 + $num2;

    echo $result;
}

function sum2() {
    $num1 = 100;
    $num2 = 100;

    $result = $num1 + $num2;

    echo $result;
}

sum1();
sum2();
```

Global:

```php
<?php
```

```php
// Now instead of declaring differently inside the different functions,
let's say, we declared those outside of the functions and want to use them
inside the functions. To use them, we write 'global' keyword before the
variables, inside the functions. This is called Global Scope Variable.
$num1 = 10;
$num2 = 10;

function sum1() {

    global $num1, $num2;

    $result = $num1 + $num2;

    echo $result;
}

function sum2() {

    global $num1, $num2;

    $result = $num1 + $num2;

    echo $result;
}

sum1();
sum2();
```

Super Global:

Super global variables in php are -

- COOKIES
- SESSION
- GET
- POST
- PUT
- PATCH
- DELETE
- HEAD
- OPTIONS

  etc.

Local scope and global scope works in a single php script but Super globals can be used in different scripts/php files, works in entire the application.

So let's say one Super Scope Variable is declared in a php file, and this variable can be used in the other php files/scripts also.

Static:

```
<?php

function total() {
    // without using 'static' keyword
    $count = 0;
    $count++;

    echo "Count : {$count} \n";
}

total();
total();
total();
total();
total();


Output:

Count : 1
Count : 1
Count : 1
Count : 1
Count : 1

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

<?php

function total() {
    // with using 'static' keyword
    static $count = 0;
    $count++;

    echo "Count : {$count} \n";
}

total();
total();
total();
total();
total();
```

```
Output:

Count : 1
Count : 2
Count : 3
Count : 4
Count : 5
```

In other scopes, the global or local scope variables, they are working with the given values, no change will be overwritten in the memory. But static will change the value, and keep that in the memory.

Loop:

For Loop:

```php
<?php

/*
method:
for ( start condition; end condition; increament/decreament ) {
execution of code
}
 */

for ( $i = 0; $i <= 10; $i++ ) {

    echo $i . "\n";
}


Output:

0
1
2
3
4
5
6
7
8
9
10
```

While Loop:

```php
<?php

// Start Condition
$i = 0;

// while (end condition)
while ( $i <= 10 ) {
    // code execution
    echo $i . "\n";
    // increament/decreament
    $i++;
}
```

Output:

```
0
1
2
3
4
5
6
7
8
9
10
```

Do-While Loop:

```php
<?php

// Start Condition
$i = 0;
/* do{
code execution
increament/decreament
}
while ( end condition );
 */
do {
    echo $i . "\n";
    $i++;
} while ( $i <= 10 );
```

```
Output:

0
1
2
3
4
5
6
7
8
9
10
```

Break:

```php
<?php


for ( $i = 0; $i <= 10; $i++ ) {
// after meeting the condition, break the loop, won't execute the rest of
the code
    if ( $i == 5 ) {
        break;
    }

    echo $i . "\n";
}


Output:

0
1
2
3
4
```

Continue:

```php
<?php

for ( $i = 0; $i <= 10; $i++ ) {
// after meeting the condition, it will be skipped at that point and jump to
the next iteration
    if ( $i % 2 == 0 ) {
        continue;
    }
```

```
    echo $i . "\n";
}


Output:

1
3
5
7
9
```

Array:

In PHP, an array is a variable that can store multiple values under a single name.

Array Declaration:

```
<?php

// two methods to create array in PHP
$fruits = [ "Apple", "Banana", "Orange" ];

$fruits = array( "Apple", "Banana", "Orange" );
```

Indexed Array:

```
<?php

// indexed array
$fruits = ["Apple", "Banana", "Orange"];
// will have index for each value, starting with 0

// to show the array output, echo won't work, we have to use print_r for
this:
print_r( $fruits );

// to print a specific index value: we can use echo and print_r both, and
inside [] -> we need to put the index
echo $fruits[0];
echo "\n";
print_r( $fruits[1] );
```

```
Output:

Array
(
    [0] => Apple
    [1] => Banana
    [2] => Orange
)
Apple
Banana
```

Looping over Indexed Array:

```php
<?php

// how to loop over on indexed array
$systems = ["android", "ios", "windows", "linux"];

/* we will use foreach for this.
method :
foreach ( 'name of the array variable' 'as keyword' 'variable to pick each
item in the array' ) {
code
}
 */
foreach ( $systems as $system ) {
    echo $system . PHP_EOL;
}


Output:

android
ios
windows
linux
```

Associative Array:

```php
<?php

// declaring associative array:
$billGates = [
    "firstName" => "Bill",
    "lastName"  => "Gates",
    "age"       => 62,
];
// it will have key and value pair. We call it associative array with Key
```

```
Value pair.
// here, firstName, lastName and age are the -> Keys
// Bill, Gates and 62 are -> Values
// all the data are representing Bill Gates
// multiple data when represent a particular scenario, we call that
associative array

// printing associative array
print_r( $billGates );
echo "\n";

// to get the specific value:
echo $billGates['firstName'];
echo "\n";
// we can use print_r also:
print_r( $billGates['lastName'] );


Output:

Array
(
    [firstName] => Bill
    [lastName] => Gates
    [age] => 68
)

Bill
Gates
```

Looping over Associative Array:

```
<?php

$billGates = [
    "firstName" => "Bill",
    "lastName"  => "Gates",
    "age"       => 68,
];

/* we will use foreach for this.
method :
foreach ( 'name of the array variable'    'as keyword'    '$key' => '$value'
) {
code
}
 */
foreach ( $billGates as $key => $value ) {
    echo "{$key} : {$value} \n";
```

```php
}

// to print the keys only:
foreach ( $billGates as $key => $value ) {
    echo "{$key}\n";
}

// to print the values only:
foreach ( $billGates as $key => $value ) {
    echo "{$value}\n";
}


Output:
firstName : Bill
lastName : Gates
age : 68
firstName
lastName
age
Bill
Gates
68
```

Multidimensional Indexed Array:

```php
<?php

// multidimensional indexed array -> array inside an array
$fruits1 = ['apple1', 'banana1', 'orange1'];
$fruits2 = ['apple2', 'banana2', 'orange2'];
$fruits3 = ['apple3', 'banana3', 'orange3'];

$allFruits = [$fruits1, $fruits2, $fruits3];

// to print the whole array:
print_r( $allFruits );

// to find an element or value:
echo $allFruits[1][0];
echo "\n";
echo $allFruits[2][2];
echo "\n";


Output:

Array
(
```

```
    [0] => Array
        (
            [0] => apple1
            [1] => banana1
            [2] => orange1
        )

    [1] => Array
        (
            [0] => apple2
            [1] => banana2
            [2] => orange2
        )

    [2] => Array
        (
            [0] => apple3
            [1] => banana3
            [2] => orange3
        )

)
apple2
orange3
```

Looping over Multidimensional Indexed Array:

```php
<?php

$fruits1 = ['apple1', 'banana1', 'orange1'];
$fruits2 = ['apple2', 'banana2', 'orange2'];
$fruits3 = ['apple3', 'banana3', 'orange3'];

$allFruits = [$fruits1, $fruits2, $fruits3];

// looping over multidimensional indexed array -> keep that in mind we need to loop 2 times:
foreach ( $allFruits as $parentItem ) {
    foreach ( $parentItem as $childItem ) {
        echo $childItem . PHP_EOL;
    }

}
```

```
Output:

apple1
banana1
```

```
orange1
apple2
banana2
orange2
apple3
banana3
orange3
```

Multidimensional Associative Array:

```php
<?php

$billGates1 = [
    "firstName1" => "Bill1",
    "lastName1"  => "Gates1",
    "age"        => 60,
];

$billGates2 = [
    "firstName2" => "Bill2",
    "lastName2"  => "Gates2",
    "age"        => 61,
];

$billGates3 = [
    "firstName3" => "Bill3",
    "lastName3"  => "Gates3",
    "age"        => 62,
];

$persons = [$billGates1, $billGates2, $billGates3];

// to print all the data:
print_r( $persons );

// to print a specific data:
echo $persons[0]['firstName1'];
echo "\n";
print_r( $persons[2]['lastName3'] );


Output:

Array
(
    [0] => Array
        (
            [firstName1] => Bill1
            [lastName1] => Gates1
```

```
            [age] => 60
        )

    [1] => Array
        (
            [firstName2] => Bill2
            [lastName2] => Gates2
            [age] => 61
        )

    [2] => Array
        (
            [firstName3] => Bill3
            [lastName3] => Gates3
            [age] => 62
        )

)
Bill1
Gates3
```

Looping over Multidimensional Associative Array:

```php
<?php

$billGates1 = [
    "firstName1" => "Bill1",
    "lastName1"  => "Gates1",
    "age"        => 60,
];


$billGates2 = [
    "firstName2" => "Bill2",
    "lastName2"  => "Gates2",
    "age"        => 61,
];

$billGates3 = [
    "firstName3" => "Bill3",
    "lastName3"  => "Gates3",
    "age"        => 62,
];

$persons = [$billGates1, $billGates2, $billGates3];

// looping on multidimensional associative array:
foreach ( $persons as $parentItem ) {
    foreach ( $parentItem as $key => $value ) {
```

```
        echo "{$key} : {$value} \n";
    }

}


Output:

firstName1 : Bill1
lastName1 : Gates1
age : 60
firstName2 : Bill2
lastName2 : Gates2
age : 61
firstName3 : Bill3
lastName3 : Gates3
age : 62
```

Different Built-in functions for Array:

array_values():

Works for associative arrays and show output as indexed array, we get the values for each key:

```
<?php

$billGates = [
    "firstName" => "Bill",
    "lastName"  => "Gates",
    "age"       => 68,
];


$values = array_values( $billGates );
print_r( $values );


Output:

Array
(
    [0] => Bill
    [1] => Gates
    [2] => 68
)
```

array_keys():

Works for associative arrays and show output as indexed array, we only get the keys:

```php
<?php

$billGates = [
    "firstName" => "Bill",
    "lastName"  => "Gates",
    "age"       => 68,
];


$values = array_keys( $billGates );
print_r( $values );


Output:

Array
(
    [0] => firstName
    [1] => lastName
    [2] => age
)
```

array_key_exists():

Works for associative arrays and show output as True/False -> 1/show nothing -> 1 - exists/show nothing - doesn't exist :

```php
<?php

$billGates = [
    "firstName" => "Bill",
    "lastName"  => "Gates",
    "age"       => 68,
];

// array_key_exists( key, arrayName )
$values = array_key_exists( "firstName", $billGates );
print_r( $values );


Output:

1
```

array_search():

```php
<?php

$billGates = [
    "firstName" => "Bill",
    "lastName"  => "Gates",
    "age"       => 68,
];

// array_key_exists( key, arrayName )
$values = array_search( "Gates", $billGates );
print_r( $values );
echo "\n";
echo "{$values} \n";


Output:

lastName
lastName
```

array_flip():

Will flip the array, key will be value, value will be key:

```php
<?php

$billGates = [
    "firstName" => "Bill",
    "lastName"  => "Gates",
    "age"       => 68,
];


// array_key_exists( key, arrayName )
$values = array_flip( $billGates );
print_r( $values );


Output:

Array
(
    [Bill] => firstName
    [Gates] => lastName
    [68] => age
)
```

count() / sizeof():

Will count the elements in the array:

```php
<?php

$fruits = ['apple', 'banana', 'orange'];

$count = count( $fruits );
$size = sizeof( $fruits );

echo $count;
echo "\n";
echo $size;


Output:

3
3
```

array_sum():

Will give the sum of the elements in the array:

```php
<?php

$numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15];

$sum = array_sum( $numbers );

echo $sum;


Output:

120
```

array_product():

Will give the product of the elements in the array:

```php
<?php


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

$sum = array_product( $numbers );

echo $sum;
```

```
Output:

120
```

array_merge():

Will merge two arrays:

```php
<?php

$numbers1 = [1, 2, 3, 4, 5];
$numbers2 = [6, 7, 8, 9, 10];

$merged = array_merge( $numbers1, $numbers2 );

print_r( $merged );
```

```
Output:

Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
    [4] => 5
    [5] => 6
    [6] => 7
    [7] => 8
    [8] => 9
    [9] => 10
)
```

array_replace():

Will replace the value with another:

```php
<?php

$systems = ["android", "ios"];
$replace = array_replace( $systems, ["windows", "linux"] );
print_r( $replace );

// another method:
$replacement = [0 => "windows", 1 => "linux"];
```

```
print_r( array_replace( $systems, $replacement ) );


Output:

Array
(
    [0] => windows
    [1] => linux
)

Array
(
    [0] => windows
    [1] => linux
)
```

sort():

Will sort the array:

```
<?php

$numbers = [21, 12, 23, 4, 55];
$sorted = sort( $numbers );

print_r( $numbers );


Output:

Array
(
    [0] => 4
    [1] => 12
    [2] => 21
    [3] => 23
    [4] => 55
)
```

array_unique():

Will remove duplicate values from the array:

```
<?php

$numbers = [21, 21, 12, 12, 23, 4, 4, 55];
$unique = array_unique( $numbers );
```

```
print_r( $unique );


Output:

Array
(
    [0] => 21
    [2] => 12
    [4] => 23
    [5] => 4
    [7] => 55
)
```

array_reverse():

Will reverse the array:

```
<?php

$numbers = [21, 21, 12, 12, 23, 4, 4, 55];

$reverse = array_reverse( $numbers );

print_r( $reverse );


Output:

Array
(
    [0] => 55
    [1] => 4
    [2] => 4
    [3] => 23
    [4] => 12
    [5] => 12
    [6] => 21
    [7] => 21
)
```

String Manipulation:

String Concatenation:

```php
<?php

$str1 = "Hello";
$str2 = "World";

echo $str1 . " " . $str2 . "!\n";

// best way to avoid these . keywords
echo "{$str1} {$str2}!\n";


Output:

Hello World!
Hello World!
```

String length (strlen()):

```php
<?php

$str1 = "Hello";

$length = strlen( $str1 );

echo $length . "\n";


Output:

5
```

Finding Positioning / offset (substr()):

To get a particular position's character in the string.

```php
<?php

$str1 = "Hello World";

// method: substr( string, starting index, length or how many characters we want from the starting index )
$position = substr( $str1, 0, 2 );
$position2 = substr( $str1, 4, 3 );

echo $position1;
echo "\n";
echo $position2;
```

```
Output:

He
o W
```

Reversing String (strrev()):

```php
<?php


$str = "Hello World";

// if we reverse the string manually with for loop:
for ( $i = strlen( $str ) - 1; $i >= 0; $i-- ) {
    echo $str[$i];
}


Output:

dlroW olleH
```

Instead of doing this, we can simply do this with strrev() function:

```php
<?php


$str = "Hello World";

$reversed = strrev( $str );
echo $reversed;


Output:

dlroW olleH
```

String Repeat (str_repeat()):

```php
<?php

$string = "***-";

$repeat = "";
```

```
for ( $i = 0; $i < 5; $i++ ) {
    $repeat = $repeat . $string;
}

echo $repeat . "\n";


Output:

***-***-***-***-***-
```

Instead of doing this manually, we can use str_repeat():

```
<?php

$string = "***-";

// repeating the string
echo str_repeat( $string, 5 );


Output:

***-***-***-***-***-
```

String Shuffling (str_shuffle()) :

```
<?php


$string = "Hello World";

echo str_shuffle( $string );


Output:

H llodleoWr
```

Converting String to lowercased/uppercased (strtolower() & strtoupper()):

```
<?php

$string = "Hello World";

echo strtolower( $string );
```

```
echo "\n";
echo strtoupper( $string );


Output:

hello world
HELLO WORLD
```

Converting a String's all first letter uppercased (ucwords()):

```
<?php

$string = "hello world";

echo ucwords( $string );
echo "\n";


Output:

Hello World
```

To cut invisible/unnecessary characters (by default spaces) from both side of the string (trim()/ltrim()/rtrim()):

```
<?php

$string = "     Hello World     ";

// to remove spaces from the beginning
echo ltrim( $string ); // Hello World     Hello World
// to remove spaces from the end
echo rtrim( $string ); //     Hello World
// to remove spaces from the beginning and end
echo trim( $string ); // Hello World


Output:

Hello World
     Hello World
Hello World
```

```
<?php
```

```php
$string = "     Hello World//%^&@#@@!#!@# ??    ";

// we can remove any unnecessary elements from the string also
echo trim( $string, " /%^&@#!?" ); // space, /, %, &, @, #, !, #, ? will be removed


Output:

Hello World
```

To explode the words of the string and get the output as array (explode()):

```php
<?php

$string = "Hello World";


// To explode the words in the string and get the output as array:
$exploded = explode( " ", $string );
// Here wherever there is a space, will be splitted

print_r( $exploded );


Output:

Array
(
    [0] => Hello
    [1] => World
)
```

Same as explode() -> To explode all characters of the string and get the output as array (str_split()):

```php
<?php

$string = "Hello World";
print_r( str_split( $string ) );

// can even pass how many characters we want to take
print_r( str_split( $string, 5 ) );


Output:
```

```
Array
(
    [0] => H
    [1] => e
    [2] => l
    [3] => l
    [4] => o
    [5] =>
    [6] => W
    [7] => o
    [8] => r
    [9] => l
    [10] => d

)
Array
(
    [0] => Hello
    [1] =>  Worl
    [2] => d

)
```

```php
<?php

$article = "HellO world, thiS Is a Sample Article. Did You Notice the mixEd caSe?";
$article = strtolower( $article );

$string1 = "hello world. how are you";

$parts = explode( ".", $article );
print_r( $parts );

$fixed = "";

for ( $i = 0; $i < count( $parts ); $i++ ) {
    $parts[$i] = ucfirst( trim( $parts[$i] ) );
}

print_r( $parts );

$fixed = implode( ". ", $parts );
echo $fixed;
```

Output:

```
Array
(
    [0] => hello world, this is a sample article
    [1] =>  did you notice the mixed case
)

Array
(
    [0] => Hello world, this is a sample article
    [1] => Did you notice the mixed case
)
Hello world, this is a sample article. Did you notice the mixed case?
```

To join the exploded words again / join the array values in a string (join()/implode()):

```php
<?php

$string = "Hello World";

// To explode the words in the string and get the output as array, we call this as delimeter also:
$exploded = explode( " ", $string );
// Here wherever there is a space, it will be splitted


// to join the words in the array: join() / implode()
$joined1 = implode( " ", $exploded );
$joined2 = join( "_", $exploded ); // join with '_'

print_r( $joined1 );
echo "\n";
print_r( $joined2 );
echo "\n";


Output:

Hello World
Hello_World
```

To replace something with another (str_replace()):

```php
<?php

$wrongName1 = "John  Doe"; // two spaces
$wrongName2 = "John..Doe"; // two dots
```

```php
$fixedName1 = str_replace( "  ", " ", $wrongName1 ); //  replace two spaces with one space

echo $fixedName1;
echo "\n";
$fixedName2 = str_replace( "..", ".", $wrongName2 ); //  replace two dots with one dot
echo $fixedName2;

// it can even accept array as well. Pass the what you want to replace and what you want to replace with in the array.
$wrongName3 = "John  ..Doe";
$fixedName3 = str_replace( ["  ", ".."], [" ", "."], $wrongName3 );
echo $fixedName3;


Output:

John Doe
John.Doe
John .Doe
```

To check if one string exists another string (strpos()):

```php
<?php

$string = "Hello World";

$search = "World";

// will show from which index the string is found. It returns the index of the first occurrence or returns `false` if the substring is not found.
$result = strpos( $string, $search );

echo $result;


Output:

6
```

```php
<?php

// strpos() can take total 3 parameters. strpos( MainString = the string to search within, SubString = the substring to search for, Postition(Optional) = the position in the string to start the search from)
```

```php
$mainString = "Hello World, Welcome to the World of PHP"; $subString =
"World"; $position = 10;

// Use strpos to find the first occurrence of $subString in $mainString
starting from $position
$result = strpos($mainString, $subString, $position);
// strpos() starts searching for "World" from the 10th index of
`$mainString`. The first "World" occurs at index 6, but since the search
starts from index 10, it will find the next "World" which is at index 23.

// Output the position (index)
echo $result;
```

Output:

```
23
```

```php
<?php

$string = "Hello World";

$search = "World";

// will show from which index the string is found
$result = strpos( $string, $search );

if ( $result ) {
    echo "{$search} found";
} else {
    echo "{$search} not found";
}
// here the output will be completely okay with found as in result we got
the index number = 6
```

```php
<?php

$string = "Hello World";
$search = "Hello";

// will show from which index the string is found
$result = strpos( $string, $search );

if ( $result ) {
    echo "{$search} found";
} else {
    echo "{$search} not found";
```

```
}
// but here it will still say "not found". Why? Because here now 'Hello' is
starting from index 0 and it's being passed in the in the if condition, and
as 0 means false in php, so program printing "not found".
```

```php
<?php

$string = "Hello World";
$search = "Hello";

// will show from which index the string is found
$result = strpos( $string, $search );

if ( $result !== false ) {
    echo "{$search} found";
} else {
    echo "{$search} not found";
}
// now it will work, because now in $result is 0, and it's checing 0 !==
false, that means, now it's not false == false, keyword mismatch, it will
print the perfect output "found"


Output:

Hello found
```

To remove any html tags from the string (strip_tags()):

```php
<?php

$string = "<strong>Hello World</strong>. This sentence contains <p>HTML
tags</p>";

echo strip_tags( $string );
// the tags will be removed. It is important because, for using thise, all
tags will be removed and hackers will not be able to inject any js script in
the data for manipulation


Output:

Hello World. This sentence contains HTML tags
```

To count the number of times a substring occurs in a string (substr_count()):

```php
<?php

// count the number of vowels in each string in an array and reverse the
string
$strings = ["Hello", "World", "PHP", "Programming"];

foreach ( $strings as $string ) {
    $count = substr_count( $string, 'a' ) + substr_count( $string, 'e' ) +
substr_count( $string, 'i' ) + substr_count( $string, 'o' ) + substr_count(
$string, 'u' );
    $reversed = strrev( $string );
    echo "Original string: {$string}, Vowel count: {$count}, Reversed
string: {$reversed}\n";
}
```

Output:

```
Original string: Hello, Vowel count: 2, Reversed string: olleH
Original string: World, Vowel count: 1, Reversed string: dlroW
Original string: PHP, Vowel count: 0, Reversed string: PHP
Original string: Programming, Vowel count: 3, Reversed string: gnimmargorP
```

Classes, Objects and Interfaces:

Class & Object:

```php
<?php

// class is a blueprint where we organize our codes and classes increase the
reusability of our codes
// we initialize a class with a keyword called 'class'
// all the variables/functions or declared inside the class are called
'properties'
class Person {

    // to declare variables in the class, we use the keywords
'public'/'private'/'protected' or 'Access Modifiers'
    public $firstName = "John";
    public $lastName = "Doe";


    // same goes for functions. To declare functions in the class, we use
the keywords 'public'/'private'/'protected' or 'Access Modifiers'
    public function getName() {
```

```
        // to use the declared properties in the class, inside that same
class, we use the keyword '$this'. '$this->property'
        $fullName = $this->firstName . " " . $this->lastName;
        echo $fullName;
    }
}


// now the class is declared, the blueprint is created, but to get the
output from the blueprint/class we create a object with a keyword 'new'
$object = new Person();
// from the created object, which is stored in the variable, if we want to
now access the 'firstName' variable or 'getName' method of the class, we use
the keyword '->'. Imagine like we are pulling out (টান মারা) that function
from the '$object' variable which is holding the object of the class
echo $object->firstName;
echo "\n";
$object->getName();
echo "\n";


// and we can make multiple objects as well if we want
$object2 = new Person();
$object2->getName();
echo "\n";


$object3 = new Person();
$object3->getName();
echo "\n";


Output:

John
John Doe
John Doe
John Doe
```

Constructor(Magic Method) & its parameters:

- Its a method or function itself
- Also called magic method
- Can be executed automatically, no need to call
- Constructor function can take/recieve parameters
- Constructor can't return/we cannot use 'return' in this function

```php
<?php

// if we can declare any property which can be executed by itself without being called is 'Constructor', in php its a function called '__construct()'
class Car {

    public function __construct() {
        $num1 = 10;
        $num2 = 20;
        $sum = $num1 + $num2;
        echo $sum;
    }
}


// now we need to just create an object
$obj = new Car();
// here we are not accessing __construct function by calling like this ($obj->__construct()), without using it the function is called automatically
```

Output:

```
30
```

```php
<?php

class Car {

    public function __construct( $num3, $num4, $num5 ) {
        $num1 = 10;
        $num2 = 20;
        $sum = $num1 + $num2 + $num3 + $num4 + $num5;
        echo $sum;
    }
}


// to use parameters in the constructor, we need to send the arguments in the parenthesis : 'Car(arguments)'
$obj = new Car( 10, 20, 30 );
```

Output:

```
90
```

Inheritance (উত্তরাধিকার সূত্রে পাওয়া):

```php
<?php

// We can compare this to the relationship between a father and a child.
Just like a child inherits his father's property. Same happens here.
class Father {

    function addTwoNumbers() {

        $num1 = 100;
        $num2 = 100;
        $sum = $num1 + $num2;
        echo $sum;
    }
}

// A class can access another class's properties and methods if we can
inherit the class. For that, we use 'extends' keyword
class Son extends Father {

}

$sonObject = new Son();
$sonObject->addTwoNumbers();
echo "\n";
// even though the 'addTwoNumbers' method is created in the Father class, it
can be accessed in the Son class, because by extends keyword the Son class
inherits the Father class and got all the properties and methods of the
Father class


// its also possible to access the properties and methods of the Father
class from the Father class with father object
$fatherObject = new Father();
$fatherObject->addTwoNumbers();


Output:

200
200
```

Method Overriding:

```php
<?php

// its possible to change the value in the inherited class, but the parent
```

```
class will remain the same
class Father {

    function addTwoNumbers() {

        $num1 = 100;
        $num2 = 100;
        $sum = $num1 + $num2;
        echo $sum;
    }
}

class Son extends Father {

    // we are changing the method 'addTwoNumbers()' here and changing the
values, its called method overriding
    function addTwoNumbers() {
        $num1 = 100;
        $num2 = 200;
        $sum = $num1 + $num2;
        echo $sum;
    }
}

$sonObject = new Son();
$sonObject->addTwoNumbers();
echo "\n";

$fatherObject = new Father();
$fatherObject->addTwoNumbers();


Output:

300
200
```

More about Inheritance:

```
<?php

class Father {

    public $num1 = 100;
    public $num2 = 100;

}
```

```php
class Son extends Father {

    public function addTwoNumbers() {

        // to access the properties of the parent class in child class, we use the keyword '$this'
        $sum = $this->num1 + $this->num2;
        echo $sum;
    }
}

$sonObject = new Son();
$sonObject->addTwoNumbers();
```

Output:

```
200
```

```php
<?php

// if the variables are declared with the keyword 'static'
class Father {

    public static $num1 = 100;
    public static $num2 = 100;
}

class Son extends Father {

    public function addTwoNumbers() {

        // then to access the properties of the parent class in child class, we use the keyword parent and :: (Scope Resolution Operator) -> (parent::$property)
        $sum = parent::$num1 + parent::$num2;
        echo $sum;
    }
}

$sonObject = new Son();
$sonObject->addTwoNumbers();
```

Output:

```
200
```

```php
<?php

class Greetings {

    // if any function is declared with the keyword 'static', it belongs to the class, not the object. So, to call the function, we don't need to declare a object first. We can directly call it with (:: Scope Resolution Operator)

    public static function sayHello() {
        echo "Hello World!";
    }
}

// write the (class name) (:: Scope Resolution Operator) (function name)
Greetings::sayHello();


Output:

Hello World!
```

If variable is static, then use parent::$variable
If variable is not static, then use $this->variable
If function is static, then use :: (class :: function())
If function is not static, then use -> (object->function())

Abstract class:

```php
<?php

// if we use the keyword 'abstract' before the parent class, this class's properties and methods will be declared as abstract. So, we can't create an object from this class and its properties and methods. But the inherited class of this, can access the properties and methods of this abstract class.
abstract class Father {

    public $num1 = 10;
    public $num2 = 10;

    public function addTwoNumbers() {

        $sum = $this->num1 + $this->num2;
        echo $sum;
    }
}
```

```php
// will show error and won't work, as we can't create an object from this abstract class
$fatherObject = new Father();
$fatherObject->addTwoNumbers();
```

```php
<?php

// if we use the keyword 'abstract' before the parent class, this class's properties and methods will be declared as abstract. So, we can't create an object from this class and its properties and methods. But the inherited class of this, can access the properties and methods of this abstract class.
abstract class Father {

    public $num1 = 10;
    public $num2 = 10;

    public function addTwoNumbers() {

        $sum = $this->num1 + $this->num2;
        echo $sum;
    }
}


// but we can create an object from the inherited class and use the properties of the parent class even though that is abstract
class Son extends Father {

}

$sonObject = new Son();
$sonObject->addTwoNumbers();


Output:

20
```

Constructor in Inheritance:

```php
<?php

// constrcutor declared in parent class
class Father {

    public function __construct() {
```

```
        echo "This is father's class constructor";
    }
}

class Son extends Father {

}

$sonObject = new Son();


Output:

This is father's class constructor
```

```
<?php

class Father {

}



// constructor declared in the child class
class Son extends Father {

    public function __construct() {

        echo "This is son's class constructor";
    }
}

$sonObject = new Son();


Output:

This is son's class constructor
```

```
<?php

// constructor declared in both classes
class Father {

    public function __construct() {

        echo "This is father's class constructor";
    }
```

```php
}

class Son extends Father {

    public function __construct() {

        echo "This is son's class constructor";
    }
}

$fatherObject = new Father();
echo "\n";
$sonObject = new Son();
```

Output:

```
This is father's class constructor
This is son's class constructor
```

```php
<?php

// constructor declared in both classes
class Father {

    public function __construct() {

        echo "This is father's class constructor";
    }
}


// though constructor declared in parent class, can be accessed in child class automatically, but here both class have their own constructor and if the child class want to access parent's class constructor function, then child class need to use parent:: keyword ( parent::__construct() )
class Son extends Father {

    public function __construct() {

        parent::__construct();
        echo "\n";

        echo "This is son's class constructor";
    }
}

$sonObject = new Son();
```

```
Output:

This is father's class constructor
This is son's class constructor
```

final keyword (Preventing method overriding):

previously we saw that, its possible to change the value of the parent class's methods in the inherited class, but the parent class's methods will remain the same. Its called method overriding. But we can prevent this method overriding by simply putting a keyword 'final' while the parent class's method declaration.

```php
<?php

class Father {

    public final function addTwoNumbers() {

        $num1 = 100;
        $num2 = 100;
        $sum = $num1 + $num2;
        echo $sum;
    }
}

class Son extends Father {

    function addTwoNumbers() {

        $num1 = 100;
        $num2 = 200;
        $sum = $num1 + $num2;
        echo $sum;
    }
}


$sonObject = new Son();
$sonObject->addTwoNumbers();
echo "\n";

$fatherObject = new Father();
$fatherObject->addTwoNumbers();


Output:
```

```
PHP Fatal error:  Cannot override final method Father::addTwoNumbers() in
D:\5. PHP - Tutorials\p83.php on line 16

Fatal error: Cannot override final method Father::addTwoNumbers() in D:\5.
PHP - Tutorials\p83.php on line 16
```

Example of class and objects and constructor:

```php
<?php

class Account{
    public $accountNumber;
    public $balance;

    function __construct($accountNumber, $balance){
        $this->accountNumber = $accountNumber;
        $this->balance = $balance;
    }

    function getBalance(){
        return $this->balance;
    }

    function deposit($amount){
        $this->balance += $amount;
    }

    function withdraw($amount){
        if($amount > $this->balance){
            echo "Insufficient balance \n";
            return;
        }
        $this->balance -= $amount;
    }

}

$rahimsAccount = new Account('12344555', 5000); //instantiate
$karimsAccount = new Account('12344556', 6000);

echo $rahimsAccount->getBalance();
echo PHP_EOL;

$rahimsAccount->deposit(10000);
echo $rahimsAccount->getBalance();
echo PHP_EOL;

$rahimsAccount->withdraw(25000);
echo $rahimsAccount->getBalance();
```

```
// echo PHP_EOL;
// echo $karimsAccount->getBalance();
```

```
<?php
class Account{
    // for using the keyword 'private', we cannot access these variables outside of this class ( Like this after creating an object of this class : $object->balance; ) which we can do with 'public' keyword. To access them, we need methods only.
    private $accountNumber;
    private $balance;

    function __construct($accountNumber, $balance){
        $this->accountNumber = $accountNumber;
        $this->balance = $balance;
    }

    function getBalance(){
        return $this->balance;
    }

    function deposit($amount){
        $this->balance += $amount;
    }

    function withdraw($amount){
        if($amount > $this->balance){
            echo "Insufficient balance \n";
            return;
        }
        $this->balance -= $amount;
    }

}

$rahimsAccount = new Account("12345","10000");
echo $rahimsAccount->getBalance();
$rahimsAccount->withdraw(30000);

$rahimsAccount->balance = 100000;

$rahimsAccount->withdraw(30000);
echo $rahimsAccount->getBalance();
```

Interface:

Interfaces allow you to specify what methods a class should implement.
Interfaces make it easy to use a variety of different classes in the same way. When one or more classes use the same interface, it is referred to as "polymorphism".
Interfaces are declared with the `interface` keyword.

```php
<?php

// interface is a blueprint for classes. It allows us to define the methods that a class must implement
interface AccountInterface{

    function __construct($accountNUmber, $balance);
    public function getBalance();
    public function deposit($amount);
    public function withdraw($amount);
}

// to use the blueprint 'interface', we use the keyword 'implements', if any method of interface is missing in the class, it will throw error
class SavingsAccount implements AccountInterface{
    private $accountNUmber;
    private $balance;
    function __construct($accountNUmber, $balance){
        $this->$accountNUmber = $accountNUmber;
        $this->$balance = $balance;
    }
    public function getBalance(){
        return $this->balance;
    }
    public function deposit($amount){
        $this->balance += $amount;
    }

    public function withdraw($amount){}
}

class CurrentAccount implements AccountInterface{
    private $accountNUmber;
    private $balance;
    function __construct($accountNUmber, $balance){
        $this->$accountNUmber = $accountNUmber;
        $this->$balance = $balance;
    }
    public function getBalance(){
        return $this->balance;
    }
    public function deposit($amount){
        $this->balance += $amount;
    }
```

```
    public function withdraw($amount){}
}

class PriorityAccount implements AccountInterface{
    private $accountNUmber;
    private $balance;
    function __construct($accountNUmber, $balance){
        $this->$accountNUmber = $accountNUmber;
        $this->$balance = $balance;
    }
    public function getBalance(){
        return $this->balance;
    }
    public function deposit($amount){
        $this->balance += $amount;
    }
    public function withdraw($amount){}
}
```

Interface, Abstract, Implements, Extends all in one example:

```
<?php

interface AccountInterface {

    function __construct($accountNUmber, $balance);
    public function getBalance();
    public function deposit($amount);
    public function withdraw($amount);
}

abstract class AbstractAccount implements AccountInterface {
    protected $accountNUmber;
    protected $balance;
    function __construct($accountNUmber, $balance) {
        $this->accountNUmber = $accountNUmber;
        $this->balance = $balance;
    }
    public function getBalance() {
        return $this->balance;
    }
    public function deposit($amount) {
        $this->balance += $amount;
    }
    public function withdraw($amount) {
        if ($amount > $this->balance) {
            echo "Insufficient balance \n";
            throw new Exception("Insufficient Balance");
        }
```

```php
    }
}


class SavingsAccount extends AbstractAccount {
    public function withdraw($amount) {
        parent::withdraw($amount);

        if ($amount > 5000) {
            echo "You can not withdraw more than 5000 \n";
            return;
        }

        $this->balance -= $amount;
    }
}

class CurrentAccount extends AbstractAccount {
    public function withdraw($amount) {

        parent::withdraw($amount);
        $this->balance -= $amount;
    }
}

class PriorityAccount extends AbstractAccount {
    public function withdraw($amount) {

        parent::withdraw($amount);
        if ($amount > 50000) {
            echo "You can not withdraw more than 50000 \n";
            return;
        }

        $this->balance -= $amount;
    }
}

$rahimsAccount = new SavingsAccount(1234, 1000);
// $rahimsAccount = new PriorityAccount(1234,100000);
echo $rahimsAccount->getBalance();
echo PHP_EOL;
$rahimsAccount->deposit(1000);
echo $rahimsAccount->getBalance();
echo PHP_EOL;
$rahimsAccount->withdraw(4000);
echo $rahimsAccount->getBalance();
echo PHP_EOL;
$rahimsAccount->withdraw(60000);
echo $rahimsAccount->getBalance();
```

Polymorphism:

```php
<?php

// multiple interfaces
interface canEat{
    function eat();
}

interface canSleep{
    function sleep();
}

interface canFly{
    function fly();
}

interface canTalk{

}

interface canSwim{
    function swim();
}

// implementing multiple interfaces in one class, this is called
'Polymorphism'
class Human implements canEat, canSleep, canSwim {
    function eat(){
        echo "Human can eat \n";
    }

    function sleep(){
        echo "Human can sleep \n";
    }

    function swim(){
        echo "Human can swim \n";
    }
}
class Bird implements canEat, canSleep, canFly, canTalk {
    function eat(){
        echo "Bird can eat \n";
    }

    function sleep(){
        echo "Bird can sleep \n";
    }
```

```
    function fly(){
        echo "Bird can fly \n";
    }
}

class Fish implements canEat, canSleep, canSwim {
    function eat(){
        echo "Fish can eat \n";
    }

    function sleep(){
        echo "Fish can sleep \n";
    }

    function swim(){
        echo "Fish can swim \n";
    }
}

$human = new Human();
$bird = new Bird();
$fish = new Fish();

function canYouSwim(canSwim $object){
    echo $object->swim();
}


canYouSwim($human);
// canYouSwim($bird);
canYouSwim($fish);
```

JSON with PHP:

JSON stands for **J**ava**S**cript **O**bject **N**otation
JSON is a **text format** for storing and transporting data
JSON helps to communicate between different ecosystems. Data interchanging format. A common data format. Its a string data.

JSON encode in PHP:

PHP encode means -> associative array to JSON (JSON object)
PHP decode means -> JSON (JSON object) to PHP associative array

Single Dimension Associative Array to JSON encode:

```php
<?php

// JSON encode -> will be encoded from an associative array
$associativeArray = [

    "name" => "John",
    "age"  => 20,
];

// converting to JSON -> we will use 'json_encode()' function
$json = json_encode( $associativeArray );
echo $json;
// we will see that, the output is not an associative array, but a JSON
string or object starting and ending with { and }


Output:

{"name":"John","age":20}
```

Multi-Dimension Associative Array to JSON encode:

```php
<?php

// multi-dimension associative array
$associativeArray = [

    ["firstName" => "Bill", "lastName" => "Gates"],
    ["firstName" => "Mark", "lastName" => "Zuckerberg"],
];

// converting to JSON -> we will use 'json_encode()' function
$json = json_encode( $associativeArray );
echo $json;
// we will see that, the output is now a JSON array with JSON objects
starting and ending with [ and ]. JSON array will hold JSON objects.


Output:

[{"firstName":"Bill","lastName":"Gates"},
{"firstName":"Mark","lastName":"Zuckerberg"}]
```

So, for single dimensional associative array, if we encode to JSON, we will get a JSON OBJECT.
For, multi-dimensional associative array, if we encode to JSON, we will get JSON ARRAY.

JSON decode in PHP:

JSON object decode to single dimensional Associative array:

```php
<?php

// JSON string/object
$stringData = '{"firstName":"Bill","lastName":"Gates"}';

// decoding JSON to PHP Associative array
$data = json_decode( $stringData );
print_r( $data );


Output:

stdClass Object
(
    [firstName] => Bill
    [lastName] => Gates
)
```

JSON array decode to multi-dimensional Associative array:

```php
<?php

// JSON array
$stringData = '[{"firstName":"Bill","lastName":"Gates"},
{"firstName":"Mark","lastName":"Zuckerberg"}]';

// decoding JSON to PHP Associative array
$data = json_decode( $stringData );
print_r( $data );


Output:

Array
(
    [0] => stdClass Object
        (
            [firstName] => Bill
            [lastName] => Gates
        )

    [1] => stdClass Object
        (
            [firstName] => Mark
```

```
            [lastName] => Zuckerberg
        )
)
```

So, for JSON OBJECT, if we decode to PHP, we will get Single Dimensional Associative Array.
For, JSON ARRAY, if we decode to PHP, we will get Multi-Dimensional Associative Array.

SESSION:

Session in PHP is **a way of temporarily storing and making data accessible across all the website pages**. It will create a temporary file that stores various session variables and their values. This will be destroyed when you close the website.

A session in PHP allows the webserver to get the information about when you started a website, what you were doing, when you closed the website and other related information. It is required because, unlike the PC or mobile, the webserver does not have any information about you. That's where sessions come into the picture.

These sessions have session variables that store all the necessary information into a temporary file. By default, it will destroy this file when you close the website. Thus, to put it simply, a session in PHP helps in storing information about users and makes the data available to all the pages of a website or application until you close it.

A way to make the application memorize something until the application isn't closed by the user. Session is a temporary storage system for backend.

Use Case: Login/Registration (Auth)

To store data in SESSION:

```php
<?php

// to start session:
session_start();

// $_SESSION["variable name"] = "value";
$_SESSION["name"] = "John Doe";
// SESSION is a super global variable, that means, from any other file we
can access it.
```

in another file:

```php
<?php

// to start session:
session_start();

// $_SESSION["variable name"] = "value";
$_SESSION["email"] = "johndoe@gmail.com";
```

To retrieve data stored in SESSION:

```php
<?php

session_start();

echo $_SESSION["email"];
echo "<br/>";
echo $_SESSION["name"];
// serve the project on PHP server

Output:

johndoe@gmail.com
John Doe
```

To erase all data from SESSION:

```php
<?php

session_start();

// will delete all session data
session_destroy();
```

To erase a single data from SESSION:

```php
<?php

session_start();

// destroying a single session variable
unset( $_SESSION["email"] );
```

COOKIES:

Same as SESSION, but Session is a temporary storage system for backend. And COOKIES is a temporary storage system for frontend. It will be seen from the browser inspection also.

To store data in COOKIES:

```php
<?php

// setcookie("variable", "value");
setcookie("token1", "abcd");
// COOKIE is a super global variable, that means, from any other file we can access it.
```

in another file:

```php
<?php

setcookie("token2", "efgh");
setcookie("token3", "ijkl");
setcookie("token4", "mnop");
```

To retrieve data stored in COOKIES:

```php
<?php

echo $_COOKIE["token1"];
echo "<br/>";
echo $_COOKIE["token2"];
echo "<br/>";
echo $_COOKIE["token3"];
echo "<br/>";
echo $_COOKIE["token4"];
// serve the project on PHP server and inspect from the browser, go to storage and the cookie will be set there


Ouput:

abcd
efgh
ijkl
mnop
```

To erase data from COOKIES:

```php
<?php
```

```
// just keep the value blank
setcookie("token1", "");
```

Types of Error:

How many error can we get in PHP.

Parse Error:

A parse error: syntax error, unexpected appears when the PHP interpreter detects a missing element.

```
<?php

// parse error:
echo "Hello World" // no semicolon at the end of the line


Output:

Parse error: syntax error, unexpected end of file, expecting "," or ";" in
D:\5. PHP - Tutorials\p103.php on line 4
PHP Parse error:  syntax error, unexpected end of file, expecting "," or ";"
in D:\5. PHP - Tutorials\p103.php on line 4
```

Fatal Error:

Startup fatal errors occur when the system cannot run the code during program startup and installation.

```
<?php

// fatal error:
function sum( $a, $b ) {
    $sum = $a + $b;
    return $sum;
}

// function name didn't match
summation( 1, 2 );


Output:

Fatal error: Uncaught Error: Call to undefined function summation() in D:\5.
```

```
PHP - Tutorials\p103.php:11
Stack trace:
#0 {main}
  thrown in D:\5. PHP - Tutorials\p103.php on line 11
PHP Fatal error:  Uncaught Error: Call to undefined function summation() in
D:\5. PHP - Tutorials\p103.php:11
Stack trace:
#0 {main}
  thrown in D:\5. PHP - Tutorials\p103.php on line 11
```

Fatal Error(Strict Error/Static Property Error):

```php
<?php

// fatal error (strict/static property error):
class MyClass {
    function myFunction( $a, $b ) {
        echo $a + $b;
    }
}

MyClass::myFunction( 10, 10 );
// we all know that, to access a class's static function we use the scope
resolution operator, but here somehow we are trying to access the function
with scope resolution operator, but didn't use the keyword 'static' in the
function declaration. So this '::' operator won't access the static
function. That means we are trying to access the function with something
that isn't appropriate.


Output:

Fatal error: Uncaught Error: Non-static method MyClass::myFunction() cannot
be called statically in D:\5. PHP - Tutorials\p103.php:10
Stack trace:
#0 {main}
  thrown in D:\5. PHP - Tutorials\p103.php on line 10
PHP Fatal error:  Uncaught Error: Non-static method MyClass::myFunction()
cannot be called statically in D:\5. PHP - Tutorials\p103.php:10
Stack trace:
#0 {main}
  thrown in D:\5. PHP - Tutorials\p103.php on line 10
```

Warning Error:

They show a problem in the code that is not severe enough to stop script execution.

```php
<?php

// warning error:
include "bangladesh.php";
// code won't break, but it will show warning error, as there is no file named bangladesh.php


Output:

Warning: include(): Failed opening 'bangladesh.php' for inclusion (include_path='C:\xampp\php\PEAR') in D:\5. PHP - Tutorials\p103.php on line 4
```

Warning Error(Notice Error):

Just like Fatal Error. A notice error in PHP is a non-critical error that indicates a minor issue in the code, but does not stop the script from running.

```php
<?php

// warning error (notice error):
$a = "Hello World";
echo $b;
// code won't break, but it will show warning error, as there is no value assigned in the variable $b


Output:

Warning: Undefined variable $b in D:\5. PHP - Tutorials\p103.php on line 5
PHP Warning:  Undefined variable $b in D:\5. PHP - Tutorials\p103.php on line 5
```

User Error:

In PHP, a user error is an error that is generated by the user.

Error Handling:

Where to do Error Handling:

- Database
- File Access

- Networking
  If something happens to the server sides, which won't be a problem caused by the written codes.

With try-catch block:

```php
<?php

// handle error with try catch block. It won't break the application if an error occurs.
try {
    $a = "hello";
    echo $b; // $b isn't declared as a variable, which can't be printed
} catch ( Exception $e ) {
    echo $e->getMessage();
}
// will show error message in the console but it won't break the application


Output:

Warning: Undefined variable $b in D:\5. PHP - Tutorials\p103.php on line 6
PHP Warning:  Undefined variable $b in D:\5. PHP - Tutorials\p103.php on line 6
```

try-catch block with error_log():

```php
<?php

// we use error_log() and generate an error.log file so that, we can also get to know which error was encountered by the user.

// Set error reporting to log warnings and notices
error_reporting( E_ALL ); // Report all types of errors
ini_set( 'log_errors', 1 ); // Enable error logging
ini_set( 'error_log', 'error.log' ); // Set the error log file

try {
    $a = "hello";
    echo $b; // This will trigger a notice, which will be logged
} catch ( Exception $e ) {
    error_log( $e->getMessage(), 3, 'error.log' ); // this file will be automatically generated if any error occurs.
    echo "Error!";
}


Output:
```

```
Warning: Undefined variable $b in D:\5. PHP - Tutorials\p103.php on line 12

in error.log file:

[02-Sep-2024 06:22:08 Europe/Berlin] PHP Warning:  Undefined variable $b in
D:\5. PHP - Tutorials\p103.php on line 12
```

File Handling in PHP:

To handle a file in php, we use fopen() function.
Methods of handling a file in php:

- fopen() -> open a file
- fclose() -> close a file
- fread() -> read the file
- fwrite() -> write the file
- feof() -> command to go through until end of file

Types of Modes:

- r -> read
- w -> write (Will erase the file if it already exists and will create a new file if it doesn't exist. Previous contents will be erased every time you run the code)
- r+ -> read and write
- w+ -> write and read
- a -> append (Will append the contents at the end of the file)
- a+ -> append and read
- x -> exclusive (Will create a new file if it doesn't exist and will throw an error if it already exists)
- x+ -> exclusive and read

To Open and Write(w) On a File:

```php
<?php

// opening a file : fopen(filename, mode)
// for 'w' mode, it will erase the file if it already exists and will create
a new file if it doesn't exist. Previous contents will be erased every time
you run the code
$file = fopen( "test.txt", "w" );

fwrite( $file, "Hello World" );
```

```php
fclose( $file );


// test.txt file will be created and the file will contain 'Hello World'
Output:

Hello World
```

```php
<?php

$books = [
    "book1",
    "book2",
    "book3",
    "book4",
    "book5",
];

$file = fopen( "books.txt", "w" );

foreach ( $books as $book ) {
    fwrite( $file, $book . PHP_EOL );
}

fclose( $file );


Output:

book1
book2
book3
book4
book5
```

To Open and Write(w) The File With 'append'(a) mode:

```php
<?php

// for 'a' mode, it will not erase previous contents every time you run the
code and will add the contents at the end of the file
$file = fopen( "test.txt", "a" );
$data = "Hello World\n";

fwrite( $file, $data );
fclose( $file );
```

```
Output:

Hello World
Hello World
```

```php
<?php

$books = [
    "book1",
    "book2",
    "book3",
    "book4",
    "book5",
];

$file = fopen( "books.txt", "a" );

foreach ( $books as $book ) {
    fwrite( $file, $book . PHP_EOL );
}

fclose( $file );
```

```
Output:

book1
book2
book3
book4
book5
book1
book2
book3
book4
book5
```

file_put_contents() -> to write on a file:

```php
<?php

$books = <<<EOD
book 1
book 2
book 3
```

```
EOD; // EOD = End Of Document

file_put_contents( "books2.txt", $books );


Output:

book 1
book 2
book 3
```

file_put_contents() -> with append mode:

```
<?php

$books = <<<EOD
book 1
book 2
book 3
EOD;// EOD = End Of Document

// to add the contents at the end of the file with new line
file_put_contents( 'books2.txt', $books . PHP_EOL, FILE_APPEND );


Output:

book 1
book 2
book 3
book 1
book 2
book 3
```

To read file:

```
<?php

// read the file
$file = fopen( "books.txt", "r" );

while ( !feof( $file ) ) {
    echo fgets( $file );
}


Output:
```

```
book1
book2
book3
book4
book5
book1
book2
book3
book4
book5
```

More smart way:

file():

```php
<?php

$file = file( "books.txt" );
// contents of books.txt will be saved as array in $file variable

// as we got an array, so we can run a foreach loop to print the contents
foreach ( $file as $line ) {
    echo $line;
}
```

```
Output:

book1
book2
book3
book4
book5
book1
book2
book3
book4
book5
```

```php
<?php

$file = file( "books.txt" );

// count the number of lines
echo "Total books : " . count( $file );
```

```
output:

Total books : 10
```

Another smart way:

file_get_contents():

```php
<?php

$content = file_get_contents( "books.txt" );
// This won't return an array

echo $content;


Output:

book1
book2
book3
book4
book5
book1
book2
book3
book4
book5
```

```php
<?php

$content = file_get_contents( "books.txt" );

$books = explode( "\n", $content );
echo "Total books : " . count( $books ) - 1;


Output:

Total books : 10
```

file_get_contents() -> opening any file even with url:

```php
<?php
```

```php
$content = file_get_contents(
"https://raw.githubusercontent.com/sickcodes/Docker-OSX/master/Dockerfile"
);
echo $content;
```

Working with csv (comma separated values) files:

Read with composer:

```php
<?php

// after running "composer require league/csv" in terminal
include "vendor/autoload.php";
use League\Csv\Reader;


$price = 0;
$reader = Reader::createFromPath( "books.csv", "r" );
$books = $reader->getRecords();

foreach ( $books as $book ) {
    $output = "Book Name = {$book[0]}, Author = {$book[1]}" . PHP_EOL;
    echo $output;
    $price += $book[2];
}

echo "Total price = {$price}";


Output:

Book Name = To Kill a Mockingbird, Author = Harper Lee
Book Name = The Lord of the Rings, Author = J.R.R. Tolkien
Book Name = The Hunger Games, Author = Suzanne Collins
Book Name = Pride and Prejudice, Author = Jane Austen
Book Name = The Catcher in the Rye, Author = J.D. Salinger
Book Name = The Hitchhiker's Guide to the Galaxy, Author = Douglas Adams
Book Name = The Handmaid's Tale, Author = Margaret Atwood
Book Name = The Nightingale, Author = Kristin Hannah
Book Name = The Power, Author = Naomi Alderman
Book Name = The Immortal Life of Henrietta Lacks, Author = Rebecca Skloot
Total price = 129.9
```

```php
<?php

include "vendor/autoload.php";
use League\Csv\Reader;
```

```php
// if there is headers on a csv file, if we need to exclude them:
$price = 0;
$reader = Reader::createFromPath( "books2.csv", "r" );
$reader->setHeaderOffset( 0 );
$books = $reader->getRecords();

foreach ( $books as $book ) {
    $output = "Book Name = {$book['Title']}, Author = {$book['Author']}" .
PHP_EOL;
    echo $output;
    $price += $book['Price'];
}

echo "Total price = {$price}";
```

Output:

```
Book Name = To Kill a Mockingbird, Author = Harper Lee
Book Name = The Lord of the Rings, Author = J.R.R. Tolkien
Book Name = The Hunger Games, Author = Suzanne Collins
Book Name = Pride and Prejudice, Author = Jane Austen
Book Name = The Catcher in the Rye, Author = J.D. Salinger
Book Name = The Hitchhiker's Guide to the Galaxy, Author = Douglas Adams
Book Name = The Handmaid's Tale, Author = Margaret Atwood
Book Name = The Nightingale, Author = Kristin Hannah
Book Name = The Power, Author = Naomi Alderman
Book Name = The Immortal Life of Henrietta Lacks, Author = Rebecca Skloot
Total price = 129.9
```

Read with using spl:

```php
<?php

//use spl to read csv
$file = new SplFileObject( "books.csv" );
$file->setFlags( SplFileObject::READ_CSV );
$file->setCsvControl( "," );
$price = 0;

foreach ( $file as $row ) {
    echo $row[0] . " - " . $row[1] . " - " . $row[2] . PHP_EOL;
    $price += $row[2];
}

echo "Total price = {$price}";
```

Output:

```
To Kill a Mockingbird - Harper Lee -  12.99
The Lord of the Rings - J.R.R. Tolkien -  15.99
The Hunger Games - Suzanne Collins -  10.99
Pride and Prejudice - Jane Austen -  8.99
The Catcher in the Rye - J.D. Salinger -  11.99
The Hitchhiker's Guide to the Galaxy - Douglas Adams -  9.99
The Handmaid's Tale - Margaret Atwood -  14.99
The Nightingale - Kristin Hannah -  13.99
The Power - Naomi Alderman -  12.99
The Immortal Life of Henrietta Lacks - Rebecca Skloot -  16.99
Total price = 129.9
```

Read with fopen():

```php
<?php

$file = fopen("books4.csv","r");
while($line = fgetcsv($file)){
    print_r($line);
}
```

Output:

```
Array
(
    [0] => To Kill a Mockingbird
    [1] => Harper Lee
    [2] => 12.99
)
Array
(
    [0] => The Lord of the Rings
    [1] => J.R.R. Tolkien
    [2] => 15.99
)
Array
(
    [0] => The Hunger Games
    [1] => Suzanne Collins
    [2] => 10.99
)
Array
(
    [0] => Pride and Prejudice
    [1] => Jane Austen
    [2] => 8.99
)
```

```
Array
(
    [0] => The Catcher in the Rye
    [1] => J.D. Salinger
    [2] => 11.99
)
Array
(
    [0] => The Hitchhiker's Guide to the Galaxy
    [1] => Douglas Adams
    [2] => 9.99
)
Array
(
    [0] => The Handmaid's Tale
    [1] => Margaret Atwood
    [2] => 14.99
)
Array
(
    [0] => The Nightingale
    [1] => Kristin Hannah
    [2] => 13.99
)
Array
(
    [0] => The Power
    [1] => Naomi Alderman
    [2] => 12.99
)
Array
(
    [0] => The Immortal Life of Henrietta Lacks
    [1] => Rebecca Skloot
    [2] => 16.99
)
```

Write:

```php
<?php

$books = [
    ["To Kill a Mockingbird", "Harper Lee", 12.99],
    ["The Lord of the Rings", "J.R.R. Tolkien", 15.99],
    ["The Hunger Games", "Suzanne Collins", 10.99],
    ["Pride and Prejudice", "Jane Austen", 8.99],
    ["The Catcher in the Rye", "J.D. Salinger", 11.99],
    ["The Hitchhiker's Guide to the Galaxy", "Douglas Adams", 9.99],
    ["The Handmaid's Tale", "Margaret Atwood", 14.99],
```

```
    ["The Nightingale", "Kristin Hannah", 13.99],
    ["The Power", "Naomi Alderman", 12.99],
    ["The Immortal Life of Henrietta Lacks", "Rebecca Skloot", 16.99],
];

$_books = [];

foreach ( $books as $book ) {
    $line = implode( ", ", $book );
    $_books[] = $line;
}

$data = implode( PHP_EOL, $_books );
file_put_contents( "books3.csv", $data );
```

Output:

```
To Kill a Mockingbird, Harper Lee, 12.99
The Lord of the Rings, J.R.R. Tolkien, 15.99
The Hunger Games, Suzanne Collins, 10.99
Pride and Prejudice, Jane Austen, 8.99
The Catcher in the Rye, J.D. Salinger, 11.99
The Hitchhiker's Guide to the Galaxy, Douglas Adams, 9.99
The Handmaid's Tale, Margaret Atwood, 14.99
The Nightingale, Kristin Hannah, 13.99
The Power, Naomi Alderman, 12.99
The Immortal Life of Henrietta Lacks, Rebecca Skloot, 16.99
```

Write with fputcsv():

```php
<?php

$books = [
    ["To Kill a Mockingbird", "Harper Lee", 12.99],
    ["The Lord of the Rings", "J.R.R. Tolkien", 15.99],
    ["The Hunger Games", "Suzanne Collins", 10.99],
    ["Pride and Prejudice", "Jane Austen", 8.99],
    ["The Catcher in the Rye", "J.D. Salinger", 11.99],
    ["The Hitchhiker's Guide to the Galaxy", "Douglas Adams", 9.99],
    ["The Handmaid's Tale", "Margaret Atwood", 14.99],
    ["The Nightingale", "Kristin Hannah", 13.99],
    ["The Power", "Naomi Alderman", 12.99],
    ["The Immortal Life of Henrietta Lacks", "Rebecca Skloot", 16.99],
];

$file = fopen( "books4.csv", "w" );

foreach ( $books as $book ) {
```

```
    fputcsv( $file, $book );
}


Output:

To Kill a Mockingbird, Harper Lee, 12.99
The Lord of the Rings, J.R.R. Tolkien, 15.99
The Hunger Games, Suzanne Collins, 10.99
Pride and Prejudice, Jane Austen, 8.99
The Catcher in the Rye, J.D. Salinger, 11.99
The Hitchhiker's Guide to the Galaxy, Douglas Adams, 9.99
The Handmaid's Tale, Margaret Atwood, 14.99
The Nightingale, Kristin Hannah, 13.99
The Power, Naomi Alderman, 12.99
The Immortal Life of Henrietta Lacks, Rebecca Skloot, 16.99
```

Checking the existence of a file (file_exists()):

```
<?php

$exists = file_exists( "books.txt" );
echo $exists;


output:

1
```

Deleting a file (unlink()):

```
<?php

$exists = file_exists( "books5.txt" );

if ( $exists ) {
    unlink( "books5.txt" );
}
```

To see any specific type of files existing in a directory (glob ()):

```
<?php

$phpFiles = glob( "*.php" );
print_r( $phpFiles );
```

```php
$csvFiles = glob( "*.csv" );
print_r( $csvFiles );
```