Web Development with PHP
Practical - 4
PHP Functions

PHP functions are blocks of reusable code that perform specific tasks. They allow you to encapsulate functionality and avoid code duplication. PHP provides a rich set of built-in functions, such as string manipulation, array manipulation, mathematical operations, file handling, and more. Additionally, you can create your own custom functions to meet specific requirements. Functions in PHP can accept arguments and return values, making them flexible and versatile.

PHP functions are an essential component of the PHP programming language that allow you to group a set of statements together and execute them as a single unit. Functions provide a way to organize and reuse code, making your PHP programs more efficient, modular, and maintainable.

In PHP, functions are defined using the function keyword, followed by the function name and a pair of parentheses. You can also specify parameters within the parentheses to accept input values. The function body consists of a block of code enclosed within curly braces, which contains the instructions to be executed when the function is called.

Functions in PHP offer several advantages. First and foremost, they promote code reusability. By encapsulating a specific set of instructions within a function, you can reuse that code throughout your program or in multiple projects. This saves development time and effort, as you don't have to rewrite the same code multiple times.

Another benefit of PHP functions is modularity. Functions allow you to break down complex tasks into smaller, manageable units. This promotes code organization, as you can group related operations within separate functions. It also improves code readability and comprehension, as functions act as self-contained units of code with a clearly defined purpose.

PHP functions can accept parameters, which enable you to pass data into the function for processing. Parameters make functions flexible and customizable, as you can pass different values each time the function is called. This allows you to reuse the same function logic with different inputs, producing different results as needed.

Furthermore, PHP functions can return values. This means that the function can compute a result and provide it back to the calling code. Returning values allows you to extract and use the computed results for further calculations, decision-making, or other operations within your PHP program.

Advantages

Code Reusability: Functions allow us to write reusable code. By encapsulating a block of code into a function, we can use it multiple times throughout our program without having to rewrite the same code. This saves time and effort and promotes code efficiency.

Modularity: Functions enable us to break down a complex problem into smaller, manageable units. Each function performs a specific task, making the overall program structure more organized and easier to understand. Modularity improves code readability and maintainability

Abstraction: Functions provide a level of abstraction by hiding the implementation details. Instead of worrying about how a particular task is accomplished, we can focus on using the function and relying on its functionality. This simplifies the coding process and enhances code comprehension.

Code Readability: Functions improve code readability by promoting a structured and logical approach to programming. Well-named functions with clear parameters and return types make the code more self-explanatory. This makes it easier for other developers to understand and collaborate on the codebase.

Built-in Functions

- String Functions:
  PHP offers numerous string manipulation functions, such as
  - ➔ strlen() for getting the length of a string,
  - ➔ substr() for extracting substrings,
  - ➔ str_replace() for replacing text within a string,
  - ➔ strtolower() and strtoupper() for converting case, and
  - ➔ implode() for joining array elements into a string.

- Mathematical Functions:
  PHP provides a variety of mathematical functions, including basic arithmetic operations like
  - abs() for absolute value,
  - sqrt() for square root,
  - pow() for exponentiation,
  - round() for rounding numbers, and
  - rand() for generating random numbers.

- Array Functions:
  PHP offers a rich set of functions for working with arrays. Examples includes
  - count() for getting the number of elements in an array,
  - array_push() and array_pop() for adding and removing elements from an array,
  - array_merge() for merging arrays, and
  - array_filter() for filtering array elements based on a callback function.

- Date and Time Functions:
  PHP provides functions to work with dates and times, such as
  - date() for formatting dates,
  - time() for getting the current Unix timestamp,
  - strtotime() for converting textual date representations into timestamps, and functions for manipulating and comparing dates, like strtotime() and date_diff().

- File System Functions:
  PHP offers functions for interacting with the file system, such as
  - file_exists() for checking if a file exists,
  - file_get_contents() for reading file contents,
  - file_put_contents() for writing data to a file, and
  - unlink() for deleting files.

- Database Functions:
  PHP provides functions for connecting to databases, executing queries, and retrieving results. Popular ones include
  - mysqli_connect() for establishing a connection to a MySQL database,
  - mysqli_query() for executing SQL queries, and
  - mysqli_fetch_assoc() for fetching rows from query results.

User Defined Functions

- Function Declaration: User-defined functions are declared using the function keyword, followed by the function name and parentheses. Parameters can be defined within the parentheses if the function requires input values.

- Function Body: The function body contains the code that defines the functionality of the function. It is enclosed within curly braces {}. This is where the specific tasks or operations are performed.

- Return Statement: User-defined functions can optionally have a return statement to return a value after performing the desired operations. The return statement is used to pass data back to the calling code.

- Function Call: To execute a user-defined function, it needs to be called by its name followed by parentheses. If the function expects input values, they are passed as arguments within the parentheses.

Syntax :

```
function functionName($param1, $param2,…)
{
      code to be executed;
      return $value;
}
```

Examples:

```
//-------------------------------------
function writeName() {
     echo "ABC";
}
echo "My name is ";
writeName();
//-------------------------------------
function calculateSquare($number) {
    $square = $number * $number;
    return $square;
}
// Calling the function
$result = calculateSquare(5);
echo "Square of 5 is: " . $result;
//-------------------------------------
function addNumbers($num1, $num2) {
    $sum = $num1 + $num2;
    return $sum;
}
// Calling the function
$result = addNumbers(10, 5);
echo "The sum of 10 and 5 is: " . $result;
//-------------------------------------
```

Function Arguments

In PHP, function arguments are the input values that are passed to a function when it is called. These arguments allow us to provide dynamic data to the function, which can be processed and utilized within the function's code.

- Declaring Function Arguments:
  When defining a function, you can specify the arguments it expects within the parentheses following the function name.
  For example:
  ```
  function greet($name) {
      // Function code here
  }
  ```
- Passing Arguments to Functions:
  To pass values to a function, you include them within the parentheses when calling the function.
  For example:
  ```
  greet("Students");
  ```
- Number of Arguments:
  Functions in PHP can accept any number of arguments, including none. You can define multiple arguments by separating them with commas.
  For example:

```
function calculateSum($num1, $num2) {
    // Function code here
}
```

- Default Values:
  You can assign default values to function arguments. If an argument is not provided when the function is called, it will take its default value.
  For example:
  ```
  function greet($name = "Guest") {
      // Function code here
  }
  ```

- Variable Arguments:
  PHP also supports variable-length argument lists using the func_get_arg(), func_get_args() and func_num_args() functions or using **...** token. This allows a function to accept a variable number of arguments.

In PHP, function arguments are passed by value by default, which means a copy of the variable's value is passed to the function. PHP also supports passing arguments by reference. When an argument is passed by reference, changes made to the parameter inside the function will affect the original variable outside the function. This is known as "call by reference".

```
//Declaring a Function with Reference Parameter:
function incrementByReference(&$num) {
    $num++;
}

//Calling a Function with Reference Parameter:
$value = 5;
incrementByReference($value);
echo $value;  // Output: 6
```

Examples:

Default Argument Value :

```
function setHeight($minheight = 50) {
     echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight();
```

Functions - Returning values:

```
function sum($x, $y) {
     $z = $x + $y;
     return $z;
}
```

Return Type Declarations:

```
function addNumbers(int $num1, int $num2): int {
    $sum = $num1 + $num2;
    return $sum;
}
// Calling the function and storing the returned value
$result = addNumbers(5, 3);
echo "The sum is: " . $result;
```

The addNumbers() function has a return type declaration ***: int*** *after the parameter list. This means that the function is expected to return an integer value. If the function tries to return a value of a different type or no value at all, a TypeError will be thrown.*

Recursive Functions:

```
function factorial($n) {
    if ($n < 2) {
        return 1;
    } else {
        return ($n * factorial($n - 1));
    }
}
```

Type Hints:

To enforce the types for function parameters and return value, you can use type hints. Note that PHP also allows you to use type hints for class properties and methods. The type hints ensure that PHP will check the type of a value at the call time and throw a TypeError if there is a mismatch. To add a type hint to a parameter, you place a type in front of it like this:

```
<?php
    function my_function(type $param1, type param2, ...)
    {
       // ...
}
```

In PHP 5, you can use array, callable, and class for type hints. In PHP 7+, you can also use scalar types such as bool, float, int, and string.

To specify a return value's type for a function, you add the type after the function header like this:

```
<?php
    function my_function(type $param1, type $param2,
    ...) : type
    {
        // ..
    }
```

Starting from PHP 7.0, if a function doesn't return a value, you use the void type.

Starting from PHP 8.0, if a function returns a value of several types, you can declare it as a union type. For example:

```
<?php

function add($x, $y): int | float
{
    return $x * $y;
}

echo add(10, 20); // 200 (int)
echo add(1.5, 2.5); // 3.75 (float)
```

If a function returns a value of many types, you can use the mixed type. The mixed type means one of several types. The mixed type. It's equivalent to the following union type:

```
object|resource|array|string|int|float|bool|null
```

The mixed has been available since PHP 8.0.0.

PHP implicitly coerces the values to the target types by default. For example, PHP coerces the floats into integers.
To enable strict typing, you can use the declare(strict_types=1); directive at the beginning of the file.

```
<?php

    declare(strict_types=1);
    function add(int $x, int $y){
        return $x + $y;
    }
    echo add(1.5, 2.5);
```

By adding the strict typing directive to the file, the code will execute in the strict mode. PHP enables the strict mode on a per-file basis. In the strict mode, PHP expects the values with the type matching with the target types. If there's a mismatch, PHP will issue an error.

```
Fatal error: Uncaught TypeError: Argument 1 passed to add()
must be of the type int, float given, called in ...
```

Note that you need to place the declare(strict_types=1); directive at the beginning of the script file before any other statements. The strict typing directive has a special case when the target type is float. If the target type is float, you can pass a value of type integer.

Suppose you define the add() function in the functions.php, And you include the functions.php in the index.php. Even though the functions.php declares the strict typing directive, it does not affect the index.php. When you call a function defined in a file with strict typing (functions.php) from a file without strict typing (index.php), PHP will respect the preference of the caller (index.php). That means it's up to the caller to decide whether to use the strict mode or not. In this case, the index.php won't execute in the strict mode.

The nullable type:
The following defines a function that accepts a string and returns the uppercase of that string:

```
<?php
    function upper(string $str): string
    {
        return strtoupper($str);
    }
    $str = null;
    echo upper($str);
```

If you pass an argument with null, you'll get an error:

```
Error:
Fatal error: Uncaught TypeError: Argument 1 passed to upper()
must be of the type string, null given
)
```

To fix this, you can make the $str parameter nullable like this:

```
<?php
    function upper(?string $str): string
    {
        return strtoupper($str);
    }
    $str = null;
    echo upper($str);
```

The nullable type was introduced in PHP 7.1.

PHP allows you to mark the type declarations and returns values as nullable by prefixing the type name with a question mark (?).

In the above example, we add the ? to the string type of the $str parameter. The ?string allows you to pass a string argument or null. Note that the mixed type already includes the null type. Therefore, you don't need to include nullable mixed.

Anonymous Functions:
The user defined function has no name and ends with a semicolon. This is because unlike regular functions, which are code constructions, an anonymous function is an expression.

```
function($number){
    return($number * 5);
};
```

This anonymous function code can never be called. The semicolon at the end of the function definition ensures that the function is treated as an expression. Since it is treated as an expression, it can be assigned to a variable, allowing it to be called by referencing the variable. It can also be passed to another function, which makes the anonymous function a

callback. Another point to make about anonymous functions is that they can be returned from another function. This is referred to as a closure.

The latest version of PHP, 7.4, introduced a new syntax for anonymous functions:

```
$double = fn($param1) => 2*$param1;
```

The use of the arrow function (=>) allows a short closure to be used for creating one-line functions. This is helpful in writing tighter code. Also notice that the return keyword is not necessary for single expressions. The value of that expression (2*$param1) in the above example is returned. Anonymous functions are most useful as the value of callback parameters, but they have other use cases. They are throw-away functions when you need a function that you want to only use once.

Date and Time Functions
The date() function formats a timestamp using a specified format:
Syntax : date(format, timestamp) ;

```
date(string $format, int|null $timestamp = null): string
```

Parameter	Description
format	Required. Specifies the format of the timestamp
timestamp	Optional. Specifies a timestamp. Default is the current date and time

The date() function has two parameters:

- $format is a string that determines how the $timestamp should be formatted.
- $timestamp is an Unix timestamp. The $timestamp parameter is optional. If you omit the $timestamp or use null, then it will default to the current timestamp. In other words, it defaults to the value of the time() function.

The date() function returns the formatted date string.

Format characters

- ➔ d - Represents the day of the month (01 to 31)
- ➔ m - Represents a month (01 to 12)
- ➔ Y - Represents a year (in four digits)
- ➔ l (lowercase 'L') - Represents the day of the week
- ➔ Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.
- ➔ h - 12-hour format of an hour with leading zeros (01 to 12) » i - Minutes with leading zeros (00 to 59)
- ➔ s - Seconds with leading zeros (00 to 59)
- ➔ a - Lowercase Ante meridiem and Post meridiem (am or pm)

Using the PHP date() function to show the current year. The following example uses the date()

function to show how to display the current year:

```
echo date('Y');
```

Using the date() function to format a date for the MySQL database. To insert a date into a datetime column in the MySQL database, you use the date format as YYYY-MM-DD HH:MM:SS

```
$created_at = date("Y-m-d H:i:s");
echo $created_at;
```

Computers store a date and time as a UNIX timestamp or a timestamp in short. A timestamp is an integer that refers to the number of seconds between 1970-01-01 00:00:00 UTC (Epoch) and the date and time to be stored. Computers store dates and times as timestamps because it is easier to manipulate an integer. For example, to add one day to a timestamp, it simply adds the number of seconds to the timestamp. PHP provides some helpful functions that manipulate timestamps effectively.

To get the current time, you use the time() function:

```
function time(): int
```

The time() function returns the current UNIX timestamp since Epoch (January 1 1970 00:00:00 GMT).

```
echo time(); // Time in timestamp
```

The return value is a big integer that represents the number of seconds since Epoch. To make the time human-readable, you use the date() function.

```
$current_time = time();
echo date('Y-m-d g:ia', $current_time) . '<br>';
```

The date() function has two parameters.

- ➔ The first parameter specifies the date and time format. Here's a complete list of valid date and time formats.
- ➔ The second parameter is an integer that specifies the timestamp.

Since the time() function returns a timestamp, you can add seconds to it.

Example: how to add a week to the current time:

```
$current_time = time();
// 7 days later
```

```
$one_week_later =  $current_time + 7 * 24 * 60 * 60;
echo date('Y-m-d g:ia',$one_week_later);

$current_time = time();
// 1 day ago
$yesterday = $current_time -  24 * 60 * 60;
echo date('Y-m-d g:ia',$yesterday);
```

By default, the time() function returns the current time in the timezone specified in the PHP configuration file (php.ini). To get the current timezone, you can use the date_default_timezone_get() function:

```
echo date_default_timezone_get();
```

To set a specific timezone, you use the date_default_timezone_set(). It's recommended that you use the UTC timezone. The following shows how to use the date_default_timezone_set() function to set the current timezone to the UTC timezone:

```
date_default_timezone_set('UTC');
```

Making a Unix timestamp

To make a Unix timestamp, you use the mktime() function:

```
mktime(
    int $hour,
    int|null $minute = null,
    int|null $second = null,
    int|null $month = null,
    int|null $day = null,
    int|null $year = null
): int|false
```

The mktime() function returns a Unix timestamp based on its arguments. If you omit an argument, the mktime() function will use the current value according to the local date and time instead. The following example shows how to use the mktime() function to show that July 13, 2020, is on a Tuesday:

```
echo 'July 25, 2023 is on a ' . date('l', mktime(0, 0, 0, 7, 25, 2023));
```

- Use the time() function to return the current timestamp since Epoch in the local timezone.
- Use the date_default_timezone_set() function to set a specific timezone.
- Use the date() function to format the timestamp.
- Use mktime() function to create a timestamp based on the year, month, day, hour, minute, and second.

Example:

```
<?php
     echo "Today is " . date("Y/m/d") . "<br>";
     echo "Today is " . date("Y.m.d") . "<br>";
     echo "The time is " . date("h:i:s a");
?>
```

Format Characters: To format a date, you use the following date format parameters.

format character	**Description**	**Example returned values**
Day	—	—
d	2 digits with leading zeros that represent the day of the month	01 to 31
D	Three letters that present the day name	Mon through Sun
j	Day of the month without leading zeros	1 to 31
l (lowercase 'L')	Full name of the day of the week's	Sunday through Saturday
N	Day of the week in number according to ISO-8601	1 (for Monday) through 7 (for Sunday)
S	2 characters that represent the ordinal suffix for the day of the month in English	st, nd, rd or th
w	A numeric day of the week	0 (for Sunday) through 6 (for Saturday)
z	A numeric day of the year, starting from 0	0 through 365
Week	—	—
W	Week number of the year in ISO-8601, weeks starting on Monday	Example: 3 (the 3rd week in the year)
Month	—	—

F	The full month name	January through December
m	The month number with leading zeros	01 through 12
M	Three characters that represent the month name	Jan through Dec
n	The month number without leading zero	1 through 12
t	The Number of days in a month	28 through 31
Year	—	—
L	Return 1 if it's a leap year and zero otherwise	
o	ISO-8601 week-numbering year. This has the same value as Y, except that if the ISO week number (W) belongs to the previous or next year, that year is used instead.	Examples: 1999 or 2003
Y	A four-digit represents a year number	Examples: 2020 or 2021
y	A two-digit representation of a year	Examples: 99 or 03
Time	—	—
a	Lowercase am or pm	am or pm
A	Uppercase AM or PM	AM or PM
B	Swatch Internet time	000 through 999
g	12-hour format of an hour without leading zeros	1 through 12
G	24-hour format of an hour without leading zeros	0 through 23
h	12-hour format of an hour with leading zeros	01 through 12
H	24-hour format of an hour with leading zeros	00 through 23
i	Minutes with leading zeros	00 to 59

s	Seconds with leading zeros	00 through 59
u	Microseconds.	Example: 654321
v	Milliseconds.	Example: 654
Timezone	—	—
e	Timezone identifier	Examples: UTC, GMT, Europe/Berlin
I (capital i)	Daylight Saving	1 if Daylight Saving Time, 0 otherwise.
O	Difference to GMT without colon between hours and minutes	Example: +0200
P	Difference to GMT with a colon between hours and minutes	Example: +02:00
p	The same as P, but returns Z instead of +00:00	Example: +02:00
T	Timezone abbreviation	Examples: CEST, MDT …
Z	Timezone offset in seconds.	-43200 through 50400
Full Date/Time	—	—
c	ISO 8601 date	2021-07-14T13:38:04+02:00
r	RFC 2822 formatted date	Wed, 14 Jul 2021 13:38:20 +0200
U	Seconds since January 1 1970 00:00:00 GMT (the Unix Epoch)	

PHP Timezone:

Name	Description	Default	PHP Version
date.timezone	The default timezone (used by all date/time functions)	""	PHP 5.1
date.default_latitude	The default latitude (used by date_sunrise() and date_sunset())	"31.7667"	PHP 5.0
date.default_longitude	The default longitude (used by date_sunrise() and date_sunset())	"35.2333"	PHP 5.0

date.sunrise_zenith	The default sunrise zenith (used by date_sunrise() and date_sunset())	"90.83"	PHP 5.0
date.sunset_zenith	The default sunset zenith (used by date_sunrise() and date_sunset())	"90.83"	PHP 5.0

PHP Date/Time Functions:

Function	Description
checkdate()	Validates a Gregorian date
date_add()	Adds days, months, years, hours, minutes, and seconds to a date
date_create_from_format()	Returns a new DateTime object formatted according to a specified format
date_create()	Returns a new DateTime object
date_date_set()	Sets a new date
date_default_timezone_get()	Returns the default timezone used by all date/time functions
date_default_timezone_set()	Sets the default timezone used by all date/time functions
date_diff()	Returns the difference between two dates
date_format()	Returns a date formatted according to a specified format
date_get_last_errors()	Returns the warnings/errors found in a date string
date_interval_create_from_date_string()	Sets up a DateInterval from the relative parts of the string
date_interval_format()	Formats the interval
date_isodate_set()	Sets the ISO date
date_modify()	Modifies the timestamp
date_offset_get()	Returns the timezone offset

date_parse_from_format()	Returns an associative array with detailed info about a specified date, according to a specified format
date_parse()	Returns an associative array with detailed info about a specified date
date_sub()	Subtracts days, months, years, hours, minutes, and seconds from a date
date_sun_info()	Returns an array containing info about sunset/sunrise and twilight begin/end, for a specified day and location
date_sunrise()	Returns the sunrise time for a specified day and location
date_sunset()	Returns the sunset time for a specified day and location
date_time_set()	Sets the time
date_timestamp_get()	Returns the Unix timestamp
date_timestamp_set()	Sets the date and time based on a Unix timestamp
date_timezone_get()	Returns the time zone of the given DateTime object
date_timezone_set()	Sets the time zone for the DateTime object
date()	Formats a local date and time
getdate()	Returns date/time information of a timestamp or the current local date/time
gettimeofday()	Returns the current time
gmdate()	Formats a GMT/UTC date and time
gmmktime()	Returns the Unix timestamp for a GMT date
gmstrftime()	Formats a GMT/UTC date and time according to locale settings
idate()	Formats a local time/date as integer
localtime()	Returns the local time
microtime()	Returns the current Unix timestamp with microseconds
mktime()	Returns the Unix timestamp for a date
strftime()	Formats a local time and/or date according to locale

	settings
strptime()	Parses a time/date generated with strftime()
strtotime()	Parses an English textual datetime into a Unix timestamp
time()	Returns the current time as a Unix timestamp
timezone_abbreviations_list()	Returns an associative array containing dst, offset, and the timezone name
timezone_identifiers_list()	Returns an indexed array with all timezone identifiers
timezone_location_get()	Returns location information for a specified timezone
timezone_name_from_ abbr()	Returns the timezone name from abbreviation
timezone_name_get()	Returns the name of the timezone
timezone_offset_get()	Returns the timezone offset from GMT
timezone_open()	Creates new DateTimeZone object
timezone_transitions_get()	Returns all transitions for the timezone
timezone_version_get()	Returns the version of the timezonedb

Predefined Date/Time Constants

Constant	Description
DATE_ATOM	Atom (example: 2019-01-18T14:13:03+00:00)
DATE_COOKIE	HTTP Cookies (example: Fri, 18 Jan 2019 14:13:03 UTC)
DATE_ISO8601	ISO-8601 (example: 2019-01-18T14:13:03+0000)
DATE_RFC822	RFC 822 (example: Fri, 18 Jan 2019 14:13:03 +0000)
DATE_RFC850	RFC 850 (example: Friday, 18-Jan-19 14:13:03 UTC)
DATE_RFC1036	RFC 1036 (example: Friday, 18-Jan-19 14:13:03 +0000)
DATE_RFC1123	RFC 1123 (example: Fri, 18 Jan 2019 14:13:03 +0000)
DATE_RFC2822	RFC 2822 (example: Fri, 18 Jan 2019 14:13:03 +0000)

DATE_RFC3339	Same as DATE_ATOM (since PHP 5.1.3)
DATE_RFC3339_EXTENDED	RFC3339 Extended format (since PHP 7.0.0) (example: 2019-01-18T16:34:01.000+00:00)
DATE_RSS	RSS (Fri, 18 Jan 2019 14:13:03 +0000)
DATE_W3C	World Wide Web Consortium (example: 2019-01-18T14:13:03+00:00)
SUNFUNCS_RET_TIMESTAMP	Timestamp (since PHP 5.1.2)
SUNFUNCS_RET_STRING	Hours:minutes (example: 09:41) (since PHP 5.1.2)
SUNFUNCS_RET_DOUBLE	Hours as a floating point number (example: 9.75) (since PHP 5.1.2)

Strings:

In PHP, strings are a fundamental data type used to store and manipulate text data. They are used extensively in web development for tasks such as displaying text on web pages, processing user input, and communicating with databases.

PHP offers a wide range of functions for working with strings, including functions for concatenation, splitting, trimming, replacing, and formatting strings. String literals in PHP can be enclosed in single or double quotes, and special characters such as newlines and tabs can be escaped using a backslash and they can also be concatenated using the dot (.) operator. A string is a sequence of characters, such as words, phrases, or even numbers, that can be manipulated in various ways using PHP functions.

In PHP, there are a few ways to specify a string literal. The most common way is to use single or double quotes to enclose the string. Single quotes are used when no variable substitution or escape sequence is needed within the string, while double quotes allow for variable substitution and escape sequences. Another way to specify a string is by using the backtick symbol to enclose the string, which allows for command execution within the string. PHP also supports *heredoc* and *nowdoc* syntax for multi-line string literals.

Single Quoted:

In PHP, single quotes can be used to specify a string literal. To use single quotes, simply enclose the string with single quotes as shown in the example below:

```
$string = 'This is a string literal';
```

Single quotes are useful when you don't need variable substitution or escape sequences within the string.

```
$string = 'This is a single quote: \'';
```

Double Quoted:

In PHP, double quotes can be used to specify a string literal that allows for variable substitution and escape sequences.

```
$name = "STUDENT";
$message = "Hello, $name! \nWelcome to our Lab.";
echo $message;
```

Heredoc Syntax:

Heredoc syntax is a way to specify a multi-line string literal in PHP. It allows for the inclusion of variables and special characters within the string without having to use concatenation or escape sequences.

```
$name = 'SURESH';
```

```
$age = 30;

$string = <<<EOT
Hello, my name is $name.
I am $age years old.
EOT;

echo $string;
```

In this example, the $name and $age variables are included within the string using variable interpolation. The string starts with the <<<EOT operator followed by the identifier EOT. The string itself spans two lines and ends with EOT; on a separate line. Finally, the string is printed to the screen using echo.

Nowdoc Syntax:

In PHP, the nowdoc syntax can be used to specify a string literal. It is a variant of the heredoc syntax and allows for the easy creation of multi-line string literals.

```
$str = <<<'EOD'

   This is a nowdoc string

   It behaves like a single-quoted string

   No variable substitution or escape sequences are parsed

   EOD;
```

The opening <<<'EOD' tag is followed by an identifier, which can be any string of characters except for numbers and quotes. This identifier marks the end of the string, denoted by a semicolon (;) followed by the identifier.

String function()

Function	Description
addcslashes()	Returns a string with backslashes in front of the specified characters
addslashes()	Returns a string with backslashes in front of predefined characters
bin2hex()	Converts a string of ASCII characters to hexadecimal values
chop()	Removes whitespace or other characters from the right end of a string

chr()	Returns a character from a specified ASCII value
chunk_split()	Splits a string into a series of smaller parts
convert_cyr_string()	Converts a string from one Cyrillic character-set to another
convert_uudecode()	Decodes a uuencoded string
convert_uuencode()	Encodes a string using the uuencode algorithm
count_chars()	Returns information about characters used in a string
crc32()	Calculates a 32-bit CRC for a string
crypt()	One-way string hashing
echo()	Outputs one or more strings
explode()	Breaks a string into an array
fprintf()	Writes a formatted string to a specified output stream
get_html_translation_table()	Returns the translation table used by htmlspecialchars() and htmlentities()
hebrev()	Converts Hebrew text to visual text
hebrevc()	Converts Hebrew text to visual text and new lines (\n) into
hex2bin()	Converts a string of hexadecimal values to ASCII characters
html_entity_decode()	Converts HTML entities to characters
htmlentities()	Converts characters to HTML entities
htmlspecialchars_decode()	Converts some predefined HTML entities to characters
htmlspecialchars()	Converts some predefined characters to HTML entities
implode()	Returns a string from the elements of an array
join()	Alias of implode()
lcfirst()	Converts the first character of a string to lowercase
levenshtein()	Returns the Levenshtein distance between two strings
localeconv()	Returns locale numeric and monetary formatting information

ltrim()	Removes whitespace or other characters from the left side of a string
md5()	Calculates the MD5 hash of a string
md5_file()	Calculates the MD5 hash of a file
metaphone()	Calculates the metaphone key of a string
money_format()	Returns a string formatted as a currency string
nl_langinfo()	Returns specific local information
nl2br()	Inserts HTML line breaks in front of each newline in a string
number_format()	Formats a number with grouped thousands
ord()	Returns the ASCII value of the first character of a string
parse_str()	Parses a query string into variables
print()	Outputs one or more strings
printf()	Outputs a formatted string
quoted_printable_decode()	Converts a quoted-printable string to an 8-bit string
quoted_printable_encode()	Converts an 8-bit string to a quoted printable string
quotemeta()	Quotes metacharacters
rtrim()	Removes whitespace or other characters from the right side of a string
setlocale()	Sets locale information
sha1()	Calculates the SHA-1 hash of a string
sha1_file()	Calculates the SHA-1 hash of a file
similar_text()	Calculates the similarity between two strings
soundex()	Calculates the soundex key of a string
sprintf()	Writes a formatted string to a variable
sscanf()	Parses input from a string according to a format

str_getcsv()	Parses a CSV string into an array
str_ireplace()	Replaces some characters in a string (case-insensitive)
str_pad()	Pads a string to a new length
str_repeat()	Repeats a string a specified number of times
str_replace()	Replaces some characters in a string (case-sensitive)
str_rot13()	Performs the ROT13 encoding on a string
str_shuffle()	Randomly shuffles all characters in a string
str_split()	Splits a string into an array
str_word_count()	Count the number of words in a string
strcasecmp()	Compares two strings (case-insensitive)
strchr()	Finds the first occurrence of a string inside another string (alias of strstr())
strcmp()	Compares two strings (case-sensitive)
strcoll()	Compares two strings (locale based string comparison)
strcspn()	Returns the number of characters found in a string before any part of some specified characters are found
strip_tags()	Strips HTML and PHP tags from a string
stripcslashes()	Unquotes a string quoted with addcslashes()
stripslashes()	Unquotes a string quoted with addslashes()
stripos()	Returns the position of the first occurrence of a string inside another string (case-insensitive)
stristr()	Finds the first occurrence of a string inside another string (case-insensitive)
strlen()	Returns the length of a string
strnatcasecmp()	Compares two strings using a "natural order" algorithm (case-insensitive)
strnatcmp()	Compares two strings using a "natural order" algorithm (case-sensitive)

strncasecmp()	String comparison of the first n characters (case-insensitive)
strncmp()	String comparison of the first n characters (case-sensitive)
strpbrk()	Searches a string for any of a set of characters
strpos()	Returns the position of the first occurrence of a string inside another string (case-sensitive)
strrchr()	Finds the last occurrence of a string inside another string
strrev()	Reverses a string
strripos()	Finds the position of the last occurrence of a string inside another string (case-insensitive)
strrpos()	Finds the position of the last occurrence of a string inside another string (case-sensitive)
strspn()	Returns the number of characters found in a string that contains only characters from a specified charlist
strstr()	Finds the first occurrence of a string inside another string (case-sensitive)
strtok()	Splits a string into smaller strings
strtolower()	Converts a string to lowercase letters
strtoupper()	Converts a string to uppercase letters
strtr()	Translates certain characters in a string
substr()	Returns a part of a string
substr_compare()	Compares two strings from a specified start position (binary safe and optionally case-sensitive)
substr_count()	Counts the number of times a substring occurs in a string
substr_replace()	Replaces a part of a string with another string
trim()	Removes whitespace or other characters from both sides of a string
ucfirst()	Converts the first character of a string to uppercase
ucwords()	Converts the first character of each word in a string to uppercase

vfprintf()	Writes a formatted string to a specified output stream
vprintf()	Outputs a formatted string
vsprintf()	Writes a formatted string to a variable
wordwrap()	Wraps a string to a given number of characters

Exercise:

1. Create a php script which passes two values and the choice of operation as parameters of a function . Depending on the choice function should perform the arithmetic operation on values.
2. Create a php script which passes the array into function as a parameter. The function then counts the total number of odd, even and prime numbers.
3. Create a php script which creates a function having choice and string as parameter and perform the following task on string based on choice. The function should return the answer.
    - Reverse the string.
    - Count the length of the string
    - Display the first word of the string.
    - Display the last five words from the string.
    - Display characters from position 5 to 15 from the string
4. Create a php script which demonstrates date functions mentioned below:

    - checkdate()
    - date_add()
    - date_create_from_format()
    - date_create()
    - date_default_timezone_get()
    - date_default_timezone_set()
    - date_diff()
    - date_format()
    - date_modify()
    - date_time_set()
    - date_timestamp_get()
    - date_timestamp_set()
    - date()
    - getdate()
    - gettimeofday()
    - mktime()
    - strftime()
    - strptime()
    - strtotime()
    - time()
    - timezone_abbreviations_list()
    - timezone_identifiers_list()

5. Create a php script which demonstrates string function mentioned below:

    - addcslashes()
    - addslashes()
    - bin2hex()
    - chr()
    - chunk_split()
    - count_chars()
    - echo()
    - explode()
    - implode()
    - join()
    - lcfirst()
    - ltrim()
    - md5()
    - number_format()
    - ord()
    - parse_str()
    - print()
    - printf()
    - rtrim()
    - setlocale()
    - sha1()
    - str_pad()
    - str_repeat()
    - str_replace()
    - str_shuffle()
    - str_split()
    - str_word_count()
    - strcasecmp()
    - strchr()
    - strcmp()
    - strip_tags()
    - stripslashes()
    - stripos()
    - stristr()
    - strlen()
    - strnatcasecmp()
    - strnatcmp()
    - strncasecmp()
    - strncmp()
    - strpbrk()
    - strpos()
    - strrchr()
    - strrev()
    - strripos()
    - strrpos()
    - strstr()
    - strtolower()
    - strtoupper()
    - substr()
    - substr_count()
    - substr_replace()
    - trim()

6. Create a PHP script to demonstrate possible ways for variable length arguments. Pass integer and float values to a functions and calculate
    - sum of all arguments
    - product of all arguments
    - count of integer and real number arguments
    - count of lowercase character, uppercase character and special character