

# PHP Arrays

## Create an array in PHP:-

- Arrays in PHP are a type of data structure that allows us to store multiple elements of similar or different data types under a single variable thereby saving us the effort of creating a different variable for every data. The arrays are helpful to create a list of elements of similar types, which can be accessed using their index or key.
- **For Ex.:-**  
  
→ `$bca= array("sem-1","sem-2","sem-3","sem-4");`

# Types of Arrays

- Arrays comes into play and helps us to store every element within a single variable and also allows easy access using an index or a key. An array is created using an **array()** function in PHP.
- **Types of Array in PHP:-**
  - 1) **Indexed array**
  - 2) **Associative array**
  - 3) **Multidimensional array**

**2) Indexed or numeric array:-** PHP index is represented by a number which starts from 0. We can store numbers, strings and objects in the PHP array. All PHP array elements are assigned to an index number by default.

→ These type of arrays can be used to store any type of element, but an index is always a number. By default, the index starts at zero. These arrays can be created in **two** different ways.

→ **For Ex.:- First ex.**

**<? Php**

```
$bca=array("sem1","sem2","sem3","sem4");
```

```
echo "1st array element :- <br>";
```

```
echo $bca[3],"/n";
```

```
echo $bca[0];
```

**?>**

→ For Ex.:- Second ex.

**<? Php**

**\$bca[0]="Sem1";**

**\$bca[1]="Sem2";**

**\$bca[2]="sem3";**

**\$bca[3]="sem4";**

**echo "2<sup>st</sup> array element :- <br>";**

**echo \$bca[3],"/n";**

**echo \$bca[0],"/n";**

**echo \$bca[2];**

**?>**

**2) Associative array:-** An associative array is a type of array in PHP that allows you to assign a key to each element, rather than using an index number.

We can associate names with each array element in PHP using => symbol.

→ These types of arrays are similar to the indexed arrays but instead of linear storage, every value can be assigned with a user-defined key of string type.

→ You can create an associative array using the array() function or [] operator, like this:

**For Ex.:- First ex.**

```
<? Php
```

```
    $bca=array("s1"=>"Sem1",    "s2"=>"Sem2",    "s3"=>  
"Sem3", "s4"=>"Sem4");
```

```
    echo "1st array element :- <br>";
```

```
    echo $bca[s1],"/n";
```

```
    echo $bca[s4];
```

```
?>
```

→ For Ex.:- Second ex.

**<? Php**

**\$bca[s1]="Sem1";**

**\$bca[s1]="Sem2";**

**\$bca[s2]="sem3";**

**\$bca[s3]="sem4";**


**echo "2<sup>st</sup> array element :- <br>";**

**echo \$bca[s3],"/n";**

**echo \$bca[s0],"/n";**

**echo \$bca[s2];**

**?>**



**2) Multidimensional array:-** Multi-dimensional arrays are such arrays that store another array at each index instead of a single element.

- In other words, we can define multi-dimensional arrays as an array of arrays. As the name suggests, every element in this array can be an array and they can also hold other sub-arrays within.
- Arrays or sub-arrays in multidimensional arrays can be accessed using multiple dimensions.

→ For Ex.:- Second ex.

<? Php

```
$favorites = array(
    array("name" => "Dave Punk", "mob" =>
"5689741523", "email" => "davepunk@gmail.com",),
    array("name" => "Monty Smith", "mob" =>
"2584369721", "email" => "montysmith@gmail.com",
    ), array("name" => "John Flinch", "mob" =>
"9875147536", "email" => "johnflinch@gmail.com",
    )
);
echo "Dave Punk email-id is: " . $favorites[0]["email"],
"\n";
echo "John Flinch mobile number is: " .
$favorites[2]["mob"];
?>
```



# Storing Data with array

- The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.
- Storing a value in an array will create the array if it didn't already exist, but trying to retrieve a value from an array that hasn't been defined yet won't create the array.
- ❖ In this chapter, we will go through the following PHP array sort functions:
  - **sort()** - sort arrays in ascending order
  - **rsort()** - sort arrays in descending order
  - **asort()** - sort associative arrays in ascending order, according to the value
  - **ksort()** - sort associative arrays in ascending order, according to the key
  - **arsort()** - sort associative arrays in descending order, according to the value
  - **krsort()** - sort associative arrays in descending order, according to the key

→ For Ex.:- Second ex.

**<? Php**

**<?php**

**\$numbers = array(4, 6, 2, 22, 11);**

**sort(\$numbers);**

**\$var = count(\$numbers);**

**for(\$x = 0; \$x < \$var; \$x++) {**

**echo \$numbers[\$x];**

**echo "<br>";**

**}**

**?>**

**\$name = array("Varun", "Manoj", "Toyota");**

**rsort(\$name);**

# Accessing Array Elements

→ To access an element in a PHP array, you can use the array's index, which is the position of the element within the array. The index can be either a numerical value or a string.

→ For example, to access the first element in a numerically indexed array, you would use the following syntax:

```
$myArray = array("apple", "banana", "orange");  
echo $myArray[0];    // Output: apple
```

→ To access an element in an associative array (where elements are stored as key-value pairs), you would use the key instead of the index:

```
$myAssocArray = array("name" => "John", "age" => 30,  
"city" => "New York");  
echo $myAssocArray["name"];    // Output: John
```

→ You can also use a loop (such as a foreach loop) to iterate through all elements in an array and access them one by one.

# Accessing Array Elements

→ For ex. <?php

```
function myFunction()
```

```
{
```

```
    echo "Welcome to BCA College Idar!";
```

```
}
```

```
$myArr = array("BCA Sem-4", 50, myFunction);
```

```
    $myArr[2]();
```

```
?>
```

# Displaying with Above Array types

→ To access display the array structure and its values, we can use **var\_dump()** and **print\_r()** functions. We will display the array structure using **var\_dump()** function. This function is used to dump information about a variable. This function displays structured information such as the type and value of the given variable.

For ex.

```
$array1 = array('0' => "Python", '1' => "java", '2' => "c/cpp");  
// Display array structure  
var_dump($array1);
```

→ We will display the array values by using the **print\_r()** function. This function is used to print or display information stored in a variable.

```
$array1 = array('0' => "Python", '1' => "java", '2' => "c/cpp");  
// Display array values  
print_r($array1);
```

# Array related functions

PHP has a variety of built-in functions for working with arrays. Some of the most important and commonly used array functions include:

1. **array() or []**: This function creates a new array.
2. **count()**: This function returns the number of elements in an array.
3. **array\_push()**: This function adds one or more elements to the end of an array.
4. **array\_pop()**: This function removes and returns the last element from an array.
5. **array\_shift()**: This function removes and returns the first element from an array.
6. **array\_unshift()**: This function adds one or more elements to the beginning of an array.
7. **array\_slice()**: This function returns a new array that contains a specified range of elements from the original array.

8. **array\_splice()**: This function removes and/or replaces elements in an array, and returns the removed elements.

9. **array\_merge()**: This function merges one or more arrays into a single array.

10. **array\_keys()**: This function returns an array containing all the keys of an array.

11. **array\_values()**: This function returns an array containing all the values of an array.

12. **array\_combine()**: This function creates an array by using one array for keys and another for its values

13. **array\_reverse()**: This function returns an array with elements in reverse order

14. **array\_search()**: This function search for a specified value in an array and returns the key if found

15. **sort()**: This function sorts an array in ascending order, according to the value of each element

16. **rsort()**: This function sorts an array in descending order, according to the value of each element

# PHP Function- User Defined Function

- User-defined functions are created by the programmer to perform specific tasks according to their requirements.
- These functions are declared using the **function** keyword followed by a function name and a block of code encapsulated within curly braces { }.
- They can accept parameters (input data) and return values (output data) using the return statement.
- User-defined functions provide a way to encapsulate reusable code, promote modularity, and enhance code readability and maintainability.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.



# Create a PHP Function

**Syntax:-** Function function\_name()

```
{  
    return return_value;  
}
```

Function BCA()

```
{  
    echo "Hello! BCA!";  
}
```

**Call a function:-**

```
BCA();
```

→ we create a function named BCA().

→ The opening curly brace { indicates the beginning of the function code, and the closing curly brace } indicates the end of the function.

# PHP Function- Adding Parameters

→ For example, you wanted to pass the text data to the display function that the function should display. You can do that by adding a function argument to the argument list. Like this display function.

**Syntax:-** Function function\_name(\$args1,\$args2... \$argsN)

```
{  
    Statement  
    return return_value();  
}
```

**Create function:-**

```
function Display($greeting, $message)  
{  
    echo "$greeting";  
    echo "$message";  
}
```

**Call Function:-** Display("Hello", "BCA Sem-4");

# PHP Function- Return Values

→ A simple function named adder that returns the sum of the two numbers passed to it.

Ex. :- function adder(operand1,operand2)

```
{    $sum= $operand1+operand2;  
    return $sum  
}
```

→ The sum in the variable named \$sum how do you return that value to the calling code.

→ The return keyword ends a function and, optionally, uses the result of an expression as the return value of the function.

→ If return is used outside of a function, it stops PHP code in the file from running. If the file was included using include, include\_once, require or require\_once, the result of the expression is used as the return value of the include statements

# Making Arguments be passed by reference

→ When you pass data to a function, what really passed is a copy of that data. So ex. If you pass variable a copy is made of that variable and that copy is actually passed to the function.

→ What if you wanted to actually pass the real thing to the function ? Say for ex a function to alter the value in a variable?

→ For Ex.:- <? PHP

```
$value=4;
```

```
function squarer($number) { $number *= $number; } ?>
```

→ In case of PHP call by reference, actual value is modified if it is modified inside the function. In such case, you need to use & (ampersand) symbol with formal arguments. The & represents reference of the variable.

```
<?php
```

```
function adder(&$str2) { $str2 .= 'Call By Reference'; }
```

```
$str = 'This is '; adder($str);
```

```
echo $str;    ?>
```

# Default Arguments Values

→ What happens if you have a function named display that takes two arguments like this.

Ex.    Function display(\$greeting, \$message)  
         {    echo \$greeting; echo \$message;    }

Call one argument:- display(“No worries”);

→ You can fix this problem by supplying a default argument here the way it works you add the default argument in the argument list, using equal sign.

Ex.    Function display(\$greeting, \$message=“BCA sem-4”)  
         {  
                echo \$greeting;  
                echo \$message;  
         }

→ PHP allows you to define C++ style default argument values. In such case, if you don't pass any value to the function, it will use default argument value.

# Recursive Function

→ PHP also supports recursive function call like C/C++. In such case, we call current function within function. It is also known as recursion.

→ It is recommended to avoid recursive function call over 200 recursion level because it may smash the stack and may cause the termination of script.

For Ex.:- <?php

```
function display($number)
{
    if($number<=5)
    {
        echo "$number <br/>";
        display($number+1);
    }
}
display(1);
?>
```

# Miscellaneous Function: Define

- The misc. functions were only placed here because none of the other categories seemed to fit.
- Miscellaneous functions perform a variety of operations and return specific information or values. The miscellaneous functions are summarized in the table below. Click the function name to jump to a discussion of that function.
- consisting of a mixture of various things that are not related to each other: The magazine did a price comparison of 17 miscellaneous items sold at both stores. miscellaneous charges/costs/expenses The job helps me pay for food and miscellaneous expenses.

# Define Function

→ The define() function is basically used by programmers to create constant. Constants in PHP are very similar to variables and the only difference between both are the values of constants can not be changed once it is set in a program.

→ define() returns a Boolean value. It will return TRUE on success and FALSE on failure of the expression.

→ **Syntax:-** `define(string $constant, mixed $value, bool $case_insensitive);`

**\$constant:** It is of String type that describes the name of the constant and is a required parameter.

**\$value:** It is of mixed type and it describes the Value of the constant and is required in parameter

**\$case\_insensitive:** It is of a boolean type that describes whether the name of the constant can be case-sensitive or not and it is an optional parameter.



# Define Function

→ For Ex:-

```
<?php
define("GREETINGS", "Hello GFG.", true);
echo GREETINGS;
?>
```

2.

```
<?php
    Define('LANGUAGES', array( 'C', 'C++', 'JAVA',
'PYTHON'));

    echo LANGUAGES[3];
?>
```

# Constant Function

→ Return the value of a constant. The constant() function returns the value of a constant. Required. Specifies the name of the constant to check.

```
<?php
//define a constant
define("GREETING","Hello BCA! How are you today?");

echo constant("GREETING");
?>
```

# Include Function

For Ex:- <?php

```
Include("File2.php");
```

```
echo "A $color $fruit";  
?>
```

2. <?php

```
$color="Blue";  
$fruit="banana";  
?>
```

# Require Function

For Ex:- <?php  
Require("File.php");  
echo "hi";  
echo "A \$color \$fruit";  
demo(40+60);  
?>

2. <?php  
\$color="Blue";  
\$fruit="banana";  
function demo(\$a,\$b)  
{  
    echo (\$a+\$b);  
}  
?>

# Die/ Exit Function

- Print a message and terminate the current script
- The die() function is an alias of the exit() function.

For Ex:- <?php

```
$site = "hello bca sem-4";
```

```
fopen($site,"r")
```

```
or die("Unable to connect to $site");
```

```
?>
```

```
<?php
```

```
$site = "hello bca sem-4";
```

```
fopen($site,"r")
```

```
or exit("Unable to connect to $site");
```

```
?>
```

# String Function

| Function  | Description   |
|---|---|
| <a href="#"><u>addslashes()</u></a>                 | Returns a string with backslashes in front of the specified characters      |
| <a href="#"><u>addslashes()</u></a>                 | Returns a string with backslashes in front of predefined characters         |
| <a href="#"><u>bin2hex()</u></a>                    | Converts a string of ASCII characters to hexadecimal values                 |
| <a href="#"><u>chop()</u></a>                       | Removes whitespace or other characters from the right end of a string       |
| <a href="#"><u>chr()</u></a>                        | Returns a character from a specified ASCII value                            |
| <a href="#"><u>chunk_split()</u></a>                | Splits a string into a series of smaller parts                              |
| <a href="#"><u>convert_cyr_string()</u></a>         | Converts a string from one Cyrillic character-set to another                |
| <a href="#"><u>convert_uuencode()</u></a>           | Decodes a uuencoded string  |
| <a href="#"><u>convert_uuencode()</u></a>           | Encodes a string using the uuencode algorithm                               |
| <a href="#"><u>count_chars()</u></a>                | Returns information about characters used in a string                       |
| <a href="#"><u>crc32()</u></a>                      | Calculates a 32-bit CRC for a string  |
| <a href="#"><u>crypt()</u></a>                      | One-way string hashing  |
| <a href="#"><u>echo()</u></a>                       | Outputs one or more strings   |
| <a href="#"><u>explode()</u></a>                    | Breaks a string into an array   |
| <a href="#"><u>fprintf()</u></a>                    | Writes a formatted string to a specified output stream                      |
| <a href="#"><u>get_html_translation_table()</u></a> | Returns the translation table used by htmlspecialchars() and htmlentities() |
| <a href="#"><u>hebrew()</u></a>                     | Converts Hebrew text to visual text   |

# String Function

|  |  |
|--|--|
| <code>print()</code>                   | Outputs one or more strings  |
| <code>printf()</code>                  | Outputs a formatted string   |
| <code>quoted_printable_decode()</code> | Converts a quoted-printable string to an 8-bit string                  |
| <code>quoted_printable_encode()</code> | Converts an 8-bit string to a quoted printable string                  |
| <code>quotemeta()</code>               | Quotes meta characters   |
| <code>rtrim()</code>                   | Removes whitespace or other characters from the right side of a string |
| <code>setlocale()</code>               | Sets locale information  |
| <code>sha1()</code>                    | Calculates the SHA-1 hash of a string                                  |
| <code>sha1_file()</code>               | Calculates the SHA-1 hash of a file                                    |
| <code>similar_text()</code>            | Calculates the similarity between two strings                          |
| <code>soundex()</code>                 | Calculates the soundex key of a string                                 |
| <code>sprintf()</code>                 | Writes a formatted string to a variable                                |
| <code>sscanf()</code>                  | Parses input from a string according to a format                       |
| <code>str_getcsv()</code>              | Parses a CSV string into an array                                      |
| <code>str_ireplace()</code>            | Replaces some characters in a string (case-insensitive)                |
| <code>str_pad()</code>                 | Pads a string to a new length  |
| <code>str_repeat()</code>              | Repeats a string a specified number of times                           |
| <code>str_replace()</code>             | Replaces some characters in a string (case-sensitive)                  |



# Math Function

| Function              | Description  |
|-----------------------|--|
| <u>abs()</u>          | Returns the absolute (positive) value of a number  |
| <u>acos()</u>         | Returns the arc cosine of a number                 |
| <u>acosh()</u>        | Returns the inverse hyperbolic cosine of a number  |
| <u>asin()</u>         | Returns the arc sine of a number                   |
| <u>asinh()</u>        | Returns the inverse hyperbolic sine of a number    |
| <u>atan()</u>         | Returns the arc tangent of a number in radians     |
| <u>atan2()</u>        | Returns the arc tangent of two variables x and y   |
| <u>atanh()</u>        | Returns the inverse hyperbolic tangent of a number |
| <u>base_convert()</u> | Converts a number from one number base to another  |
| <u>bindec()</u>       | Converts a binary number to a decimal number       |
| <u>ceil()</u>         | Rounds a number up to the nearest integer          |
| <u>cos()</u>          | Returns the cosine of a number                     |
| <u>cosh()</u>         | Returns the hyperbolic cosine of a number          |
| <u>decbin()</u>       | Converts a decimal number to a binary number       |
| <u>dechex()</u>       | Converts a decimal number to a hexadecimal number  |
| <u>decoct()</u>       | Converts a decimal number to an octal number       |
| <u>deg2rad()</u>      | Converts a degree value to a radian value          |
| <u>exp()</u>          | Calculates the exponent of e                       |



# Math Function

|                        |   |
|------------------------|---|
| <u>log10()</u>         | Returns the base-10 logarithm of a number   |
| <u>log1p()</u>         | Returns $\log(1+\text{number})$   |
| <u>max()</u>           | Returns the highest value in an array, or the highest value of several specified values |
| <u>min()</u>           | Returns the lowest value in an array, or the lowest value of several specified values   |
| <u>mt_getrandmax()</u> | Returns the largest possible value returned by <code>mt_rand()</code>                   |
| <u>mt_rand()</u>       | Generates a random integer using Mersenne Twister algorithm                             |
| <u>mt_srand()</u>      | Seeds the Mersenne Twister random number generator                                      |
| <u>octdec()</u>        | Converts an octal number to a decimal number  |
| <u>pi()</u>            | Returns the value of PI   |
| <u>pow()</u>           | Returns x raised to the power of y  |
| <u>rad2deg()</u>       | Converts a radian value to a degree value   |
| <u>rand()</u>          | Generates a random integer  |
| <u>round()</u>         | Rounds a floating-point number  |
| <u>sin()</u>           | Returns the sine of a number  |
| <u>sinh()</u>          | Returns the hyperbolic sine of a number   |
| <u>sqrt()</u>          | Returns the square root of a number   |
| <u>srand()</u>         | Seeds the random number generator   |
| <u>tan()</u>           | Returns the tangent of a number   |
| <u>tanh()</u>          | Returns the hyperbolic tangent of a number  |

# Global Variable (Super global)

→ Global variables are variables that can be accessed from any scope.

→ Variables of the outer most scope are automatically global variables, and can be used by any scope, e.g. inside a function.

To use a global variable inside a function you have to either define them as global with the global keyword, or refer to them by using the **\$GLOBALS** syntax.

```
→ <?php  $x = 75;      function myfunction()  
        { global $x;  
          echo $x;  
        } myfunction();  
?>
```

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

| Element/Code                                  | Description  |
|---|--|
| <code>\$_SERVER['PHP_SELF']</code>            | Returns the filename of the currently executing script   |
| <code>\$_SERVER['GATEWAY_INTERFACE']</code>   | Returns the version of the Common Gateway Interface (CGI) the server is using                      |
| <code>\$_SERVER['SERVER_ADDR']</code>         | Returns the IP address of the host server  |
| <code>\$_SERVER['SERVER_NAME']</code>         | Returns the name of the host server (such as www.w3schools.com)                                    |
| <code>\$_SERVER['SERVER_SOFTWARE']</code>     | Returns the server identification string (such as Apache/2.2.24)                                   |
| <code>\$_SERVER['SERVER_PROTOCOL']</code>     | Returns the name and revision of the information protocol (such as HTTP/1.1)                       |
| <code>\$_SERVER['REQUEST_METHOD']</code>      | Returns the request method used to access the page (such as POST)                                  |
| <code>\$_SERVER['REQUEST_TIME']</code>        | Returns the timestamp of the start of the request (such as 1377687496)                             |
| <code>\$_SERVER['QUERY_STRING']</code>        | Returns the query string if the page is accessed via a query string                                |
| <code>\$_SERVER['HTTP_ACCEPT']</code>         | Returns the Accept header from the current request   |
| <code>\$_SERVER['HTTP_ACCEPT_CHARSET']</code> | Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)              |
| <code>\$_SERVER['HTTP_HOST']</code>           | Returns the Host header from the current request   |
| <code>\$_SERVER['HTTP_REFERER']</code>        | Returns the complete URL of the current page (not reliable because not all user-agents support it) |

# Global Variable

```
Ex. echo $_SERVER['PHP_SELF'];  
    echo $_SERVER['SERVER_NAME'];  
    echo $_SERVER['HTTP_HOST'];  
    echo $_SERVER['HTTP_REFERER'];  
    echo $_SERVER['HTTP_USER_AGENT'];  
    echo $_SERVER['SCRIPT_NAME'];
```

→ **\$\_Request:-** \$\_REQUEST is a PHP super global variable which contains submitted form data, and all cookie data.

In other words, \$\_REQUEST is an array containing data from \$\_GET, \$\_POST and \$\_COOKIE. You can access this data with the \$\_REQUEST keyword followed by the name of the form field, or cookie, like this.

→ **\$\_Post:-** \$\_POST contains an array of variables received via the HTTP POST method. There are two main ways to send variables via the HTTP Post method:

→ HTML forms. JavaScript HTTP requests.

# Global Variable

**\$\_Get:-** \$\_GET contains an array of variables received via the HTTP GET method. There are two main ways to send variables via the HTTP GET method. Query strings in the URL  
HTML Forms.

**Ex. First ex. Request:-**

```
<html> <body> <form method="post" action="<?php echo  
$_SERVER['PHP_SELF'];?>"> Name: <input type="text"  
name="fname"> <input type="submit"> </form> <?php if  
($_SERVER["REQUEST_METHOD"] == "POST") { $name =  
htmlspecialchars($_REQUEST['fname']); if (empty($name)) {  
echo "Name is empty"; } else { echo $name; } } ?> </body>  
</html>
```



# Global Variable

## Ex. of Post:-

```
<html> <body> <form method="POST" action="<?php echo  
$_SERVER['PHP_SELF'];?>"> Name: <input type="text"  
name="fname"> <input type="submit"> </form> <?php if  
($_SERVER["REQUEST_METHOD"] == "POST") { $name =  
htmlspecialchars($_POST['fname']); if (empty($name)) { echo  
"Name is empty"; } else { echo $name; } } ?> </body> </html>
```

## Ex. Of Get:-

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

# Working with form in PHP

- A form in PHP is used to collect input from users through different types of input fields, such as text fields, radio buttons, checkboxes, and drop-down menus.
- The data collected through the form can be processed and used to perform various actions on the server side, such as inserting the data into a database, sending an email, or updating a file.
- When creating a form in PHP, it's important to use the appropriate type of input field for the type of data you're collecting. For example, you should use a text field for short, single-line text inputs, a text area for longer, multi-line text inputs, and a checkbox or radio button for multiple choice options.

# Working with form in PHP

The form element has two attributes:

**action:** This attribute specifies the URL of the page that will process the form data. In this example, the form data will be sent to a page called **process.php**

**method:** This attribute specifies the HTTP method used to submit the form data. In this example, the form data will be sent using the GET method.

**For Ex.:-** `<html><title>Php</title>`

`<body> <form action="/bca.php" method="get">`

`Name:-<input type="text" name="fname"`  
`placeholder="Name"><br>`

`city:- <input type="text" name="city"`  
`placeholder="City"><br>`

`<button>Submit</button>`

`</form> </body>`

`</html>`