

# Open Source Software



# PHP Forms

1. PHP Global Variables - Superglobals
2. PHP Regular Expressions
3. PHP Form Handling
4. PHP Form Validation
5. PHP Forms - Required Fields



# 1

## PHP Global Variables - Superglobals

# PHP Global Variables - Superglobals



Some predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP superglobal variables are:

- `$GLOBALS`
- `$_SERVER`
- `$_REQUEST`
- `$_POST`
- `$_GET`
- `$_FILES`
- `$_ENV`
- `$_COOKIE`
- `$_SESSION`

**\$GLOBALS** is an array that contains all global variables.

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 $GLOBALS['x'];
    echo $x;
}

myfunction();
?>
```

75

**\$GLOBALS** is an array that contains all global variables.

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 $GLOBALS['x'];
    echo $x;
}

myfunction();
?>
```

75



**\$\_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**:

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

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 <code>www.w3schools.com</code> )
<code>\$_SERVER['SERVER_SOFTWARE']</code>	Returns the server identification string (such as <code>Apache/2.2.24</code> )
<code>\$_SERVER['SERVER_PROTOCOL']</code>	Returns the name and revision of the information protocol (such as <code>HTTP/1.1</code> )
<code>\$_SERVER['REQUEST_METHOD']</code>	Returns the request method used to access the page (such as <code>POST</code> )



Element/Code	Description
<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)

Element/Code	Description
<code>\$_SERVER['HTTPS']</code>	Is the script queried through a secure HTTP protocol
<code>\$_SERVER['REMOTE_ADDR']</code>	Returns the IP address from where the user is viewing the current page
<code>\$_SERVER['REMOTE_HOST']</code>	Returns the Host name from where the user is viewing the current page
<code>\$_SERVER['REMOTE_PORT']</code>	Returns the port being used on the user's machine to communicate with the web server
<code>\$_SERVER['SCRIPT_FILENAME']</code>	Returns the absolute pathname of the currently executing script
<code>\$_SERVER['SCRIPT_NAME']</code>	Returns the path of the current script
<code>\$_SERVER['SCRIPT_URI']</code>	Returns the URI of the current page

Element/Code	Description
<code>\$_SERVER['SERVER_ADMIN']</code>	Returns the value given to the <code>SERVER_ADMIN</code> directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as <code>someone@w3schools.com</code> )
<code>\$_SERVER['SERVER_PORT']</code>	Returns the port on the server machine being used by the web server for communication (such as 80)
<code>\$_SERVER['SERVER_SIGNATURE']</code>	Returns the server version and virtual host name which are added to server-generated pages
<code>\$_SERVER['PATH_TRANSLATED']</code>	Returns the file system based path to the current script

**\$\_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**.

```
$_REQUEST[ '<Field Name>' ]
```

# Using \$\_REQUEST on \$\_POST Requests

---



POST request are usually data submitted from an HTML form.

# Using \$\_REQUEST on \$\_GET Requests



GET request can be form submissions as in the example above, with the **method** attribute of the HTML **<form>** element set to **GET**. GET requests can also be data from a query string (information added after a URL address).



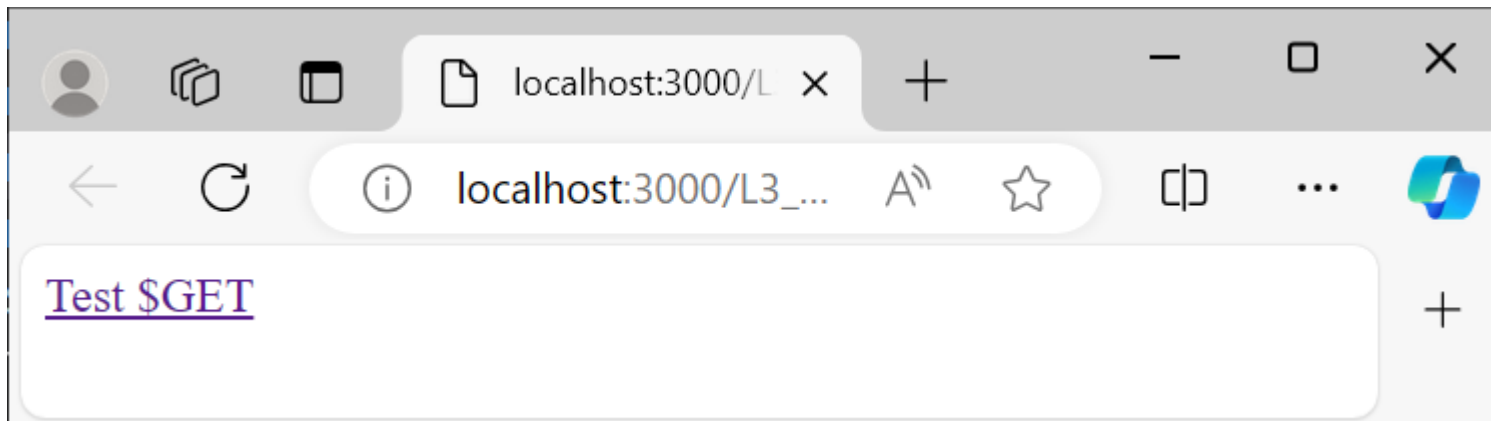
# Using \$\_REQUEST on \$\_GET Requests



```
<!DOCTYPE html>
<html>
<body>

<a href="TestRequestGet.php?subject=PHP&web=https://www.javatpoint.com/form-validation-in-php">Test $GET</a>

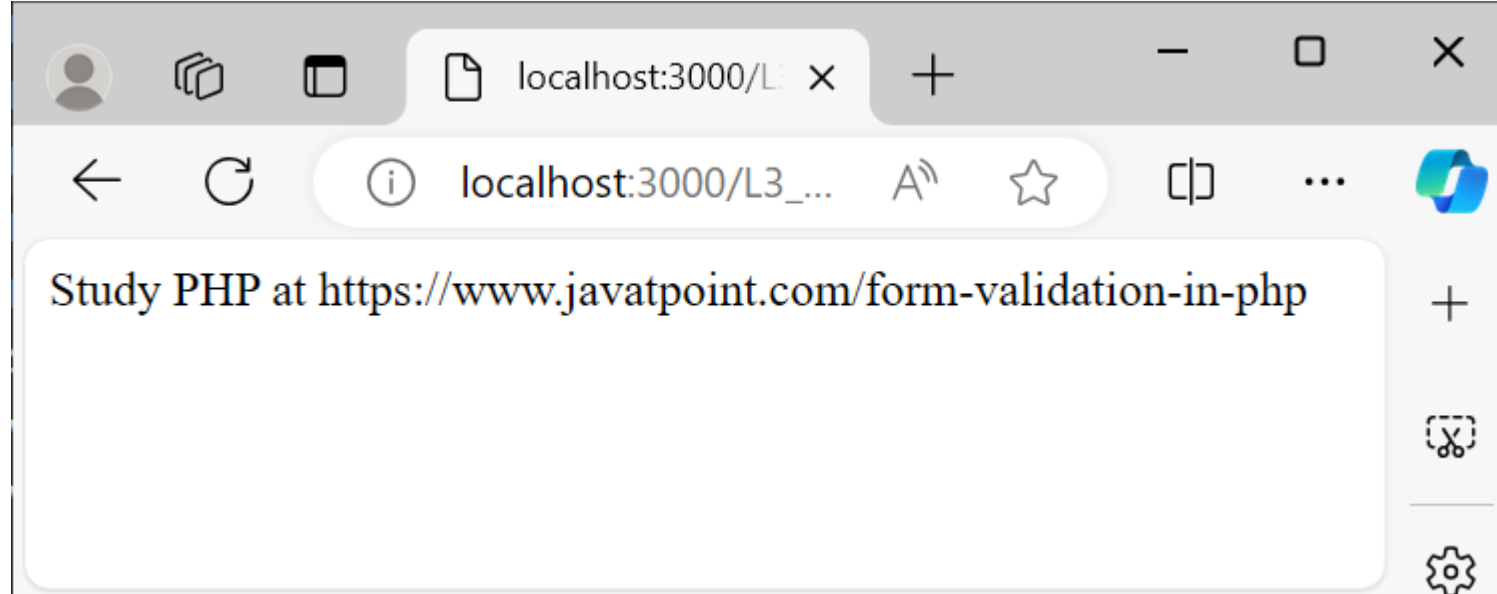
</body>
</html>
```



# Using \$\_REQUEST on \$\_GET Requests



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



# 2 PHP Regular Expressions

# What is a Regular Expression?

---



A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of text search and text replace operations.

In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers.

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

```
$exp = "/<pattern>/i";
```

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

# Regular Expression Functions



PHP provides a variety of functions that allow you to use regular expressions.

Function	Description
<code>preg_match()</code>	Returns 1 if the pattern was found in the string and 0 if not
<code>preg_match_all()</code>	Returns the number of times the pattern was found in the string, which may also be 0
<code>preg_replace()</code>	Returns a new string where matched patterns have been replaced with another string



# Using preg\_match()



The `preg_match()` function will tell you whether a string contains matches of a pattern.

```
<?php
$str = "Visit Hanoi city";
$pattern = "/Hanoi/i";
echo preg_match($pattern, $str);
?>
```

1

*Use a regular expression to do a case-insensitive search for "Hanoi" in a string:*

# Using preg\_match\_all()



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

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

4

*Use a regular expression to do a case-insensitive count of the number of occurrences of "ain" in a string*

# Using preg\_replace()



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

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

Visit Hanoi!

# Using preg\_replace()



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

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

Visit Hanoi!

# Regular Expression Modifiers



Modifiers can change how a search is performed.

Modifier	Description
i	Performs a case-insensitive search
m	Performs a multiline search (patterns that search for a match at the beginning or end of a string will now match the beginning or end of <i>each line</i> )
u	Enables correct matching of UTF-8 encoded patterns

# Regular Expression Patterns



Brackets are used to find a range of characters:

Expression	Description
[abc]	Find one or many of the characters inside the brackets
[^abc]	Find any character NOT between the brackets
[a-z]	Find any character alphabetically between two letters
[A-z]	Find any character alphabetically between a specified upper-case letter and a specified lower-case letter
[A-Z]	Find any character alphabetically between two upper-case letters.
[123]	Find one or many of the digits inside the brackets
[0-5]	Find any digits between the two numbers
[0-9]	Find any digits



Metacharacters are characters with a special meaning:

Metacharacter	Description
	Find a match for any one of the patterns separated by   as in: cat dog fish
.	Find any character
^	Finds a match as the beginning of a string as in: ^Hello
\$	Finds a match at the end of the string as in: World\$
\d	Find any digits
\D	Find any non-digits
\s	Find any whitespace character
\S	Find any non-whitespace character
\w	Find any alphabetical letter (a to Z) and digit (0 to 9)
\W	Find any non-alphabetical and non-digit character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifiers define quantities:

Quantifier	Description
$n^+$	Matches any string that contains at least one $n$
$n^*$	Matches any string that contains zero or more occurrences of $n$
$n?$	Matches any string that contains zero or one occurrences of $n$
$n\{3\}$	Matches any string that contains a sequence of 3 $n$ 's
$n\{2, 5\}$	Matches any string that contains a sequence of at least 2, but not more than 5 $n$ 's
$n\{3, \}$	Matches any string that contains a sequence of at least 3 $n$ 's

# Grouping



You can use parentheses ( ) to apply quantifiers to entire patterns. They also can be used to select parts of the pattern to be used as a match.

```
<?php
$str = "Apples and bananas.";
$pattern = "/ba(na){2}/i";
echo preg_match($pattern, $str);
?>
```

1

# 3 PHP Form Handling

# GET vs. POST



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

Both GET and POST create an array (e.g. `array( key1 => value1, key2 => value2, key3 => value3, ...)`). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

- Both GET and POST are treated as `$_GET` and `$_POST`. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- `$_GET` is an array of variables passed to the current script via the URL parameters.
- `$_POST` is an array of variables passed to the current script via the HTTP POST method.

# When to use GET?

---



Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL).

GET also has limits on the amount of information to send.

The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!



# When to use POST?

---



Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

# PHP - A Simple HTML Form



```
<form action="welcome.php" method="post">  
Name: <input type="text" name="name"><br>  
E-mail: <input type="text" name="email"><br>  
<input type="submit">  
</form>
```

Name:

E-mail:

*When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method. To display the submitted data you could simply echo all the variables.*

# PHP - POST



Name:

E-mail:

```
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
```

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



# PHP - GET



Name:   
E-mail:

```
<form action="welcome_get.php" method="get">  
Name: <input type="text" name="name"><br>  
E-mail: <input type="text" name="email"><br>  
<input type="submit">  
</form>
```

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

← ↻ ⓘ localhost:3000/L3\_FormPHP/welcome\_get.php?name=NguyenAn&email=an%40gmail.com

Welcome NguyenAn  
Your email address is: an@gmail.com

## Novell Services Login


Username:

Password:

City of

Employment:

Web server:

— Choose a server — 

Please specify  
your role:

- ☐ Admin  
☐ Engineer  
☐ Manager  
☐ Guest

Single Sign-on  
to the following:

- ☐ Mail  
☐ Payroll  
☐ Self-service

Login

Reset

```
<div class="mb-3" style="display: flex;">
  <label for="" class="form-label">Single Sign-on <br> to the following</label>
  <div>
    <div class="form-check" style="margin-left: 5pt">
      <input class="form-check-input" type="checkbox" name="sso[]" id="mailCheckbox" value="Mail">
      <label class="form-check-label" for="mailCheckbox">
        Mail
      </label>
    </div>
    <div class="form-check" style="margin-left: 5pt">
      <input class="form-check-input" type="checkbox" name="sso[]" id="payrollCheckbox" value="Payroll">
      <label class="form-check-label" for="payrollCheckbox">
        Payroll
      </label>
    </div>
    <div class="form-check" style="margin-left: 5pt">
      <input class="form-check-input" type="checkbox" name="sso[]" id="selfServiceCheckbox" value="Self-service">
      <label class="form-check-label" for="selfServiceCheckbox">
        Self-service
      </label>
    </div>
  </div>
</div>
```



```
<html>
    Username: <?php echo $_GET["name"]; ?> <br>
    Password: <?php echo $_GET["password"]; ?> <br>
    City of Employment: <?php echo $_GET["coe"]; ?> <br>
    Webserver: <?php echo $_GET["ws"]; ?> <br>
    Role: <?php echo $_GET["role"]; ?> <br>
    <?php
        $selected_sso = $_GET['sso'];
        echo "Selected SSO services: ";
        foreach($selected_sso as $sso) {
            echo $sso . ", ";
        }
    ?>
</html>
```



# 4 PHP Form Validation

## **Think SECURITY when processing PHP forms!**

These pages will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!

The HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:

## PHP Form Validation Example

*\* required field*

Name:  \*

E-mail:  \*

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other \*

Submit

# `$_SERVER["PHP_SELF"]` variable

---



The `$_SERVER["PHP_SELF"]` is a super global variable that returns the filename of the currently executing script.

So, the `$_SERVER["PHP_SELF"]` sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

# The function `test_input()`



Create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again):

- `trim()`: Strip unnecessary characters (extra space, tab, newline) from the user input data
- `stripslashes()`: Remove backslashes `\` from the user input data
- `htmlspecialchars()`: Converts special characters into HTML entities. This means that it will replace HTML characters like `<` and `>` with `&lt;` and `&gt;`;

# 5 PHP Forms - Required Fields

# PHP - Required Fields



From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required.

These fields cannot be empty and must be filled out in the HTML form.

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one



## PHP Form Validation Example

*\* required field*

Name:  \*

E-mail:  \*

Website:

Comment:

Gender: ☐ Female ☐ Male ☐ Other \*

Submit

Any

Question

