Superglobal Arrays

- PHP uses special predefined associative arrays called superglobal variables that allow the programmer to easily access HTTP headers, query string parameters, and other commonly needed information.
- They are called superglobal because these arrays are always in scope and always exist, ready for the programmer to access or modify them without having to use the global keyword.
- The following table shows the PHP global variables.

Name	Description
`$GLOBALS`	Array for storing data that needs superglobal scope
`$_COOKIES`	Array of cookie data passed to page via HTTP request
`$_ENV`	Array of server environment data
`$_FILES`	Array of file items uploaded to the server
`$_GET`	Array of query string data passed to the server via the URL
`$_POST`	Array of query string data passed to the server via the HTTP header
`$_REQUEST`	Array containing the contents of $_GET, $_POST, and $_COOKIES
`$_SESSION`	Array that contains session data
`$_SERVER`	Array containing information about the request and the server

Here, only the $_GET, $_POST and $_SERVER are discussed.

1. $_GET and $_POST Superglobal Arrays

- The $_GET and $_POST arrays are the most important superglobal variables in PHP since they allow the programmer to access data sent by the client.

- That data is formatted such that each value is associated with a name defined in the form.
- If the **form** was submitted using an **HTTP GET** request, then the resulting **URL** will contain the data in the **query string**. PHP will populate the **superglobal $_GET** array using the contents of this query string in the URL.
- **Figure 1** illustrates the relationship between an **HTML form**, the **GET request**, and the values in the **$_GET** array.

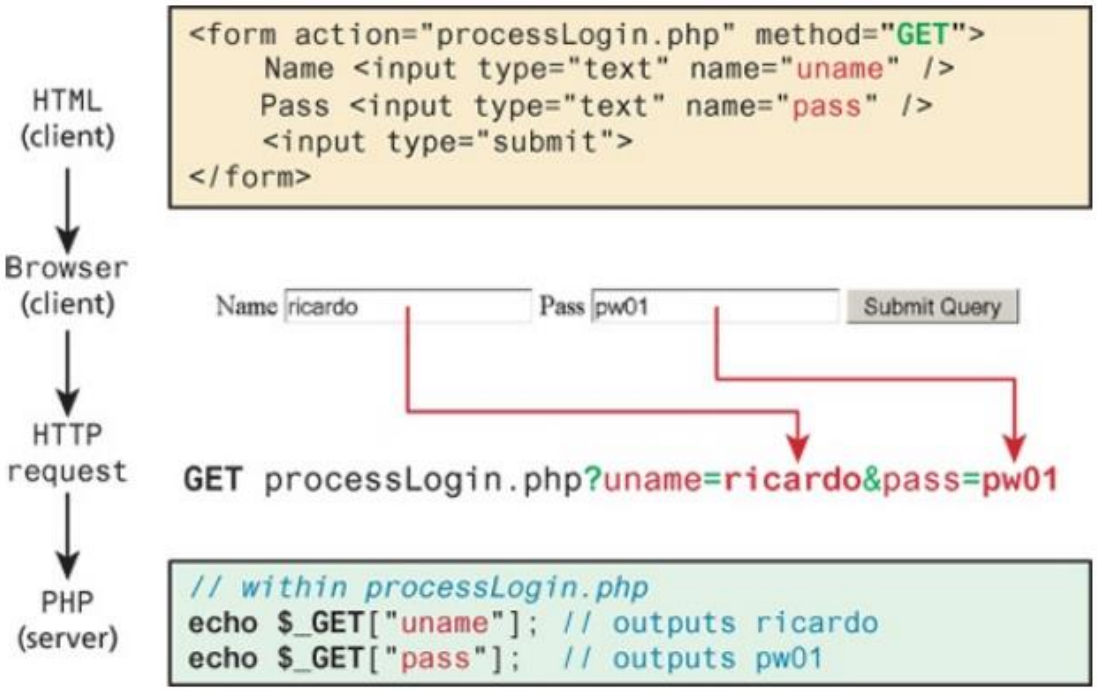

Figure 1. the relationship between an HTML form, the GET request, and the values in the $_GET array.

- If the form was sent using HTTP POST, then the values will not be visible in the URL, but will be sent through HTTP POST request body. From the PHP programmer's perspective, almost nothing changes from a GET data request except that those values and keys are now stored in the **$_POST** array.

- Figure 2 illustrates how data from a HTML form using POST populates the $_POST array in PHP.

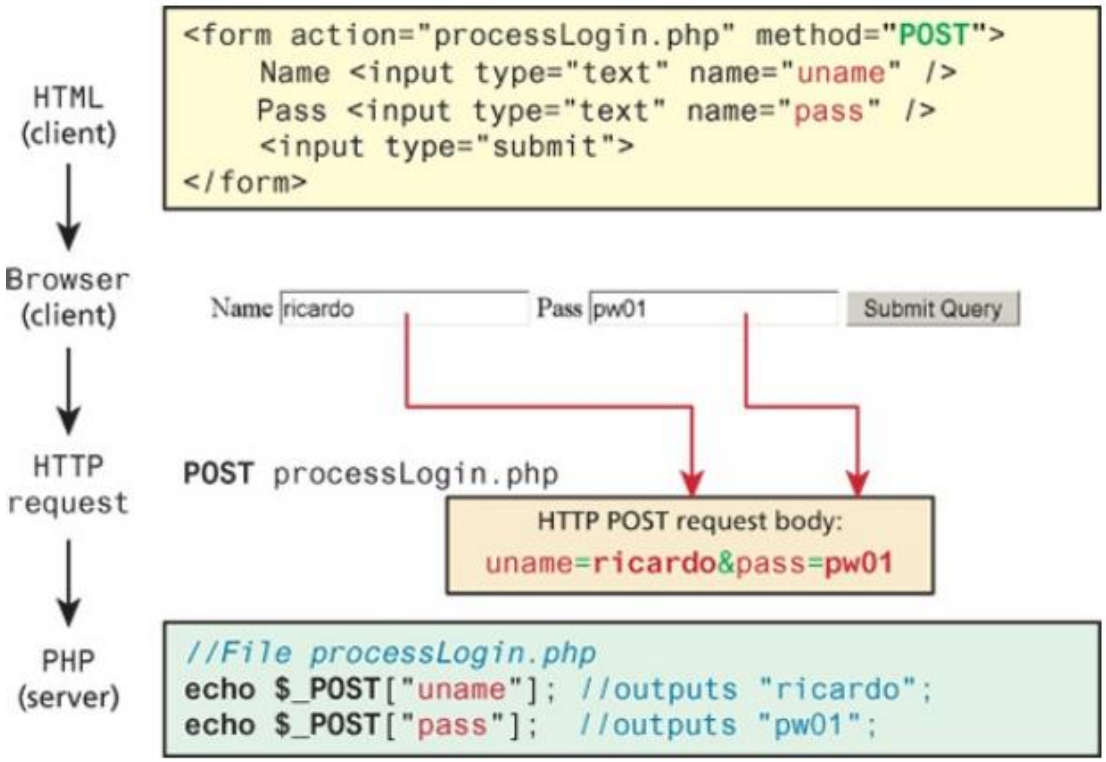

Figure 2. . the relationship between an HTML form, the POST request, and the values in the $_POST array.

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.

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

2. PHP $_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:

```php
<?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'];
?>
```

The following table lists the most important elements that can go inside $_SERVER:

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol
$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN 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 someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

3. Determining If Any Data Sent

- There will be times as you develop in PHP that you will use the same file to handle both the display of a form as well as handling the form input. For example, a single file is often used to display a login form to the user, and that same file also handles the processing of the submitted form data,

as shown in Figure 3. In such cases you may want to know whether any form data was submitted at all using either POST or GET.

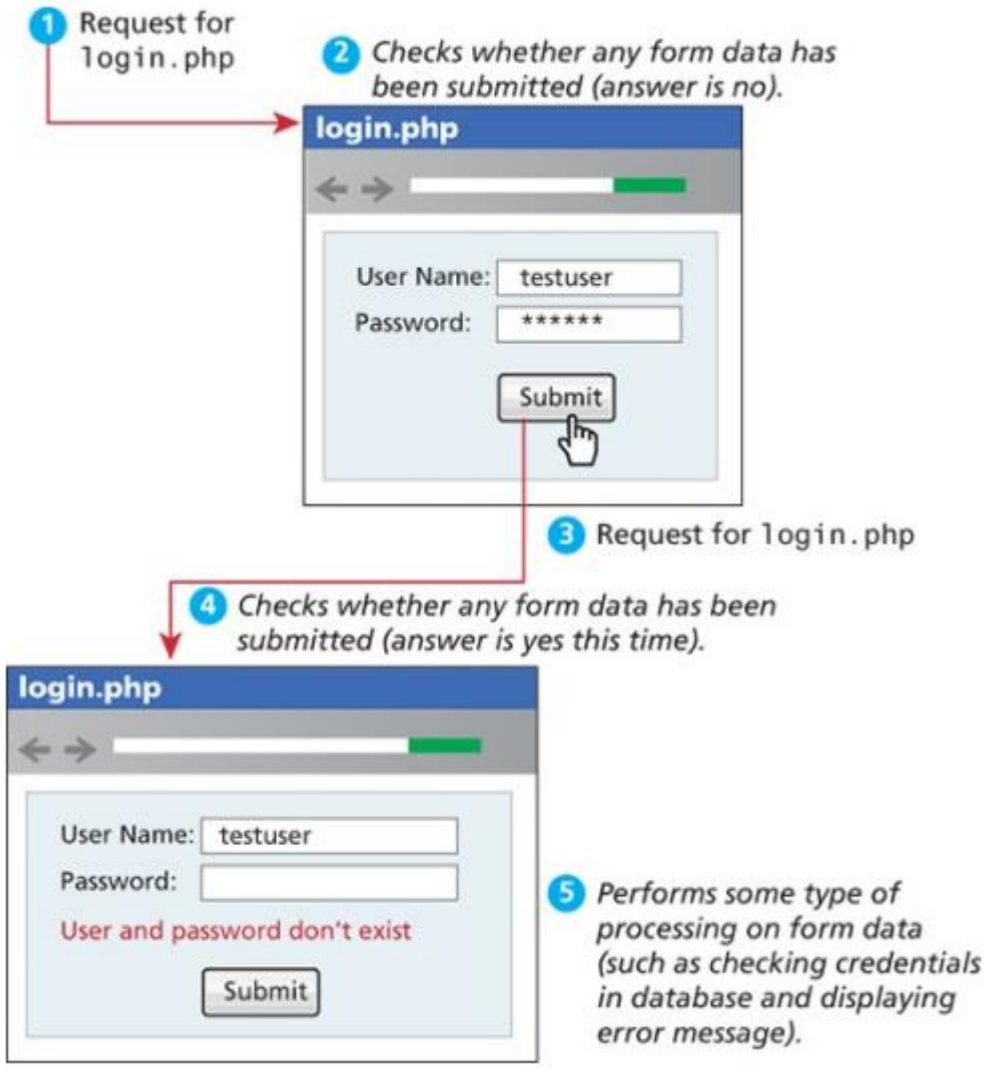

Figure 3. Form display and processing by the same PHP script.

- In PHP, there are several techniques to accomplish this task. First, you can determine if you are responding to a POST or GET by checking the $_SERVER['REQUEST_METHOD'] variable.
- It contains (as a string) the type of HTTP request this script is responding to (GET, POST, etc.).
- Even though you may know that, for instance, a POST request was performed, you may want to check if any of the fields are set. To do this

you can use the **isset()** function in PHP to see if there is any **value** set for a particular expected **key**, as shown in Listing 1.

```
<!DOCTYPE html>
<html>
<body>
<?php

if ($_SERVER["REQUEST_METHOD"] == "POST") {
        // handle the posted data.
        if (isset ($_POST[“uname”]) && isset($_POST[“pass”]) ) {
                 echo “handling user login now …”;
                echo “… here we could redirect or authenticate ”;
                echo “ and hide login form or something else”;
                }
}
?>
<h1>Some page that has a login form</h1>
<form action="<?php echo $_SERVER["PHP_SELF"];?>" method=“POST">
<Name <input type=“text” name=“uname”><br>
<Pass <input type=“password” name=“pass”><br>
<”input type=“submit>
<form/>
<body/>
</html>
```

Note

The PHP function isset() only returns false if a parameter name is missing altogether from the sent data. It still returns true if the parameter name exists and is associated with a blank value. For instance, let us imagine that the query string looks like the following:

uname=&pass=

In such a case the condition if(isset($_GET ['uname']) && isset ($_GET ['pass'])) will evaluate to **true** because something was sent for those keys، a **blank value** in this case. Thus, more coding will be necessary to further test the values of the parameters. Alternately, these two checks can be combined using the empty() function.

4. Accessing Form Array Data

- Sometimes in HTML forms you might have multiple values associated with a single name.

- Listing 2 provides an example where each checkbox has the same name value (name="day").

```
<form action="somepage.php" method="get">
    Please select days of the week you are free.<br>
     Monday <input type ="checkbox" name ="day" value="Monday"> <br>
     Tuesday <input type ="checkbox" name ="day" value ="Tuesday"> <br>
     Wednesday <input type="checkbox" name="day" value="Wednesday">
     Thursday <input type ="checkbox" name ="day" value ="Thursday"> <br>
     Friday <input type="checkbox" name ="day" value ="Friday"> <br>
     <input type ="submit" value ="Submit">
</form>
```

Unfortunately, if the user selects more than one day and submits the form, the $_GET['day'] value in the superglobal array will only contain the last value from the list that was selected.

To overcome this limitation, you must change the HTML in the form. In particular, you will have to change the name attribute for each checkbox from day to day[].

```
Monday <input type="checkbox" name="day[]" value="Monday">
Tuesday <input type="checkbox" name="day[]" value="Tuesday">
…
```

After making this change in the HTML, the corresponding variable $_GET['day'] will now have a value that is of type array. Knowing how to use arrays, you can process the output as shown in Listing 3 to echo the number of days selected and their values.

```
<?php

echo "You submitted " . count($_GET['day']) . "values";
foreach  ($_GET['day']  as  $d) {
   echo $d . " <br>";
}

?>
```

5. Using Query Strings in Hyperlinks

- Anchor tags (i.e., hyperlinks) also use the HTTP GET method.
- Indeed, it is extraordinarily common in web development to programmatically construct the URLs for a series of links from, for instance, database data.

- Imagine a web page in which we are displaying a list of book links. One approach would be to have a separate page for each book (as shown in Figure 1). This is not a very sensible approach. Our database may have hundreds or thousands of books in it: surely it would be too much work to create a separate page for each book!

- It would make a lot more sense to have a single Display Book page that receives as input a query string that specifies which book to display, as shown in Figure 2. Notice that we typically pass some type of unique identifier in the query string (in this case, the book's ISBN).
-