

MODULE IV

PASSING INFORMATION BETWEEN PAGES

PHP will catch the variable entered from one page to the next and make it available for further use. PHP is good in this form handling technology that is data passing function which makes it fast and easy to do for a wide variety of websites tasks.

HTML forms are mostly useful for passing a few values from a given page to one single other page of a website. In HTTP, the most basic technologies of information passages between web pages utilize GET and POST methods to create dynamically generated pages and to handle form data. These \$_GET and \$_POST are used to collect form data.

GET AND POST METHOD

The GET method passes arguments or values from one page to another as the part of uniform resource indicator query string. When it is used the GET method appends the indicated variable names and values to the URL designed in the ACTION attribute with a “?” (Question marks) separates and submit the whole thing to the processing agents. Here it is the web server.

Example	Output
<pre>login.php file <html > <body > <form action="welcome.php" method="get"> Name: <input type="text" name="name">
 E-mail: <input type="text" name="email">
 <input type="submit"> </form> </body> </html></pre>	<p>Name: <input type="text" value="Sharukh K"/></p> <p>E-mail: <input type="text"/></p> <p><input type="submit" value="Submit"/></p>
<pre>welcome.php file <html> <body> Welcome <?php echo \$_GET["name"]; ?>
 Your email address is: <?php echo \$_GET["email"]; ?> </body> </html></pre>	<p>Welcome Sharukh Khan Your email address is: sharook@gmail.com</p>

\$_GET Function

The built in \$_GET function is used to collect values from send with method="GET". Information sent from a form with the GET method is visible to every one. First will be displayed in the web browser title bar and has limits on the amount of information to send. When the user enters the name and email and click the submit button from the above example, the URL sent to server could look something like this:

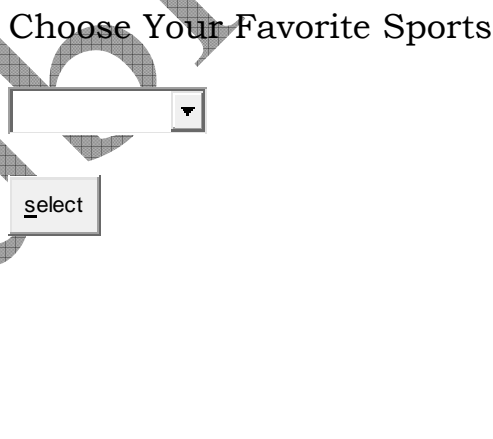
<http://localhost/MODULE%20%20II/welcome.php?name=Sharukh+Khan&mail=sharook%40gmail.com>

This welcome.php file can now use the \$_GET function to collect form data. When using method=GET in HTML forms all variable names and values are displayed in the URL. So this method should not be used when sending passwords or sensitive information, because the variables are displayed in the URL. It is possible to bookmark the page.

The GET method is not suitable for vary large variable values. It should not be used with values exceeding 2000 characters. So we use POST method and \$_POST function.

The built in \$_POST function is used to collect values from a form send with method="POST". Information send from a form with the post method is invisible to others and has no limit on the amount of information to send. There is an 8MB of maximum size for POST method.

Example	Output
<pre>form1.php file <html ><body > <form action="welcome1.php" method="post"> Choose Your Favorite Sports
 <select name="sports"> <option value="Select"> <option value="Base Ball">Base Ball <option value="Foot Ball">Foot Ball <option value="Cricket">Cricket <option value="Hockey">Hockey </select>

 <input type="submit" name="submit" value="select"> </form></body></html></pre>	
<pre>welcome1.php file <html><body> <p> You have indicated that you like <?php echo \$_POST["sports"]; ?>
 </body></html></pre>	You have indicated that you like Foot Ball

When the user fills the form and clicks the select button the URL will look like:

<http://localhost/MODULE%20%20II/welcome1.php>

The welcome1.php file can now use the \$_POST function to collect form data, because the variables are not displayed on the URL. It is not possible to bookmark the page.

Disadvantage of POST

The result of a given moment cannot be bookmarked. The web browser exhibits different behavior when the visible users the back and forward navigation button within the function.

GET vs. POST

Both GET and POST create an array (e.g. array(key => value, 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 super global, which means that they are always accessible, regardless of scope - and we 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. 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.

\$_REQUEST Function

The built in function `$_REQUEST` contains the contents of both `$_GET` and `$_POST`. It is used to collect form data and send with both the GET and POST method.

STRING FUNCTIONS:

A string is a sequence of characters that can be treated as a unique assigned to variables given as functions to appear on user's webpage. There are so many string functions are used in PHP.

➤ **strlen()**

It is used to return the length of a string. The length of the string often used in loop or other functions when it is important to know when the string ends. In a

loop we want to stop the loop after the last character in the string. The example below returns the length of the string "Hello world!":

Example	Output
<pre><?php echo strlen("Hello world!"); ?></pre>	12

➤ **strpos()**

It is used to search for a character or text within a string. If a match is found, this function returns the character position of the first match. If no match is found, it will return false. The first character position in a string is 0 (not 1). The example below searches for the text "world" in the string "Hello world!":

Example	Output
<pre><?php echo strpos("Hello world!", "world"); ?></pre>	6

➤ **strstr()**

➤ **strcmp()**

This function compares two strings and return
zero → if *String1 = String2*
less than zero → if *String1 < String2*
grater than zero → if *String1 > String2*

Example	Output
<pre><?php echo strcmp("Hello World!", "Hello World!")."
"; echo strcmp("Hello World!", "hello world!")."
"; echo strcmp("hello world!", "Hello World!"); ?></pre>	0 -1 1

➤ **substr()**

The function returns a part of a string.

Substr(String, Start, Length);

Where string specifies the string to return a path. Start specifies where to start in the string. Length specifies the length of the returned string.

Example	Output
<pre><?php echo substr("Hello World",6)."
"; echo substr("Hello World",10)."
"; echo substr("Hello World",3)."
"; echo substr("Hello World",-8)."
"; echo substr("Hello World",18)."
"; ?></pre>	World d lo World lo World

➤ **str_replace()**

The function replaces a part of a string with another string.

Substr_replace(string, replacement, start, length);

Example	Output
<pre><?php echo substr_replace("Hello world", "Earth", 6); ?></pre>	Hello Earth

➤ **strtolower()**

The function is used to convert a string into lowercase.

Example	Output
<pre><?php echo strtolower("HelLo WoRlD"); ?></pre>	hello world

➤ **ucfirst()**

The function converts the first character of the string to uppercase.

Example	Output
<pre><?php echo ucfirst("hello World, How are you"); ?></pre>	Hello World, How are you

➤ **ucwords()**

The function converts the first character of the each word in a string to uppercase.

Example	Output
<pre><?php echo ucwords("helLo world how are you"); ?></pre>	HelLo World How Are You

ARRAY CONSTRUCTS:

An array is a collection of variables indexed and bundled into single easy referenced super variables that offers an easy way to pass multiple values between lines of code, functions and even webpages. Each element in the array has its own index. So that it can be easily accessed.

Example

```
$abc=array("blue","red","green","yellow");
$abc[]=blue
$abc[]=red
$abc[]=green
$abc[]=yellow
```

In PHP there are three types of arrays

➤ **Numeric Array**

An array with a numeric Index

➤ **Associative Array**

An array where each id key is associated with a value

➤ **Multidimensional Array**

An array containing one or more array

1. Numeric Array

A numeric array stores each array element with a numeric index. There are two methods to create a numeric array.

a) Automatically assigned index method

Example	Output
<pre><html><body> <?php \$scars=array("suzuki","vovlvo","bmw","toyoto"); print_r(\$scars); ?> </body></html></pre>	Array ([0] => suzuki [1] => vovlvo [2] => bmw [3] => toyoto)

b) Manually assigned index method

Example	Output
<pre><html><body> <?php \$scars=array("suzuki","vovlvo","bmw","toyoto"); echo \$scars[0]." and ".\$scars[1]." are swedish cars"; ?> </body></html></pre>	suzuki and vovlvo are swedish cars

2. Associative Array

In this case id key is associated with a value associative array. W can use the value s as keys and assigns values to them.

```
$args=array("peter"→32, "Rose"→30, "Joe"→34);
```

In this example we use an array to assign ages to different persons.

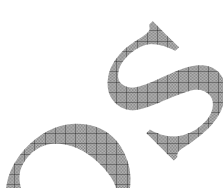
Example

```
<?php
$ages['Peter']="32";
$ages['Rose']="30";
$ages['Joe']="34";
echo "Peter is ". $ages['Peter']. "Years old";
?>
```

3. Multi-dimensional Array

In this case each element in the main array can also be an array and each element in the sub array can be array and soon.

Example	Output
<pre><html><body> <?php \$scars = array (array("Volvo",22,18), array("BMW",15,13), array("Saab",5,2), array("Land Rover",17,15)); echo \$scars[0][0].": In stock:</pre>	Volvo: In stock: 22, sold: 18. BMW: In stock: 15, sold: 13. Saab: In stock: 5, sold: 2. Land Rover: In stock: 17, sold: 15.

<pre> ".\$cars[0][1].", sold: ".\$cars[0][2]."
"; echo \$cars[1][0].": In stock: ".\$cars[1][1].", sold: ".\$cars[1][2]."
"; echo \$cars[2][0].": In stock: ".\$cars[2][1].", sold: ".\$cars[2][2]."
"; echo \$cars[3][0].": In stock: ".\$cars[3][1].", sold: ".\$cars[3][2]."
"; ?> </body></html> </pre>	
--	---

Some of Array Functions

The array functions allow us to access and manipulate arrays.

- array() – Creates an array
- array_change_key_case — Changes the case of all keys in an array
- array_chunk — Split an array into chunks
- array_column — Return the values from a single column in the input array
- array_count_values — Counts all the values of an array
- array_fill_keys — Fill an array with values, specifying keys
- array_fill — Fill an array with values
- array_flip — Exchanges all keys with their associated values in an array
- array_intersect — Computes the intersection of arrays
- array_key_exists — Checks if the given key or index exists in the array
- array_key_first — Gets the first key of an array
- array_key_last — Gets the last key of an array
- array_keys — Return all the keys or a subset of the keys of an array
- array_merge — Merge one or more arrays
- array_multisort — Sort multiple or multi-dimensional arrays
- array_pop — Pop the element off the end of array
- array_push — Push one or more elements onto the end of array
- array_rand — Pick one or more random keys out of an array
- array_replace — Replaces elements from passed arrays into the first array
- array_reverse — Return an array with elements in reverse order
- array_search — Searches the array for a given value and returns the first corresponding key if successful
- array_shift — Shift an element off the beginning of array
- array_sum — Calculate the sum of values in an array
- array_unique — Removes duplicate values from an array
- array_values — Return all the values of an array
- arsort — Sort an array in reverse order and maintain index association
- asort — Sort an array and maintain index association
- count — Count all elements in an array, or something in an object
- current — Return the current element in an array
- key — Fetch a key from an array

- krsort — Sort an array by key in reverse order
- ksort — Sort an array by key
- range — Create an array containing a range of elements
- rsort — Sort an array in reverse order
- shuffle — Shuffle an array
- sort — Sort an array

List()

Assign variables as if they were an array. The list() function is used to assign values to a list of variables in one operation. This function only works on numerical arrays.

Syntax

list(var1,var2...)

Example	Output
<pre><?php \$my_array = array("Dog","Cat","Horse"); list(\$a, , \$c) = \$my_array; echo "Here I only use the \$a and \$c variables."; ?></pre>	Here I only use the Dog and Horse variables.

Some of Examples

Examples	Output
<pre><?php \$people = array("Peter", "Joe", "Glenn", "Cleveland"); echo pos(\$people) . "
"; ?></pre>	Peter
<pre><?php \$cars=array("Volvo","BMW","Toyota"); sort(\$cars); \$length=count(\$cars); for(\$x=0;\$x<\$length;\$x++) { echo \$cars[\$x]; echo "
"; } ?></pre>	BMW Toyota Volvo
<?php	45

<pre>\$a=array(5,15,25); echo array_sum(\$a); ?></pre>	
<pre><?php \$a=array("a"=>"red","b"=>"green","c"=>"red"); print_r(array_unique(\$a)); ?></pre>	Array ([a] => red [b] => green)
<pre><?php \$cars=array("Volvo","BMW","Toyota"); echo count(\$cars); ?></pre>	3
<pre><?php \$a=array("Name"=>"Peter","Age"=>"41","Country"=>"USA"); print_r(array_values(\$a)); ?></pre>	Array ([0] => Peter [1] => 41 [2] => USA)
<pre><?php \$my_array = array("red","green","blue","yellow","purple"); shuffle(\$my_array); print_r(\$my_array); ?></pre> <p><i><p>Refresh the page to see how shuffle() randomizes the order of the elements in the array.</p></i></p>	Array ([0] => green [1] => purple [2] => red [3] => blue [4] => yellow) Refresh the page to see how shuffle() randomizes the order of the elements in the array.

PHP ADVANCED FUNCTIONS:

PHP contains three advanced functions:

1. HEADER

PHP headers are bits of information that are sent to a computer before anything else like a webpage is sent. When we view a website in a browser, we will never see these headers. They come before the webpage and all the browser will display is the content of the webpage. The header must come before any of our other content on the page.

header('Location:')

One header is the location header. It is the URL has of the page we are requesting. It tells the browser where to find the page that we are looking for. This is useful for when we have an old page that people may have bookmarked, we can send them to the new location automatically when they try to load the page.

Example

```
<?php
header('Location://www.google.com');
exit;
?>
<html> <body>
```

```
<?php
echo "Hello";
?>
</body> </html>
```

header('Refresh:')

Redirecting a user to another page right away is fine and all, but what if we wanted to give the user a little bit of time to see a message or something before we sent them elsewhere.

Example:

```
<?php
header('Refresh:10; url=http://www.google.com');
echo "you will be redirect to GOOGLE in 9 seconds";
exit;
?>
<html><body>
<?php
echo "Hello";
?>
</body></html>
```

header('Content-Type:')

With a PHP Content-Type header, we can change how we want the browser to read the page. For the normal page the Content-type is text/html. But we could change that in the header to be text/plain and the browser will display the source code of our site. We can also use it to display PDF's and other documents.

Example

```
<?php
header('Content-Type: text/plain');
echo "you may change text/plain as html or pdf";
exit;
?>
<html> <body>
<?php
echo "Hello";
?>
</body> </html>
```

header('Content-Disposition:')

With Content-Disposition we tell the browser how to handle the document. If we have a PDF that we want user to download, we can use this Content-Disposition to make the browser display a save dialogue box. We must pt the filename so that it can be downloaded.

Example:

```
<?php
header('Content-Disposition: attachment; filename="example.pdf");
echo "The file example.pdf were downloading... ";
```

```

exit;
?>
<html> <body>
<?php
echo "Hello";
?>
</body> </html>

```

header('Cache-Control:')

When we view webpages, our browser might be storing them in cache somewhere to reference later. When we come back to the site, it might load faster. The problem is that if we have a site that constantly updates, like a news site we don't want it to be cached. Otherwise there is no recent news updates. If it does not get cached we specify no-cache and then tell the browser to revalidate the page with the original server.

Example:

```

<?php
header('Cache-Control: no-cache, must-revalidate');
echo "This page not cached ";
exit;
?>
<html> <body>
<?php
echo "Hello";
?>
</body> </html>

```

header("Expires:")

Using Expires header we can set the date for when the page cache is to expire. To make sure that the page is never cached. We can set date in the past so that it will always expire and must reload with new content.

Example:

```

<?php
header('Expires: Sat, 26 Sep 2016 05:00:00 GMT');
echo "This page never cached ";
exit;
?>
<html> <body>
<?php
echo "Hello";
?>
</body> </html>

```

2. SESSION

A session is a way to store information (in variables) to be used across multiple pages. Unlike a cookie, the information is not stored on the user's computer. When we work with an application, we open it, do some changes, and then we close it. This is much like a Session. The computer knows who we are. It

knows when we start the application and when we end. But on the internet there is one problem: the web server does not know who we are or what we do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser. So; Session variables hold information about one single user, and are available to all pages in one application. If we need a permanent storage, we may want to store the data in a database. A session is started with the `session_start()` function. This function must be the very first thing in our document. Before any HTML tags, A Session variables are set with the PHP global variable: `$_SESSION`.

A Session variables hold information about one single user and are available to all pages in one application. The PHP allows storing information on the server for later use such as user name, shopping items etc. however session information is temporarily and will be deleted after the user has left the website. Session works by creating a unique id (UID) for each visitor and store variable based on this new id, which is stored in cookie and is used to reference the session file on the server. As such the user has no access to the content of the session file there by providing secure alternatives to cookies. Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example	Output
<pre data-bbox="118 1113 678 1505"><?php session_start(); ?> <html><body> <?php \$_SESSION["favcolor"] = "green"; \$_SESSION["favanimal"] = "cat"; echo "Session variables are set."; ?> </body></html></pre>	<p data-bbox="826 1151 1257 1189">Session variables are set.</p>

Making a PHP Session Variable

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php"). Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (`session_start()`). Also notice that all session variable values are stored in the global `$_SESSION` variable:

Example	Output
<pre data-bbox="118 1883 375 2069"><?php session_start(); ?> <html><body> <?php</pre>	<p data-bbox="826 1921 1214 2002">Favorite color is green. Favorite animal is cat.</p>

<pre> echo "Favorite color is " . \$_SESSION["favcolor"] . "
"; echo "Favorite animal is " . \$_SESSION["favanimal"] . "."; ?> </body></html> </pre>	
--	--

Note: Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

Modify a PHP Session Variable

To change a session variable, just overwrite it:

Example	Output
<pre> <?php session_start(); ?> <html><body> <?php \$_SESSION["favcolor"] = "yellow"; print_r(\$_SESSION); ?> </body></html> </pre>	<pre> Array ([favcolor] => yellow [favanimal] => cat) </pre>

Destroy a PHP Session

To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`:

Example	Output
<pre> <?php session_start(); ?> <html><body> <?php echo "All session variables are now removed, and the session is destroyed." ?> </body></html> </pre>	<pre> All session variables are now removed, and the session is destroyed. </pre>

3. COOKIE

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, we can both create and retrieve cookie values. A cookie is created with the `setcookie()` function. The `isset()` function is used to find out if a cookie has been set or not. It enables us to maintain the state of a users visit to a website. So that we can write their movement through the site or to store the information such as user name, password and address etc.

After they have entered it one page so that they have to keep reentering it on different pages. Cookies are sent by a script or web server to the web browser. The browser is responsible for sending the cookies through http request headers to all successive pages that belong to the web application. Cookies are limited in size and quantity (4KB each and 20 cookies per domain). A cookie often identifies a user. It is a small file that server enables on the user's computer. Each time the same computer request a page with a browser, it will set the cookie too.

Syntax

```
setcookie(name, value, expire, path, domain, secure);
```

Only the name parameter is required. All other parameters are optional.

Name: - the name of the cookie

Value:- the value of the cookie (this value stored in the client computer. So don't store sensitive information)

Expire:- the time the cookie expire

Path:- the path on the server in which the cookie will be available on.

Domain:- the domain for which the cookie is available

Secure:- the cookie should only be transmitted over a secure http connection

Example:

```
<?php
setcookie("user","Bill Gates",time(63400));
echo "cookie would set";
?>
```

Create/Retrieve a Cookie

We retrieve the value of the cookie "user", we also use the isset() function to find out if the cookie is set:

Example

```
<?php
echo $_cookie['user'];
print_r($cookie);
?>
```

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URL encoding, use setrawcookie() instead).

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

Example	Output
<pre><?php \$cookie_name = "user"; \$cookie_value = "Alex Porter"; setcookie(\$cookie_name, \$cookie_value, time() + (86400 * 30), "/");</pre>	<p><u>First Output</u> Cookie named 'user' is not set</p> <p>Note: You might have to reload the page to see the new value of the cookie.</p>

<pre> ?> <html><body> <?php if(!isset(\$_COOKIE[\$cookie_name])) { echo "Cookie named '" . \$cookie_name . "' is not set!"; } else { echo "Cookie '" . \$cookie_name . "' is set!
"; echo "Value is: " . \$_COOKIE[\$cookie_name]; } ?> <p>Note: You might have to reload the page to see the new value of the cookie.</p> </body></html> </pre>	<p><u>Second Output</u> Cookie 'user' is set! Value is: Alex Porter</p> <p>Note: You might have to reload the page to see the new value of the cookie.</p>
---	--

Delete a Cookie

To delete a cookie, use the `setcookie()` function with an expiration date in the past:

Example	Output
<pre> <?php setcookie("user", "", time() - 3600); ?> <html><body> <?php echo "Cookie 'user' is deleted."; ?> </body></html> </pre>	<p>Cookie 'user' is deleted.</p>

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the `setcookie()` function, then count the `$_COOKIE` array variable:

Example	Output
<pre> <?php setcookie("test_cookie", "test", time() + 3600, '/'); ?> <html><body><?php if(count(\$_COOKIE) > 0) { echo "Cookies are enabled."; } else { echo "Cookies are disabled."; } ?></body></html> </pre>	<p>Cookies are enabled.</p>