

FORM ELEMENTS, VALIDATING USER INPUT, ERROR HANDLING



INTRODUCTION

- Form Handling
- Validation user inputs
- Using radio buttons, checkbox, list box, buttons, text box, etc., processing user input,
- Handling and Avoiding errors
- Exception Handling.



FORM HANDLING

- *Forms are how users communicate with your server:*
 - signing up for a new account,
 - searching a forum for all the posts about a particular subject,
 - finding a nearby restaurant or shoemaker,
 - or buying a book



USING A FORM

- Using a form in a PHP program is a two-step activity.
 - Step one is to display the form.
 - This involves constructing HTML form that has tags for the appropriate user-interface elements in it, such as text boxes, checkboxes, and buttons
- When a user sees a page with a form in it, she/he inputs the information into the form and then clicks a button or hits Enter to send the form information back to your server.
- Processing that submitted form information is step two of the operation.

GET VS. POST


- Both GET and POST create an array (e.g. array(key1 => 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 (super globals).
- \$_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.

SUPER GLOBALS

- 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



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.
 - 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.



USING THE POST/GET METHOD IN A PHP FORM

- Post data is accessed with the `$_POST` array in PHP.
- Get data is accessed with the `$_GET` array.
- **Using “isset”**
- You can use the “isset” function on any variable to determine if it has been set or not.
- You can use this function on the `$_POST` array to determine if the variable was posted or not.
- This is often applied to the submit button value, but can be applied to any variable.



CAN BOTH GET AND POST BE USED IN THE SAME PAGE?

- GET and POST occupy different spaces in the server's memory, so both can be accessed on the same page if you want.
- One use might be to display different messages on a form depending on what's in the query string.



THE \$_REQUEST VARIABLE

- The PHP \$_REQUEST variable contains the contents of both \$_GET, \$_POST, as well as \$_COOKIE.
- The PHP \$_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.
- E.g

```
<!doctype html>
<head><title> request demo </title> </head>
<body>

    <form method = "get" action = "<?php
                                $_SERVER['PHP_SELF'];?> " >
        Name : <input type='text' name='fname'>
        <input type ='submit'>
    </form>
```



```
<?php
    $name = $_REQUEST[ 'fname' ];
    if(empty($name))
        echo "Name is empty <br>";
    else
        echo $name;

?>
</body>
</html>
```



What is the `$_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.

What is the `htmlspecialchars()` function?

- The `htmlspecialchars()` function converts special characters to HTML entities. This means that it will replace HTML characters like `<` and `>` with `<` and `>`. This prevents attackers from exploiting the code by injecting HTML or Javascript code



PHP INCLUDE() FUNCTION

- The include() function will take all the content in a specified file and includes it in the current file where the include statement is called.



PHP – INCLUDE() AND REQUIRE()

- These functions can be used to insert content of one PHP file into another PHP file (on the server side) before the server executes the file
- It can be very useful, for instance to include header, menu or footer files.




REQUIRE() FUNCTION

- The require() function is almost identical as the include() function. The only difference is how they handle errors.
- If the file cannot be found or the page cannot be opened, the require() function considers the error to be fatal and stops executing the remainder of the file
- E.g.

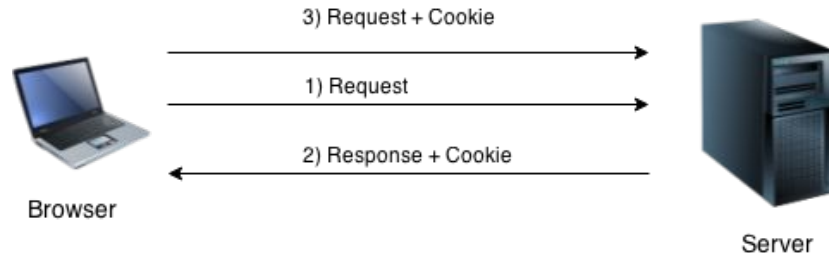


THE DIFFERENCES

- If an **include()** statement fails it will generate a warning, but the script will continue the execution.
 - If a **require()** statement fails it will generate a fatal error and the execution will stop.
 - The **require_once()** and **include_once()** are identical to **require()** and **include()** functions except PHP will check if the file has already been included, and if so, not include (require) it again.
 - If a **require_once()** statement fails it will generate a fatal error and the execution will stop.
- 

PHP COOKIES

- A cookie in PHP is a small file with a maximum size of 4KB that the web server stores on the client computer.
- They are typically used to keep track of information such as a username that the site can retrieve to personalize the page when the user visits the website next time.
- PHP cookie is a small piece of information which is stored at client browser. It is used to recognize the user.
- Cookie is created at server side and saved to client browser.
- Each time when client sends request to the server, cookie is embedded with request. Such way, cookie can be received at the server side.



- `setcookies()` function
- It is used to set cookie with HTTP response. Once cookie is set, you can access it by `$_COOKIE` superglobal variable.
- Cookies are usually set in an HTTP header but JavaScript can also set a cookie directly on a browser.
- **`setcookie(name, value, expire, path, domain, security);`**
 - Name: It is used to set the name of the cookie.
 - Value: It is used to set the value of the cookie.
 - Expire: It is used to set the expiry timestamp of the cookie after which the cookie can't be accessed.
 - Path: It is used to specify the path on the server for which the cookie will be available.
 - Domain: It is used to specify the domain for which the cookie is available.
 - Security: It is used to indicate that the cookie should be sent only if a secure HTTPS connection exists.

- For delete the cookies
- `setcookie('usrname', "", time() - 3600);`
- `Cookiesdemo.php` , `cookies2.php`



PHP SESSIONS

- ❑ Sessions are a simple way to store data for individual users against a unique session ID. This can be used to persist state information between page requests.
- ❑ Global most variable.
- ❑ Session IDs are normally sent to the browser via session cookies and the ID is used to retrieve existing session data.
- ❑ PHP session technique is widely used in shopping websites where we need to store and pass cart information e.g. username, product code, product name, product price etc from one page to another



- ❑ Cookies and Sessions are used to store information. Cookies are only stored on the client-side machine,
- ❑ while sessions get stored on the client as well as a server.
- ❑ Session
- ❑ A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.
- ❑ Cookies are text files stored on the client computer and they are kept of use tracking purpose. Server script sends a set of cookies to the browser. For example name, age, or identification number etc. The browser stores this information on a local machine for future use.

- `Session_Start()` – This event is fired whenever a new user visits the application i.e. a new session is started.
- `Session_End()` – This event is fired whenever a session of user times out.
- **Starting a PHP Session:** The first step is to start up a session. After a session is started, session variables can be created to store information. The PHP `session_start()` function is used to begin a new session. It also creates a new session ID for the user.

```
<?php
```

```
session_start();
```



- `session_unset()` : frees all session variables currently registered.

```
<?php
session_start();
?>
<html>
<body>
<?php
$_SESSION["user"] = "Sachin";
echo "Session information are set successfully.<br/>";
?>
<a href="session2.php">Visit next page</a>
```



□ Session2.php

```
<?php
session_start();
?>
<html>
<body>
<?php
    echo "User is: ".$_SESSION["user"];
?>
</body>
</html>
```



- PHP `session_destroy()` function is used to destroy all session variables completely.
- `Sessionset.php` `sessionview.php` `sessionout.php`



- PHP session technique is widely used in shopping websites where we need to store and pass cart information e.g. username, product code, product name, product price etc from one page to another.



FILE HANDLING

- File handling is an important part of any web application.
- You often need to open and process a file for different tasks.
- When you are manipulating files you must be very careful.
- You can do a lot of damage if you do something wrong.
 - Common errors are:
 - editing the wrong file,
 - filling a hard-drive with garbage data,
 - and deleting the content of a file by accident.



FILE FUNCTIONS

- ❑ **readfile():** The readfile() function reads a file and writes it to the output buffer.
- ❑ E.g. echo readfile("file.txt");
- ❑ **fopen():** A better method to open files is with the fopen() function.
- ❑ It gives you more options than the readfile() function.
- ❑ Syntax: fopen(filename, mode);




FILE OPENING MODES

- The file may be opened in one of the following modes:

Modes	Description
r	Open a file for read only. File pointer starts at the beginning of the file
w	Open a file for write only. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	Creates a new file for write only. Returns FALSE and an error if file already exists
r+	Open a file for read/write. File pointer starts at the beginning of the file
w+	Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	Creates a new file for read/write. Returns FALSE and an error if file already exists

FILE FUNCTIONS

- ❑ **fread()** reads from an open file.
 - ❑ Syntax: `fread(filename, size);` where size is in number of bytes to be read.
 - ❑ E.g. `fread($myfile, filesize("file.txt"));`

 - ❑ **fclose()** is used to close an open file.
 - ❑ It's a good programming practice to close all files after you have finished with them.
 - ❑ The `fclose()` requires the name of the file (or a variable that holds the filename) we want to close.
 - ❑ Syntax: `fclose($filename);`
 - ❑ E.g. `fclose($filename);`
- 

FILE FUNCTIONS

- ❑ **fgets()** is used to read a single line from a file.
- ❑ Syntax: fgets(\$filename);
- ❑ **fgetc()** is used to read a single character from a file.
- ❑ Syntax: fgetc(\$filename);
- ❑ **feof()** checks if the "end-of-file" (EOF) has been reached.
- ❑ It is useful for looping through data of unknown length.
- ❑ File2.php



FILE FUNCTIONS

- **fwrite()** is used to write to a file.
- The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.
- E.g. fwrite(\$myfile, \$txt);



FILE UPLOAD

- With PHP, it is easy to upload files to the server.
- In the form tag do the following, without which upload won't work:
 - Make sure that the form uses `method="post"`
 - The form also needs the following attribute: `enctype="multipart/form-data"`. It specifies which content-type to use when submitting the form.
- Set the attribute `type="file"` which shows the input field as a file-select control, with a "Browse" button next to the input control.



CREATING AN UPLOAD SCRIPT

- There is one global PHP variable called `$_FILES`.
- This variable is an associate double dimension array and keeps all the information related to uploaded file.
- So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create following five variables –
 - `$_FILES['file']['tmp_name']` – the uploaded file in the temporary directory on the web server.
 - `$_FILES['file']['name']` – the actual name of the uploaded file.
 - `$_FILES['file']['size']` – the size in bytes of the uploaded file.
 - `$_FILES['file']['type']` – the MIME type of the uploaded file.
 - `$_FILES['file']['error']` – the error code associated with this

PHP MOVE_UPLOADED_FILE() FUNCTION

- The `move_uploaded_file()` function moves an uploaded file to a new location.
- This function returns `TRUE` on success, or `FALSE` on failure.
- **Syntax**
- `move_uploaded_file(file, newloc)`
 - **File:** Required. Specifies the file to be moved.
 - **Newloc:** Required. Specifies the new location for the file.
- **Note:** This function only works on files uploaded via HTTP POST. If the destination file already exists, it will be overwritten.
- Upload.php ,upload.html



ERROR HANDLING

- ❑ Error handling is the process of catching errors raised by your program and then taking appropriate action.
- ❑ In the absence of error handling,
 - Your program may look very unprofessional and you may be open to security risks.
- ❑ There are three main ways of error handling:
 - Simple "die()" statements
 - Custom errors and error triggers
 - Error reporting



USING THE DIE() FUNCTION

- Die() function stops the execution of the code once the error is detected and displays a meaningful and user friendly message to the user.
- E.g. `die("File does not exist");`



CUSTOM ERROR HANDLING

- You can write your own function to handle any error. PHP provides you a framework to define error handling function.

- **Syntax**

- `error_function(error_level, error_message, error_file, error_line error context).`

Parameter	Description
error_level	Required - Specifies the error report level for the user-defined error. Must be a value number.
error_message	Required - Specifies the error message for the user-defined error
error_file	Optional - Specifies the filename in which the error occurred
error_line	Optional - Specifies the line number in which the error occurred
error_context	Optional - Specifies an array containing every variable and their values in use when the error occurred



CUSTOM ERROR HANDLER

- The function then needs to be registered as your custom error handler:

```
set_error_handler( 'err_handler' );
```

- You can 'mask' the custom error handler so it only receives certain types of error. e.g. to register a custom handler just for user triggered errors:

```
set_error_handler( 'err_handler' ,  
E_USER_NOTICE | E_USER_WARNING | E_USER_ERROR );
```

- A custom error handler is never passed **E_PARSE**, **E_CORE_ERROR** or **E_COMPILE_ERROR** errors as these are considered too dangerous.
- Often used in conjunction with a 'debug' flag for neat combination of debug and production code display.

TYPES OF ERRORS

- ❑ There are 3 basic types of runtime errors in PHP
- ❑ **Notices** : These are trivial, non-critical errors that PHP encounters while executing a script. By default, such errors are not displayed to the user at all – we can change this default behavior.
- ❑ **Warnings** : These are more serious errors - for example, attempting to include() a file which does not exist. By default, these errors are displayed to the user, but they do not result in script termination.
- ❑ **Fatal errors** : These are critical errors - for example, instantiating an object of a non-existent class, or calling a non-existent function. **These errors cause the immediate termination of the script, and PHP's default behavior**

POSSIBLE ERROR LEVEL

Sr. No	Constant & Description	Value
1	.E_ERROR Fatal run-time errors. Execution of the script is halted	1
2	E_WARNING Non-fatal run-time errors. Execution of the script is not halted	2
3	E_PARSE Compile-time parse errors. Parse errors should only be generated by the parser.	4
4	E_NOTICE Run-time notices. The script found something that might be an error, but could also happen when running a script normally	8
5	E_CORE_ERROR Fatal errors that occur during PHP's initial start-up.	16



ERROR REPORTING

- ❑ Error reporting is used to control which errors are displayed, and which are simply ignored.
- ❑ The effect only lasts for the duration of the execution of your script.
- ❑ The `error_reporting()` function specifies which errors are reported.
- ❑ Syntax: `error_reporting($level)`
 - E.g.
- ❑ Returns the old error reporting level or the current error reporting level if no *level* parameter is given
- ❑ The level can also be set to zero but hiding errors is NOT a solution to a problem.
- ❑ While developing and debugging code, displaying **all** errors is

Sr. No	Constant & Description	Value
7	E_USER_ERROR Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()	256
8	E_USER_WARNING Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()	512
9	E_USER_NOTICE User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()	1024
10	E_STRICT Run-time notices. Enable to have PHP suggest changes to your code which will ensure the best interoperability and forward compatibility of your code.	2048
11	E_RECOVERABLE_ERROR Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())	4096
12	E_ALL All errors and warnings, except level E_STRICT (E_STRICT will be part of E_ALL as of PHP 6.0)	8191

All the above error level can be set using following PHP built-in library function

```
int error_reporting ( [int $level] );
```



SET ERROR REPORTING SETTINGS

```
<?php
```

```
//Turn off all error reporting
```

```
error_reporting(0);
```

```
// Report simple running errors
```

```
error_reporting(E_ERROR | E_WARNING | E_PARSE);
```

```
/*Reporting E_NOTICE can be good too (to report  
uninitialized variables or catch variable name  
misspellings ...) */
```

```
error_reporting(E_ERROR | E_WARNING | E_PARSE |  
E_NOTICE);
```

```
// Report all errors except E_NOTICE
```

```
error_reporting(E_ALL ^ E_NOTICE);
```

```
// Report ALL PHP
```



IDENTIFYING ERRORS

E_STRICT	Feature or behaviour is depreciated (PHP5).
E_NOTICE	Detection of a situation that could indicate a problem, but might be normal.
E_USER_NOTICE	User triggered notice.
E_WARNING	Actionable error occurred during execution.
E_USER_WARNING	User triggered warning.
E_COMPILE_WARNING	Error occurred during script compilation (unusual)
E_CORE_WARNING	Error during initialization of the PHP engine.
E_ERROR	Unrecoverable error in PHP code execution.
E_USER_ERROR	User triggered fatal error.
E_COMPILE_ERROR	Critical error occurred while trying to read script.
E_CORE_ERROR	Occurs if PHP engine cannot startup/etc.
E_PARSE	Raised during compilation in response to syntax error

notice

warning

fatal



TRIGGER AN ERROR

- ❑ In a script where users can input data it is useful to trigger errors when an illegal input occurs.
- ❑ This is done by the `trigger_error()` function.
- ❑ `trigger_error()` function.
- ❑ Generates a user-level error/warning/notice message
- ❑ An error can be triggered anywhere you wish in a script, and by adding a second parameter, error level.
- ❑ Possible error types:
 - `E_USER_ERROR` - Fatal user-generated run-time error. Errors that can not be recovered from. Execution of the script is halted
 - `E_USER_WARNING` - Non-fatal user-generated run-time warning. Execution of the script is not halted