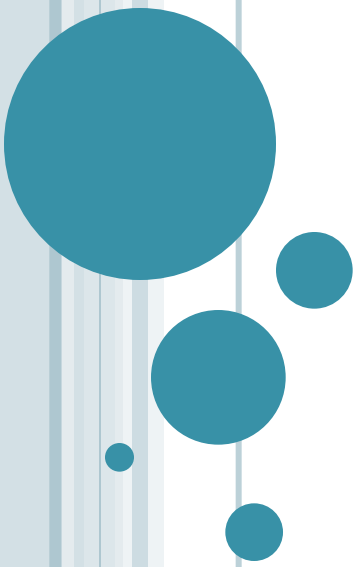
Form Elements, Validating user Input, Error Handling



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,
  - retrieving a lost password,
  - 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.

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.

- E.g. form1.php, welcome.php, welcome_get.php, form2.php

Using PHP_SELF in the action field of a form

- PHP_SELF is a variable that returns the current script being executed.
- It returns the name and path of the current file (from the root folder).
- You can use this variable in the action field of the FORM.
- Example:
  echo $_SERVER['PHP_SELF']; (form3.php, nameAge.php)
- Some more example for requesting server information (golbalServer.php, globalRequest.php)

Using the POST/GET method in a PHP form

- Post data is accessed with the $_POST array in PHP. E.g. (form4.html and form4.php)
- Get data is accessed with the $_GET array. E.g. (askSum.php and showSum.php)
- **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.
- E.g. (form5.html and form5.php, form_action.php)

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. sumArray.php, DataEntrySumArray.php, ProcessSumArray.php
- Different form controls:
  - processform.html, processform.php

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.
- The three functions are almost identical, but there are some minor differences between the functions
- For example, how errors are handled.

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 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.
- So for example the following file: (includeTest.php, header.php)

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. requireTest.php

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);
- file1.php, file2.php

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")); (file3.php)

- **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($filename);
- E.g. fclose($filaname);

File Functions

- **fgets()** is used to read a single line from a file.
- Syntax: fgets($filaname);
- E.g. file2.php

- **fgetc()** is used to read a single character from a file.
- Syntax: fgetc($filename);
- E.g. file3.php

- **feof()** checks if the "end-of-file" (EOF) has been reached.
- It is useful for looping through data of unknown length.
- E.g. file4.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); (file5.php)

File upload

- With PHP, it is easy to upload files to the server.
- To configure file upload, search for the file_uploads directive in your "php.ini" file, and set it to On. (C:\xampp\php)
- In the form tag do the following, without which upload wont 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.
- E.g. upload.html, upload.php

File upload

- ; Whether to allow HTTP file uploads.
  - file_uploads=On
- ; Temporary directory for HTTP uploaded files (will use system default if not specified).
  - upload_tmp_dir="C:\xampp\tmp"
- ; Maximum allowed size for uploaded files.
  - upload_max_filesize=2M
- ; Maximum number of files that can be uploaded via a single request
  - max_file_uploads=20

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 file upload.

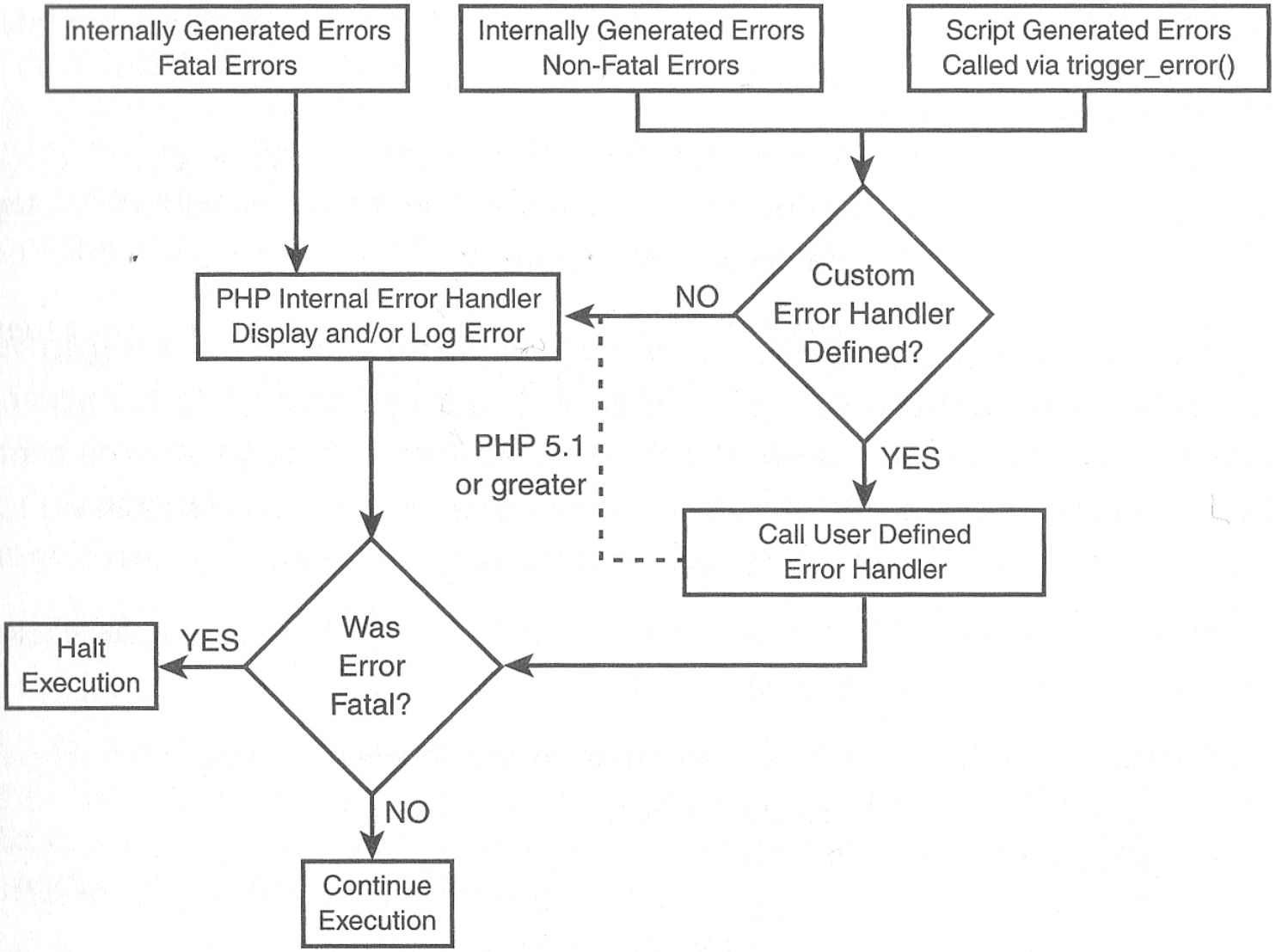
PHP MOVE_UPLOADED_FILE() FUNCTION

- 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.

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

Error Handling mechanism



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

- A custom error handler is never passed E_PARSE, E_CORE_ERROR or E_COMPILE_ERROR errors as these are considered too dangerous.

Possible Error levels

Value	Constant	Description
1	E_ERROR	Fatal run-time errors. Execution of the script is halted
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted
4	E_PARSE	Compile-time parse errors. Parse errors should only be generated by the parser.
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally
16	E_CORE_ERROR	Fatal errors that occur during PHP's initial startup.
32	E_CORE_WARNING	Non-fatal run-time errors. This occurs during PHP's initial startup.
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()
2048	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.
4096	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())
8191	E_ALL	All errors and warnings, except level E_STRICT (E_STRICT will be part of E_ALL as of PHP 6.0)

All the above error level can be set using following PHP built-in library function
int error_reporting ([int $level]); (ErrorHandler.1php, ErrorHandler2.php)

Identifying Errors

E_STRICT	Feature or behaviour is depreciated (PHP5).	notice
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.	warning
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.	fatal
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	

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 errors
error_reporting(E_ALL);
?>
```

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.
- E.g. ErrorHandler1.php, ErrorHandler2.php

Set error reporting settings

- **Hiding errors is NOT a solution to a problem.**
- It is useful, however, to hide any errors produced on a live server.
- While developing and debugging code, displaying all errors is highly recommended!

Suppressing Errors

- The special `@` operator can be used to suppress function errors.
- Any error produced by the function is suppressed and not displayed by PHP regardless of the error reporting setting.
- E.g. @fopen($file); **OR**
- if (file_exists($file)) { fopen($file); } else { die('File not found'); }
- **Error suppression is NOT a solution to a problem.**
- It can be useful to locally define your own error handling mechanisms.
- If you suppress any errors, you must check for them yourself elsewhere.
- @ operator is also quite slow as PHP incurs some overhead to suppress the errors, warnings or notices.
  - You can pretty much always make use of checks like isset() or file_exists() to avoid the notices or warnings that you were trying to hide with @ or else debugging might become a nightmare for you later.

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.
- 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
  - E_USER_NOTICE - Default. User-generated run-time notice. The script found something that might be an error, but could also happen when running a script normally
- E.g. TriggorError1.php, TriggorError2.php

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. ErroReporting.php
- 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 highly recommended.

Error Log

- Returns True on success and False on failure.

Syntax

```
error_log(message,type,destination,headers);
```

Parameter	Description
message	Required. Specifies the error message to log
type	Optional. Specifies where the error message should go. Possible values: • 0 - Default. Message is sent to PHP's system logger, using the OS' system logging mechanism or a file, depending on what the error_log configuration is set to in php.ini • 1 - Message is sent by email to the address in the *destination* parameter • 2 - No longer in use (only available in PHP 3) • 3 - Message is appended to the file specified in *destination* • 4 - Message is sent directly to the SAPI logging handler
destination	Optional. Specifies the destination of the error message. This value depends on the value of the *type* parameter
headers	Optional. Only used if the *type* parameter is set to 1. Specifies additional headers, like From, Cc, and Bcc. Multiple headers should be separated with a CRLF (\r\n)

E.g. ErrorLog.php (See the log file C:\xampp\mailoutput)

Error Reporting

- The following will report all the errors:
  - ini_set('display_startup_errors',1);
  - ini_set('display_errors',1);
  - error_reporting(-1);

Exception Handling

- Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.
- Advantages:
  - The current code state is saved
  - The code execution will switch to a predefined (custom) exception handler function
  - Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code
- E.g. Exception1.php

Basic Use of Exceptions

- When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.
- If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

Try, throw and catch

- Proper exception code should include:
- **Try -** A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
- **Throw -** This is how you trigger an exception. Each "throw" must have at least one "catch"
- **Catch -** A "catch" block retrieves an exception and creates an object containing the exception information
- E.g. Exception2.php

Set a Top Level Exception Handler

- The set_exception_handler() function sets a user-defined function to handle all uncaught exceptions.
- E.g. topException.php
- **Rules for exceptions**
- Code may be surrounded in a try block, to help catch potential exceptions
- Each try block or "throw" must have at least one corresponding catch block
- Multiple catch blocks can be used to catch different classes of exceptions
- Exceptions can be thrown (or re-thrown) in a catch block within a try block
- **A simple rule:** If you throw something, you have to catch it.