Error handling and Debugging

Types of Errors:

- Syntactical – Errors in the syntax of the program
    1. Ex :- Forgetting a semicolon
- Run-time - Run-time errors include those things that don't stop a PHP script from executing but do stop the script from doing everything it was supposed to do.
    1. Ex :- calling a function using the wrong number or types of parameters
- Logical – out-and-out bugs, don't stop the execution of a script.
    1. Ex :- The inability to access the database
    2. When a SQL query doesn't return the records or returns the incorrect records

If display_ errors are *on* in your PHP configuration, PHP will show an error. If display_errors is off, you'll see a blank page.

Basic debugging steps:

- Make sure that you are running the right page.
- Make sure that you have saved your latest changes.
- Make sure that you run all PHP pages through the URL.
- Know what versions of PHP and MySQL you are running.
- Know what Web server you are running.
- Try executing pages in a different Web browser.
- If possible, try executing the page using a different Web server, version of PHP, and/or version of MySQL.

Debugging HTML:

- Check the source code
- Use a validation tool
- Use Firefox
- Use Firefox's add-on widgets
- Test the page in another browser

Displaying PHP errors:

- To turn on display_errors in a script, use the ini_set( ) function

  Ex: - ini_set('display_errors', 1);

- Including this line in a script will turn on display_errors for that script

Ex:
```
<head>
<title>Displaying Errors</title>
</head>
<body>
<h2>Testing Display Errors</h2>
<?php     # Script 8.1 - display_errors.php

// Show errors:
ini_set('display_errors', 1);

// Create errors:
foreach ($var as $v) {}
$result = 1/0;
?>
</body>
</html>
```

Adjusting Error Reporting in PHP:

Three kinds of error reports:

- Notices, which do not stop the execution of a script and may not necessarily, be a problem.
- Warnings, which indicate a problem but don't stop a script's execution.
- Errors, which stop a script from continuing

Table 8.1 Error-Reporting Levels

Number	Constant	Report On
1	**E_ERROR**	Fatal run-time errors (that stop execution of the script)
2	**E_WARNING**	Run-time warnings (non-fatal errors)
4	**E_PARSE**	Parse errors
8	**E_NOTICE**	Notices (things that could or could not be a problem)
256	**E_USER_ERROR**	User-generated error messages, generated by the **trigger_error( )** function
512	**E_USER_WARNING**	User-generated warnings, generated by the **trigger_error( )** function
1024	**E_USER_NOTICE**	User-generated notices, generated by the **trigger_error()** function
2048	**E_STRICT**	Recommendations for compatibility and interoperability
8192	**E_DEPRECATED**	Warnings about code that won't work in future versions of PHP
30719	**E_ALL**	All errors, warnings, and recommendations

Suppressing errors with @ :

Individual errors can be suppressed in PHP using the error suppression operator, @.

For example, if you don’t want PHP to report if it couldn’t include a file, you would code

```
@include ('config.inc.php');
```

Or if you don’t want to see a “division by zero” error:

```
$x = 8;
$y = 0;
$num = @ ($x/$y);
```

The @ symbol will work only on expressions, like function calls or mathematical operations.

You cannot use @ before conditionals, loops, function definitions, and so forth.

Example:

```
</head>
<body>
 <h2>Testing Error Reporting</h2>
 <?php        # Script 8.2 - report_errors.php

 // Show errors:
 ini_set('display_errors', 1);

 // Adjust error reporting:
 error_reporting(E_ALL | E_STRICT);

 // Create errors:
foreach ($var as $v) {}
 $result = 1/0;
```

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

If you give `error_reporting(0);` // Show no errors.

`error_reporting (E_ALL)` will tell PHP to report on

every error that occurs.

The trigger_error ( ) function is a way to programmatically generate

an error

Creating Custom error Handlers:

```
function report_errors (arguments)
 {
        // Do whatever here.
}
```

The PHP set_error_handler( ) function is used to name the user-defined function to be called when an error occurs.

```
set_error_handler ('report_errors');
```

```
function report_errors ($num, $msg,  $file, $line, $vars)
{
        // Do whatever here.
}
```

This function can be written to take up to five arguments.

1. an error number

2. a textual error message
3. the name of the file where the error was found
4. the specific line number on which it occurred
5. the variable that existed at the time of the error

```
<head>
	<title>Handling Errors</title>
</head>
<body>

<h2>Testing Error Handling</h2>
<?php	# Script 8.3 - handle_errors.php

	// Flag variable for site status:
define('LIVE', FALSE);

	// Create the error handler:
function my_error_handler ($e_number, $e_message, $e_file,
$e_line, $e_vars)
{
	// Build the error message:
$message = "An error occurred in script '$e_file' on line $e_line:
$e_message\n";

	// Append $e_vars to $message:
$message .= print_r ($e_vars, 1);

if (!LIVE)
{	// (print the error).
	echo '<pre>' . $message . "\n";
	debug_print_backtrace( );
	echo '</pre><br />';
} else
{	// Don't show the error.
```

```
echo  ' A system error occurred. We apologize for the
            inconvenience.<br />';
}

} // End of my_error_handler( ) definition.

// Use my error handler:
set_error_handler ('my_error_handler');

// Create errors:
foreach ($var as $v) {}
$result = 1/0;
?>

</body>
</html>
```

Logging PHP errors :

The error_log( ) function instructs PHP how to file an error

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

- The message value should be the text of the logged error.
- The type dictates how the error is logged. The options are the numbers 0, 1, 3, and 4:
    - use the computer's default logging method (0),
    - send it in an email (1),
    - write it to a text file (3),
    - or send it to the Web server's logging handler (4).
- The destination parameter can be either the name of a file (for log type 3) or an email address (for log type 1).

- The extra headers argument is used only when sending emails (log type 1).

PHP Debugging Techniques:

- End every statement with a semicolon.
- Balance all quotation marks, parentheses, curly braces, and square brackets
- Be consistent with your quotation marks
- Escape, using the backslash, all single and double-quotation marks within strings

To debug your scripts:

- Turn on display_errors.
- Use comments.
- Use the **print** and **echo** functions.
- Check what quotation marks are being used for printing variables.
- Track variables

Using die( ) and exit( ) :

When a **die( )** or **exit( )** is called in your script, the entire script is terminated. Both are useful for stopping a script from continuing should something important—like establishing a database connection— fail to happen.

SQL and MySQL Debugging Techniques:

The most common SQL errors are caused by the following issues:

- Unbalanced use of quotation marks or parentheses
- Unescaped apostrophes in column values
- Misspelling a column name, table name, or function
- Ambiguously referring to a column in a join
- Placing a query's clauses (**WHERE**, **GROUP BY**, **ORDER BY**, **LIMIT**) in the wrong order