**BASIC DEBUGGING**

**Error types & basic debugging**

PHP errors fall in to 3 general areas:

1. Syntactical – syntax error e.g. not inserting a semi colon
2. Run-time – these don’t stop PHP scripts running but does stop the script doing everything its supposed to. E.g. calling a function using the wrong number or types of parameters
3. Logical – not reported by PHP. These are not obvious bugs

SQL errors are normally a matter of syntax and they’ll be reported when you try to run the query in MySQL.

To begin debugging any problem:

* Make sure you’re running the right page
* Make sure you have saved your latest changes
* Make sure that you run all PHP pages through the URL
* Know what version of PHP & MySQL you are running (use phpinfo())
* Know what web server you are using
* Try executing pages in a different web browser
* If possible, try executing the page using a different; web server, version of PHP, and/or MySQL

\*Every web developer should have and use at least 2 web browsers\*

**Displaying PHP errors**

To display PHP error messages, you must turn on the *display\_errors* directive, either in an individual script or for the PHP configuration as a whole.

To turn on *display\_errors* in a script, use the ini\_set() function. As its arguments, this function takes a directive name & what setting that directive should have:

ini\_set(‘display\_errors’, 1);

The only downside is that if your script has a syntax error that prevents it from running at all, then you’ll still see a blank page.

**Adjusting error reporting in PHP**

As a rule of thumb – you’ll want PHP to report on any kind of error while you are developing a site, but report no specific errors once the site goes live.

Error\_reporting() – this function is used to establish what type of errors PHP should report on within a specific page. The function takes either a number or a constant.

|  |  |  |
| --- | --- | --- |
| **No.** | **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\_SCRIPT | Recommendations for compatibility and interoperability |
| 8192 | E\_DEPRECATED | Warnings about code that wont work in future versions of PHP |
| 30719 | E\_ALL | All errors, warnings and recommendations |

*A setting of 0 turns error reporting off entirely: error\_reporting(0);*

You can use | (OR), ~ (NOT), & (AND) with constant:

Error\_reporting(E\_ALL & ~ E\_NOTICE);

*This will display any non-notice errors*

\*See p253 on how to create own custom error handlers as opposed to default PHP ones\*

**PHP debugging techniques**

|  |  |
| --- | --- |
| **Error** | **Likely cause** |
| Blank page | HTML problem, or PHP error and display\_errors or error\_reporting is off |
| Parse error | Missing semicolon; unbalanced curly braces, parenthesis, or quotation marks; or use of an un-escaped quotation mark in a string |
| Empty variable value | Forgot the initial $; misspelled or miscapitalised the variable name, or inappropriate variable scope (with functions) |
| Undefined variable | Reference made to a variable before it is given a value or an empty variable value |
| Call to undefined function | Misspelled function name, PHP is not configured to use that function (like a MySQL function), or document that contains the function definition was not included |
| Cannot redeclare function | Two definitions of your own function exist; check within included files |
| Headers already sent | White spaces exist in the script before the PHP tags, data has already been printed, or a file has been included |

To debug your scripts:

* Turn on display\_errors
* Use comments
* Use print & echo functions – leave notes using these functions to see what is happening as the script is executed
* Check what quotation marks are being used for printing variables
* Track variables – make sure you are referring to the correct variable and that variable has the value you expected. Use print & echo to make sure – echo “<p>\$var is $var</p>”; - the first dollar sign is escaped so that the variables name is printed
* Print array values – the print\_r() & var\_dump() functions will print out their values without the need for loops – echo ‘<pre>’ . print\_r($var, 1) . ‘</pre>#’; // <pre> = preformatted tags

**SQL & MySQL debugging techniques**

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

* Unbalanced use of quotation marks or parenthesis
* Un-escaped 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

To debug your SQL queries:

* Print out any applicable queries in your PHP script
* Run the query in the MySQL client
* If the problem still isn’t evident, rewrite the query in its most basic form, and keep adding dimensions back in until you discover which clause is causing the problem

**Test**

1. How do you display errors in PHP?
2. What does the error\_reporting() function do?
3. Give examples of error\_reporting() using the table above
4. What 6 debugging techniques can you use when debugging PHP scripts?
5. What does print\_r() & var\_dump() do? Give example of each
6. How do you debug SQL queries?