SWINBURNE UNIVERSITY OF TECHNOLOGY

Advanced Web Development: Error Handling and Debugging

Week 11

Outline

- Study debugging concepts
- Handle and report errors
- Learn how to use basic debugging techniques

Reading: Textbook Appendix E

PHP: Error Handling and Logging

http://php.net/manual/en/book.errorfunc.php

Understanding Logic and Debugging

Basic Concepts

- **Logic** refers to the order in which various parts of a program run, or execute
- A **bug** is any error in a program that causes it to function incorrectly, because of incorrect syntax or flaws in logic
- **Debugging** refers to the act of tracing and resolving errors in a program

Some Tips to Mitigate Bugs

- Use good syntax such as ending statements with semicolons
- Initialize variables when you first declare them
- When creating functions and `for` statements, type the opening and closing braces before adding any code to the body of the structure
- Include the opening and closing parentheses and correct number of arguments when calling a function

Three Types of Errors

- **Syntax Errors**
  - **Syntax errors**, or **parse errors,** occur when the scripting engine fails to recognize code
  - Syntax errors can be caused by:
    - Incorrect use of PHP code
    - References to objects, methods, and variables that do not exist
    - Incorrectly spelled or mistyped words
  - Syntax errors in compiled languages, such as C++, are also called **compile-time errors**
- In PHP, Syntax Errors generate *Parse error messages*

Three Types of Errors (continued)

- **Run-Time Errors**
  - A **run-time error** occurs when the PHP scripting engine encounters a problem while a program is executing
  - Run-time errors do not necessarily represent PHP language errors
  - Run-time errors occur when the scripting engine encounters code that it cannot execute
- In PHP, depending on severity, Run-Time Errors generate either *Fatal error messages,* or *Warning messages,* or *Notice messages*

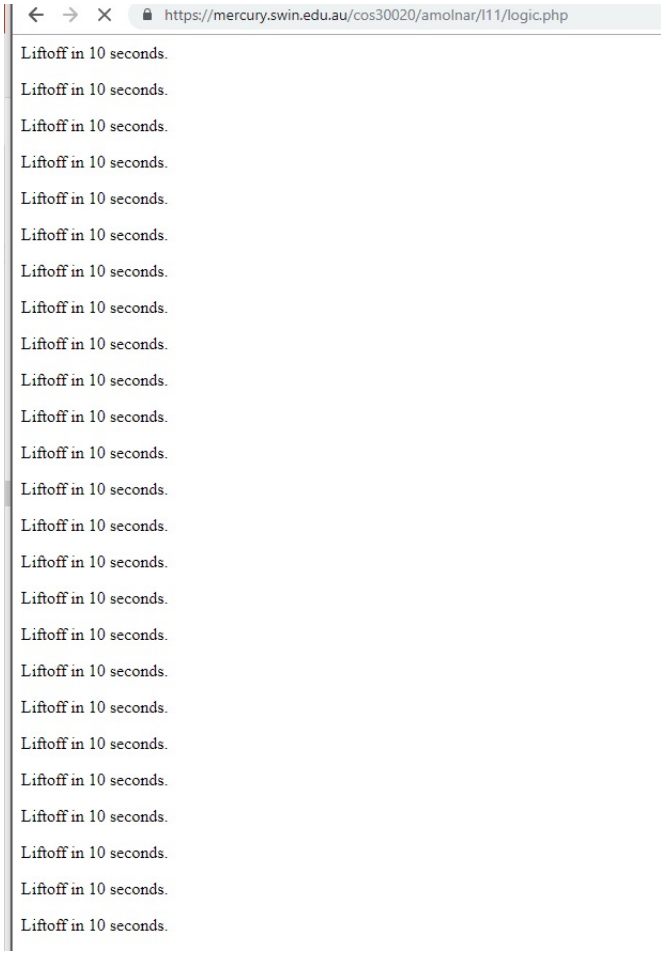
Three Types of Errors (continued)

- **Logic Errors**
  - **Logic errors** are flaws in a program's design that prevent the program from running as anticipated

```
for($count = 10; $count >= 0; $count) {
  if ($count == 0)
        echo "<p>We have liftoff!</p>";
  else
        echo "<p>Liftoff in $count seconds.</p>";
}
```

- Logic Errors do not generate error messages ☹

Three Types of Errors (continued)



The code in the browser

Handling and Reporting Errors

Four Types of Error Messages

- Parse error messages
- Fatal error messages
- Warning messages
- Notice messages

} *Syntax errors*

} *Run-time errors*

- Relationships with three types of errors
  - Parse error messages correspond to *Syntax errors*
  - The other three types of error messages correspond to *Run-time errors*.
  - *Logic errors* do not generate error messages.

Parse Error Messages

- **Parse error messages** occur when a PHP script contains a syntax error that prevents your script from running

```
<?php
for ($count = 10; $count >= 0; $count--)
  if ($count == 0){
        echo "<p>We have liftoff!</p>";
  } else {
        echo "<p>Liftoff in $count seconds.</p>";
  }
}
?>
```

Parse Error Messages (continued)

PHP parse error message in a Web browser

Parse Error Messages (continued)

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8" />
5    <meta name="description" content="COS30020 " />
6    <meta name="keywords" content="COS30020" />
7    <meta name="author" content="" />
8    <title>Parse Error</title>
9  </head>
10 <body>
11
12     <?php
13     for ($count = 10; $count >= 0; $count--)
14             if ($count == 0){
15                 echo "<p>We have liftoff!</p>";
16         } else {
17                 echo "<p>Liftoff in $count seconds.</p>";
18       }
19     }
20     ?>
21 </body>
22 </html>
```

Web page document illustrating line numbers

Fatal Error Messages

- **Fatal error messages** are raised when a script contains a run-time error that prevents it from executing

```
function beginCountdown() {
   for($count = 10; $count >= 0; $count--) {
          if ($count == 0) {
                 echo "<p>We have liftoff!</p>";
          } else {
                 echo "<p>Liftoff in $count seconds.</p>";
           }
   }
}
beginCntdown();
```

Fatal Error Messages (continued)

PHP fatal error message in a Web browser

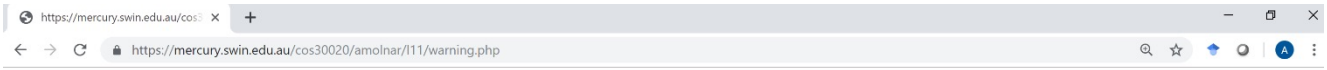
Warning Messages

- **Warning messages** are raised for run-time errors that do not prevent a script from executing
- A warning message occurs when you attempt to divide a number by 0
- A warning message occurs if you pass the wrong number of arguments to a function

Warning Messages (continued)

```
function beginCountdown($time) {
    if (!isset($time))
        $time = 10;
    for($count = $time; $count >= 0; $count--) {
        if ($count == 0) {
            echo "<p>We have liftoff!</p>";
        } else {
            echo "<p>Liftoff in $count seconds.</p>";
        }
    }
}
beginCountdown();
```

Warning Messages (continued)



Warning: Missing argument 1 for beginCountdown(), called in /home/staff/accounts/amolnar/cos30020/www/htdocs/l11/warning.php on line 13 and defined in **/home/staff/accounts/amolnar/cos30020/www/htdocs/l11/warning.php** on line **2**

Liftoff in 10 seconds.

Liftoff in 9 seconds.

Liftoff in 8 seconds.

Liftoff in 7 seconds.

Liftoff in 6 seconds.

Liftoff in 5 seconds.

Liftoff in 4 seconds.

Liftoff in 3 seconds.

Liftoff in 2 seconds.

Liftoff in 1 seconds.

We have liftoff!

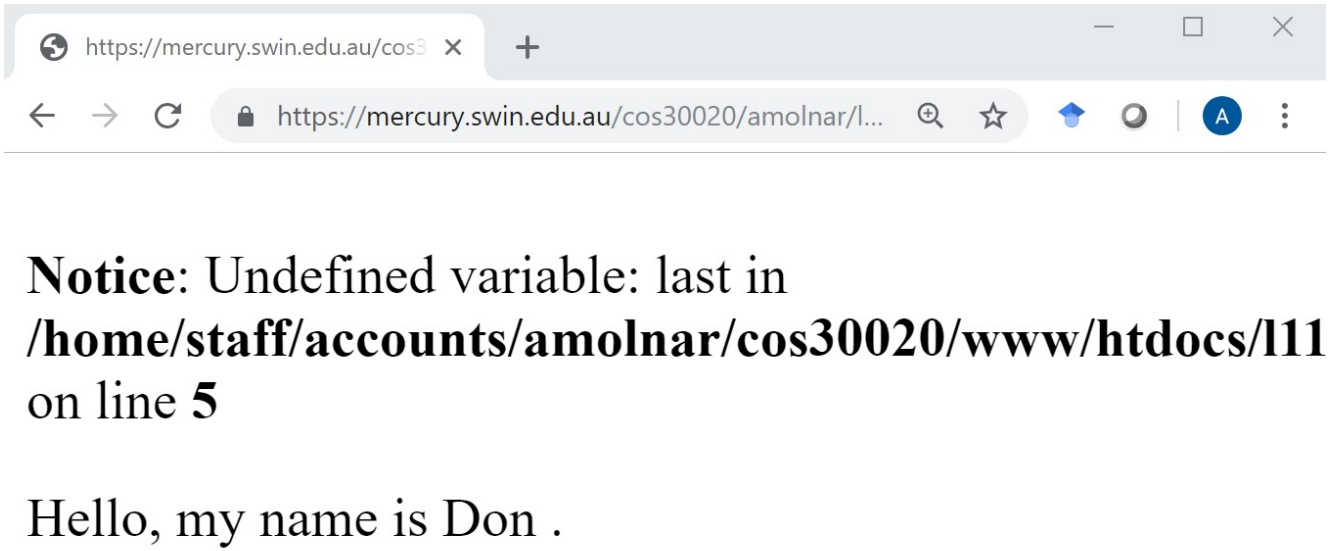
PHP warning message in a Web browser

Notice Messages

- **Notice messages** are raised for potential run-time errors that do not prevent a script from executing
- Notices are less severe than warnings and are typically raised when a script attempts to use an undeclared variable

```
$firstName = "Don";
$lastName = "Gosselin";
echo "<p>Hello, my name is $firstName $last.</p>";
```

Notice Messages (continued)



PHP notice message in a Web browser

Printing Errors to the Web Browser

- The php.ini configuration file contains two directives that determine whether error messages print to a Web browser:
  - `display_errors` directive prints script error messages and is assigned a value of "On"
  - `display_startup_errors` directive displays errors that occur when PHP first starts and is assigned a value of "Off"

Setting the Error Reporting Level

- The `error_reporting` directive in the php.ini configuration file determines which types of error messages PHP should generate
- By default, the `error_reporting` directive is assigned a value of "E_ALL," which generates all errors, warnings, and notices to the Web browser

php.ini

- php.ini

  Create a file called phpinfo.php with code:

  ```
  <?php phpinfo(); ?>
  ```

  Put it in your web directory on mercury ~/hddocs/

  https://mercury.swin.edu.au/cos30020/amolnar/phpinfo.php

Setting the Error Reporting Level (continued)

Error reporting levels

Constant	Integer	Description
--	0	Turns off all error reporting
E_ERROR	1	Reports fatal run-time errors
E_WARNING	2	Reports run-time warnings
E_PARSE	4	Reports syntax errors
E_NOTICE	8	Reports run-time notices
E_CORE_ERROR	16	Reports fatal errors that occur when PHP first starts
E_CORE_WARNING	32	Reports warnings that occur when PHP first starts
E_COMPILE_WARNING	32	Reports warnings generated by the Zend Scripting Engine

Setting the Error Reporting Level (continued)

Error reporting levels (continued)

Constant	Integer	Description
E_COMPILE_ERROR	64	Reports errors generated by the Zend Scripting Engine
E_USER_ERROR	256	Reports user-generated error messages
E_USER_WARNING	512	Reports user-generated warnings
E_USER_NOTICE	1024	Reports user-generated notices
E_ALL	2047	Reports errors, warnings, and notices with the exception of E_STRICT notices
E_STRICT	2048	Reports strict notices, which are code recommendations that ensure compatibility with PHP 5

Setting the Error Reporting Level (continued)

- To generate a combination of error levels, separate the levels assigned to the `error_reporting` directive with the bitwise `Or` operator (|):

```
error_reporting = E_ERROR | E_PARSE
```

- To specify that the `E_ALL` error should exclude certain types of messages, separate the levels with bitwise `And` (&) and `Not` operators (~)

```
error_reporting = E_ALL &~ E_NOTICE
```

Setting the Error Reporting Level (continued)

- Use `error_reporting()` function to specify the messages to report in a particular script.
- Place the function at the beginning of a script section.
  - `error_reporting(E_ALL &~ E_NOTICE);`
  - `error_reporting(E_ERROR | E_PARSE);`
  - `error_reporting(0)` to disable error messages

Logging Errors to a File

- PHP logs errors to a text file according to:
  - The error reporting level assigned to the `error_reporting` directive in the php.ini configuration file
  - What you set for an individual script with the `error_reporting()` function
- The `log_errors` directive determines whether PHP logs errors to a file and is assigned a default value of "Off"

Logging Errors to a File (continued)

- The `error_log` directive identifies the text file where PHP will log errors
  - `error_log = /usr/local/php5/logs/error.log`
- Assign either a path and filename or `syslog` to the `error_log` directive
- A value of `syslog`
  - On UNIX/Linux systems specifies that PHP should use the `syslog` protocol to forward the message to the system log file
  - On Windows systems a value of `syslog` forwards messages to the Event Log service

Writing Custom Error-Handling Functions

- Use the `set_error_handler()` function to specify a custom function to handle errors
  - Suppose `processErrors()` is a custom error-handling function, then `set_error_handler("processErrors")`
- Custom error-handling functions can only handle the following types of error reporting levels:
  - `E_WARNING`
  - `E_NOTICE`
  - `E_USER_ERROR`
  - `E_USER_WARNING`
  - `E_USER_NOTICE`

Writing Custom Error-Handling Functions

(continued)

- All other types of error reporting levels are handled by PHP's built-in error-handling functionality.
- Once you use `set_error_handler()`, none of PHP's default error-handling functionality executes for the preceding types of error reporting levels.
- To print the error message to the screen, you must include `echo()` statements in the custom error-handling function

Writing Custom Error-Handling Functions

(continued)

- The `switch` statement checks the value of the `$ErrLevel` parameter and then uses `echo()` statements to print the type of error message
- To log an error with a custom error-handling function, call the `error_log()` function by passing it a string containing the error message you want to log
  - `error_log($Log)`
- By default, `error_log()` logs the error message to the location specified by the `error_log` directive in the php.ini configuration file.

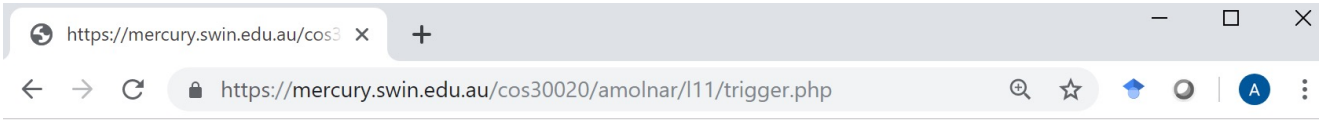
The `trigger_error()` Function

- Use the `trigger_error()` function to generate an error in your scripts
- The `trigger_error()` function accepts two arguments:
  - Pass a custom error message as the first argument and
  - Either the `E_USER_ERROR`, `E_USER_WARNING`, or `E_USER_NOTICE` error reporting levels as the second argument

The `trigger_error()` Function (continued)

```
if (isset($_GET["height"]) && isset($_GET["weight"])) {
      if (!is_numeric($_GET["weight"])
            || !is_numeric($_GET["height"])) {
            trigger_error("User did not enter numeric values",
                  E_USER_ERROR);
            exit();
      }
} else {
      trigger_error("User did not enter values", E_USER_ERROR);
}
$bodyMass = $_GET["weight"] / ($_GET["height"]
                  * $_GET["height"]) * 703;
printf("<p>Your body mass index is %d.</p>", $bodyMass);
```

The `trigger_error()` Function (continued)



Fatal error: User did not enter values in **/home/staff/accounts/amolnar/cos30020/www/htdocs/l11/trigger.php** on line **10**

Error generated by the `trigger_error()` function

Using Basic Debugging Techniques

Examining Your Code

- An **integrated development environment**, or **IDE**, is a software application used to develop other software applications.
- **IDE**s often use colors to identify different types of code. This is a valuable debugging technique.
- In a basic text editor, you can use `highlight_file()` function to print a color highlighted version of a file to a Web browser

Examining Your Code (continued)

- The `highlight_file()` function prints everything contained in the specified file including HTML elements and text

```
<?php
highlight_file("bodymassindex.php")
?>
```

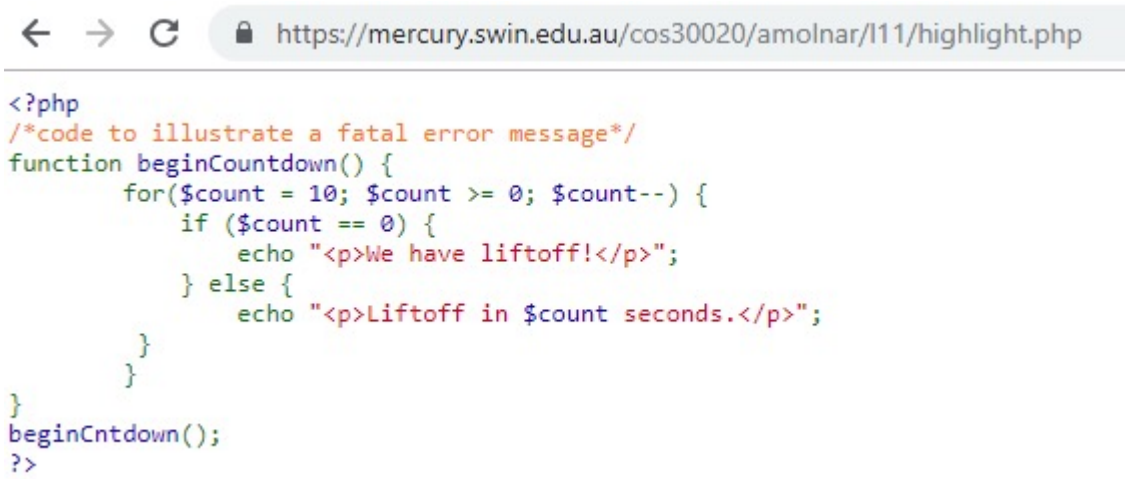
Examining Your Code (continued)

- By default, the `highlight_file()` function prints each of these elements with the following colors:
  - Code: blue
  - Strings: red
  - Comments: orange
  - Keywords: green
  - Page text (default color): black
  - Background color: white
  - HTML elements: black

Examining Your Code (continued)

- Change the default highlighting colors by modifying the following directives in the php.ini configuration file:
  - highlight.string = #DD0000
  - highlight.comment = #FF9900
  - highlight.keyword = #007700
  - highlight.default = #0000BB
  - highlight.bg = #FFFFFF
  - highlight.html = #000000

Examining Your Code (continued)



Output of the code presented in the fatal error section with the `highlight_file()` function

Tracing Errors with `echo()` Statements

- **Tracing** is the examination of individual statements in an executing program
- The `echo()` statement provides one of the most useful ways to trace PHP code
- Place an `echo()` method at different points in your program and use it to display the contents of a variable, an array, or the value returned from a function

Tracing Errors with `echo()` Statements

(continued)

```
function calculatePay() {
      $payRate = 15;
      $numHours = 40;
      $grossPay = $payRate * $numHours;
      $federalTaxes = $grossPay * .06794;
      $stateTaxes = $grossPay * .0476;
      $socialSecurity = $grossPay * .062;
      $medicare = $grossPay * .0145;
      $netPay = $grossPay - $federalTaxes;
      $netPay -= $stateTaxes;
      $netPay -= $socialSecurity;
      $netPay -= $medicare;
      return number_format($netPay, 2);
}
```

Tracing Errors with `echo()` Statements

(continued)

```
function calculatePay() {
      $payRate = 15; $numHours = 40;
      $grossPay = $payRate * $numHours;
echo "<p>$grossPay</p>";
      $federalTaxes = $grossPay * .06794;
      $stateTaxes = $grossPay * .0476;
      $socialSecurity = $grossPay * .062;
      $medicare = $grossPay * .0145;
      $netPay = $grossPay - $federalTaxes;
      $netPay -= $stateTaxes;
      $netPay -= $socialSecurity;
      $netPay -= $medicare;
      return number_format($netPay, 2);
}
```

Tracing Errors with `echo()` Statements

(continued)

- An alternative to using a single `echo()` statement is to place multiple `echo()` statements throughout your code to check values as the code executes
- When using `echo()` statements to trace bugs, it is helpful to use a driver program
- A **driver program** is a simplified, temporary program that is used for testing functions and other code

Tracing Errors with `echo()` Statements

(continued)

- A driver program is simply a PHP program that contains only the code being tested without having to worry about Web page elements, global variables and other code.
- **Stub functions** are empty functions that serve as placeholders (or "stubs") for a program's actual functions
- A stub function returns a hard-coded value that represents the result of the actual function

Using Comments to Locate Bugs

- Another method of locating bugs in a PHP program is to "comment out" problematic lines
- The cause of an error in a particular statement is often the result of an error in a preceding line of code

Using Comments to Locate Bugs (continued)

```
$amount = 100000;
$percentage = .08;
printf("<p>The interest rate or a loan in the amount of $%.2f
      is %s%%.<br />", $amount, $percentage * 100);
$yearlyInterest = $amount * $percentage;
// printf("The amount of interest for one year is $%.2f.<br />",
      $yearlyIntrest);
// $monthlyInterest = $yearlyIntrest / 12;
// printf("The amount of interest for one month is $%.2f.<br />",
      $monthlyInterest);
// $dailyInterest = $yearlyIntrest / 365;
// printf("The amount of interest for one day is $%.2f.</p>",
      $dailyInterest);
```

Combining Debugging Techniques

```
function calculatePay() {
      $payRate = 15; $numHours = 40;
      $grossPay = $payRate * $numHours;
echo "<p>$GrossPay</p>";
//    $stateTaxes = $grossPay * .0476;
//    $socialSecurity = $grossPay * .062;
//    $medicare = $grossPay * .0145;
//    $netPay = $grossPay - $federalTaxes;
//    $netPay -= $stateTaxes;
//    $netPay -= $socialSecurity;
//    $netPay -= $medicare;
//    return number_format($netPay, 2);
}
```

Analysing Logic

- Errors from Logic problems are difficult to spot using tracing techniques.
- When you suspect that your code contains logic errors, you must analyse each statement on a case-by-case basis

```
if (!isset($_GET['firstName']))
      echo "<p>You must enter your first name!</p>";
      exit();
echo "<p>Welcome to my Web site, " . $_GET["firstName"] ."!";
```

Analysing Logic (continued)

- For the code to execute properly, the `if` statement must include braces as follows:

```
if (!isset($_GET['firstName'])) {
      echo "<p>You must enter your first name!</p>";
      exit();
}
echo "<p>Welcome to my Web site, " . $_GET["firstName"] .
   "!</p>";
```

Analysing Logic (continued)

- The following for statement shows another example of an easily overlooked logic error:

```
for ($count = 1; $count < 6; $count++);
     echo "$count<br />";
```

Summary

- Logic refers to the order in which various parts of a program run, or execute
- A bug is any error in a program that causes it to function incorrectly, because of incorrect syntax or flaws in logic
- Debugging refers to the act of tracing and resolving errors in a program
- Syntax errors, or parse errors, occur when the scripting engine fails to recognise code

Summary (continued)

- A run-time error occurs when the PHP scripting engine encounters a problem while a program is executing
- Logic errors are flaws in a program's design that prevent the program from running as anticipated
- Parse error messages occur when a PHP script contains a syntax error that prevents your script from running
- Fatal error messages are raised when a script contains a run-time error that prevents it from executing

Summary (continued)

- Warning messages are raised for run-time errors that do not prevent a script from executing
- Notice messages are raised for potential run-time errors that do not prevent a script from executing
- Tracing is the examination of individual statements in an executing program
- A driver program is a simplified, temporary program that is used for testing functions and other code