

ERROR HANDLING AND DEBUGGING

MIS 4530
DR. GARRETT

Understanding Logic and Debugging

2

- **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

Understanding Logic and Debugging (continued)

3

- To avoid 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

Syntax Errors

4

- **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**

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

Logic Errors

6

- **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>";  
}
```

Handling and Reporting Errors

7

- **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>";
}
?>
```

Handling and Reporting Errors (continued)

8

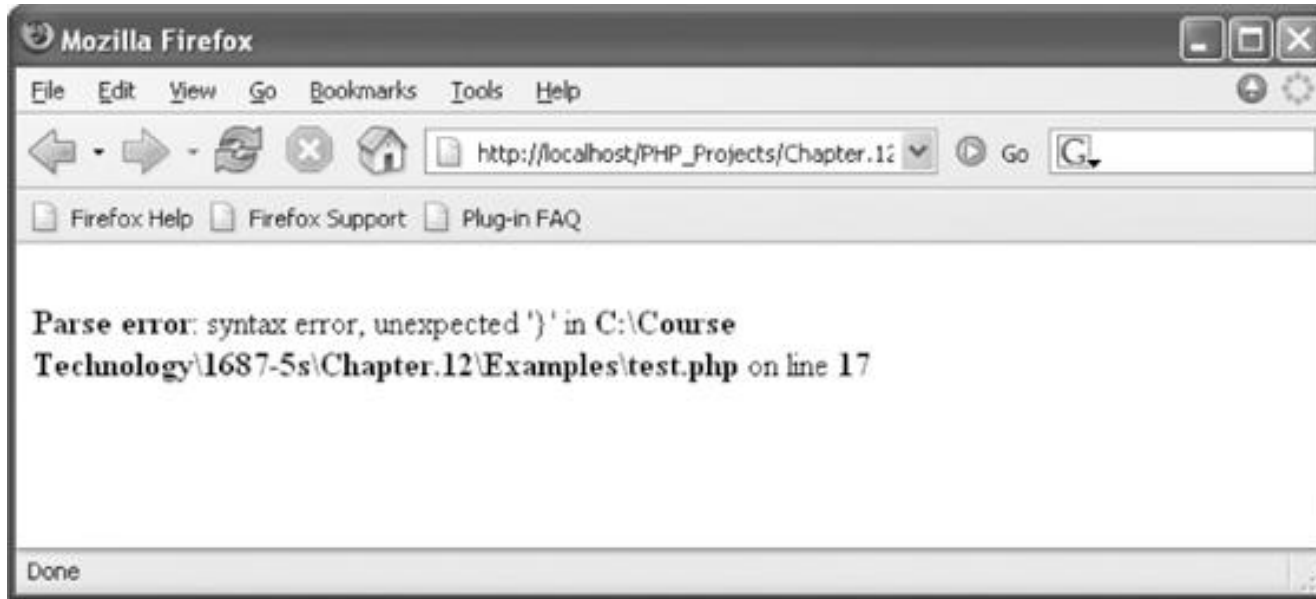


Figure 12-2 PHP parse error message in a Web browser

Handling and Reporting Errors (continued)

9

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

Handling and Reporting Errors (continued)

10

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

```
$FirstName = "Don";
```

```
$LastName = "Gosselin";
```

```
echo "<p>Hello, my name is $FirstName $Last.</p>";
```

Printing Errors to the Web Browser

11

- 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

12

- 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

Setting the Error Reporting Level (continued)

13

Table 12-1 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)

14

Table 12-1 Error reporting levels (continued)

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

Setting the Error Reporting Level (continued)

15

- 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
```

Logging Errors to a File

16

- 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)

17

- The `error_log` directive identifies the text file where PHP will log errors
- 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

18

- Use the `set_error_handler()` function to specify a custom function to handle errors
- 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)

19

- To print the error message to the screen, you must include `echo ()` statements in the custom error-handling function
- 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

Raising Errors with the `trigger_error()` Function

20

- 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)

21

```
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)

22

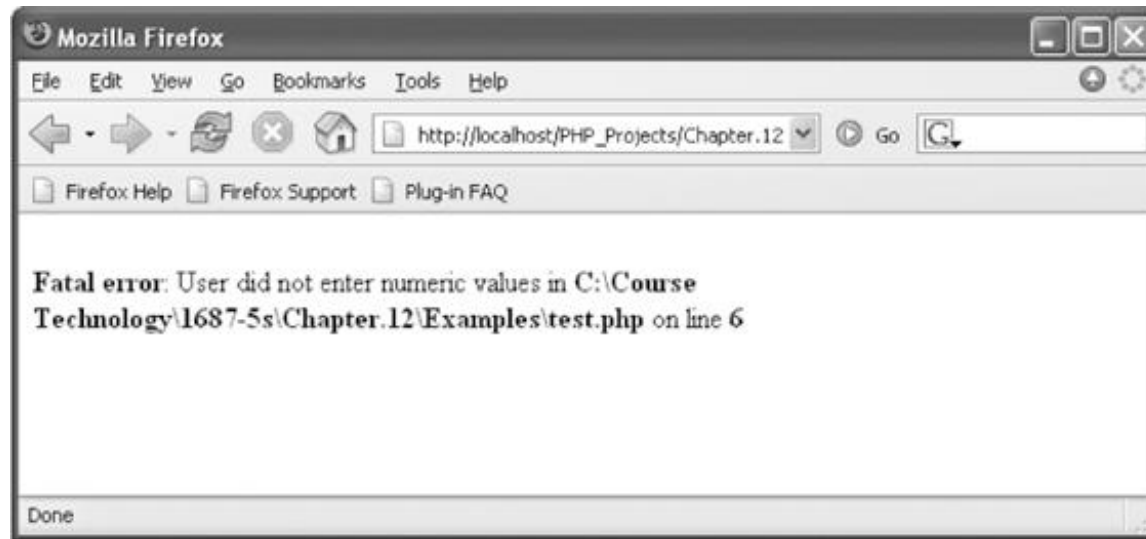


Figure 12-10 Error generated by the `trigger_error()` function

Examining Your Code

23

- An **integrated development environment**, or **IDE**, is a software application used to develop other software applications
- The `highlight_file()` function prints a color highlighted version of a file to a Web browser
- The `highlight_file()` function prints everything contained in the specified file including HTML elements and text

Examining Your Code (continued)

24

- 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)

25

- 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)

26

```
<?php
// The following statements sets the error handler to the processErrors() function
set_error_handler("processErrors");
function processErrors($ErrLevel, $ErrorMessage, $File, $LineNum) {
    $Message = "";
    switch ($ErrLevel) {
        case E_WARNING:
            $Message = "<p><strong>PHP Warning</strong>: $ErrorMessage<br />";
            break;
        case E_NOTICE:
            $Message = "<p><strong>PHP Notice</strong>: $ErrorMessage<br />";
            break;
        case E_USER_ERROR:
            $Message = "<p><strong>User Error</strong>: $ErrorMessage<br />";
            break;
        case E_USER_WARNING:
            $Message = "<p><strong>User Warning</strong>: $ErrorMessage<br />";
            break;
        case E_USER_NOTICE:
            $Message = "<p><strong>User Notice</strong>: $ErrorMessage<br />";
            break;
    }
    $Log = strip_tags($Message);
    $Message .= "<strong>Filename</strong>: $File<br />";
    $Message .= "<strong>Line Number</strong>: $LineNum</p>";
    $Log .= " in $File on line $LineNum";
    echo $Message;
}
```

Comments in orange

Code in blue

Strings in red

Keywords in green

Figure 12-11 Output of a script with the `highlight_file()` function

Examining Your Code (continued)

27

```
<?php
class MovingEstimator (
    private $MileageCost = 0;
    private $LaborCost = 0;
    private $FlightsCost = 0;
    private $AppliancesCost = 0;
    private $PianosCost = 0;
    private $TotalEstimate = 0;
    function __construct() (
        if (isset($_GET[distance]))
            $this->MileageCost = $_GET['distance'];
        if (isset($_GET[weight]))
            $this->LaborCost = $_GET['weight'];
        if (isset($_GET[flights]))
            $this->FlightsCost = $_GET['flights'];
        if (isset($_GET[appliances]))
            $this->AppliancesCost = $_GET['appliances'];
        if (isset($_GET[pianos]))
            $this->PianosCost = $_GET['pianos'];
    )
    function calcTotalEstimate() (
        $this->TotalEstimate = $this->MileageCost;
        $this->TotalEstimate += $this->LaborCost;
```

Code in blue

Strings in red

Keywords in green

Figure 12-12 Output of the `MovingEstimator` class with the `highlight_file()` function

Tracing Errors with `echo ()` Statements

28

- **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)

29

```
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)

30

```
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)

31

- 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)

32

- A driver program is a PHP program that contains only the code being tested
- **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

33

- 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)

34

```
$Amount = 100000;
$Percentage = .08;
printf("<p>The interest rate on 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 />",
        $YearlyInterest);
// $MonthlyInterest = $YearlyInterest / 12;
// printf("The amount of interest for one month is $%.2f.<br />",
        $MonthlyInterest);
// $DailyInterest = $YearlyInterest / 365;
// printf("The amount of interest for one day is $%.2f.</p>",
        $DailyInterest);
```

Combining Debugging Techniques

35

```
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);
}
```

Analyzing Logic

36

- When you suspect that your code contains logic errors, you must analyze 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'] . "!";
```

Analyzing Logic (continued)

37

- 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'] .
    "!";
```

Analyzing Logic (continued)

38

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

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