**PHP Try Catch Example: Exception & Error Handling Tutorial**

**What is an Exception?**

An error is an unexpected program result that cannot be handled by the program itself.

Errors are resolved by fixing the program. An example of an error would be an infinite loop that never stops executing.

An exception is unexpected program result that can be handled by the program itself.

Examples of exception include trying to open a file that does not exist.

This exception can be handled by either creating the file or presenting the user with an option of searching for the file.

**In this tutorial, you will learn-**

* [Why handle exception?](https://www.guru99.com/error-handling-and-exceptions.html#1)
* [PHP Error handling](https://www.guru99.com/error-handling-and-exceptions.html#2)
* [Error handling examples](https://www.guru99.com/error-handling-and-exceptions.html#3)
* [Difference between Errors and Exception](https://www.guru99.com/error-handling-and-exceptions.html#4)
* [Multiple Exceptions](https://www.guru99.com/error-handling-and-exceptions.html#5)
* [Testing the code](https://www.guru99.com/error-handling-and-exceptions.html#6)

**Why handle exception?**

* Avoid unexpected results on our pages which can be very annoying or irritating to our end users
* Improve the security of our applications by not exposing information which malicious users may use to attack our applications
* Php Exceptions are used to change the normal flow of a program if any predictable error occurs.

**PHP Error handling**

When an error occurs, depending on your configuration settings, PHP displays the error message in the web browser with information relating to the error that occurred.

  PHP offers a number of ways to handle errors.

We are going to look at three (3) commonly used methods;

1. **Die statements**– the die function combines the echo and exit function in one. It is very useful when we want to output a message and stop the script execution when an error occurs.

1. **Custom error handlers** – these are user defined functions that are called whenever an error occurs.

1. **PHP error reporting** – the error message depending on your PHP error reporting settings. This method is very useful in development environment when you have no idea what caused the error. The information displayed can help you debug your application.

**Error handling examples**

Let’s now look at some simple examples with error handling routines.

Let’s suppose that we have developed an application that uses text files to store data. We might want to check for the file’s existence before we attempt to read data from it.

**The code below implements the above example.**

<?php

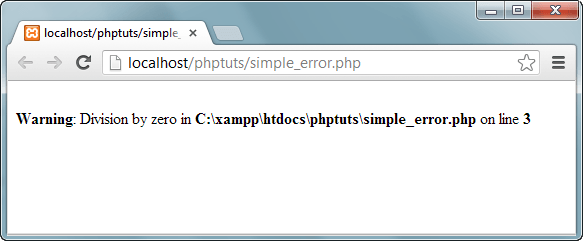
$denominator = 0;

echo 2 / $denominator;

?>

Assuming you saved the file simple\_error.php in phptuts folder, open the URL **http://localhost/phptuts/simple\_error.php**

You will get the following results

[](https://www.guru99.com/images/2013/04/error.png)

As you can see from the above results, it makes our application look unprofessional and can be annoying to the user.

We will modify the above code and write an error handler for the application

<?php

$denominator = 0;

if ($denominator != 0) {

echo 2 / $denominator;

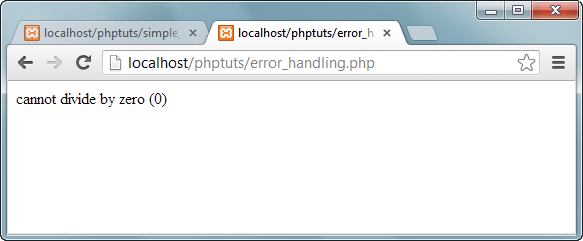
} else {

echo "cannot divide by zero (0)";

}

?>

Assuming you saved the above code as error\_handling.php, open the URL **http://localhost/phptuts/error\_handling.php**

[](https://www.guru99.com/images/2013/04/division_by_zero_error.png)

*Note: it’s a good security practice to display a message as the one shown above instead of showing the message like “File not found”.*

Let’s look at another example that uses a custom error handler.

The custom error handler will be set as the default PHP error handling function and will basically display an error number and message.

The code below illustrates the implementation of the above example

<?php

function my\_error\_handler($error\_no, $error\_msg)

{

echo "Opps, something went wrong:";

echo "Error number: [$error\_no]";

echo "Error Description: [$error\_msg]";

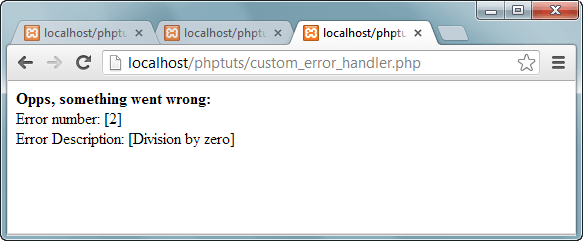
}

set\_error\_handler("my\_error\_handler");

echo (5 / 0);

?>

  Open the URL **http://localhost/phptuts/custom\_error\_handler.php** you will get the following results

. [](https://www.guru99.com/images/2013/04/custom_error_handler.png)

As you can see from the above example, custom error handlers are powerful in the sense that

* They allow us to customize the error messages.
* The custom error handler can also include error logging in a file/database, emailing the developer etc.

Let’s now look at the third type of error handling. We will be using the PHP built in function error\_reporting function. It has the following basic syntax

<?php

error\_reporting($reporting\_level);

?>

  HERE,

* “error\_reporting” is the PHP error reporting function
* “$reporting\_level" is optional, can be used to set the reporting level. If no reporting level has been specified, PHP will use the default error reporting level as specified in the php.ini file.

| **Reporting Level** | **Description** | **Example** |
| --- | --- | --- |
| E\_WARNING | Displays warning messages only. Does not halt the execution of the script | error\_reporting(E\_WARNING); |
| E\_NOTICE | Displays notices that can occur during normal execution of a program or could be an error. | error\_reporting(E\_ NOTICE); |
| E\_USER\_ERROR | Displays user generated errors i.e. custom error handler | error\_reporting(E\_ USER\_ERROR); |
| E\_USER\_WARNING | Displays user generated warning messages | error\_reporting(E\_USER\_WARNING); |
| E\_USER\_NOTICE | Displays user generated notices | error\_reporting(E\_USER\_NOTICE); |
| E\_RECOVERABLE\_ERROR | Displays error that are not fatal and can be handled using custom error handlers | error\_reporting(E\_RECOVERABLE\_ERROR); |
| E\_ALL | Displays all errors and warnings | error\_reporting(E\_ ALL); |

**Difference between Errors and Exception**

* Exceptions are thrown and intended to be caught while errors are generally irrecoverable.
* Exceptions are handled in an object oriented way.

This means when an exception is thrown; an exception object is created that contains the exception details.

The table below shows the exception object methods

| **Method** | **Description** | **Example** |
| --- | --- | --- |
| getMessage() | Displays the exception’s message | <?php  echo $e->getMessage();  ?> |
| getCode() | Displays the numeric code that represents the exception | <?php  echo $e->getCode();  ?> |
| getFile() | Displays the file name and path where the exception occurred | <?php  echo $e->getFile();  ?> |
| getLine() | Displays the line number where the exception occurred | <?php  echo $e->getLine();  ?> |
| getTrace() | Displays an array of the backtrace before the exception | <?php  print\_r( $e->getTrace());  ?> |
| getPrevious() | Displays the previous exception before the current one | <?php  echo $e->getPrevious();  ?> |
| getTraceAsString() | Displays the backtrace of the exception as a string instead of an array | <?php  echo $e->getTraceAsString();  ?> |
| \_\_toString() | Displays the entire exception as a string | <?php  echo $e->\_\_toString();  ?> |

  Below is the basic syntax for throwing an exception.

<?php

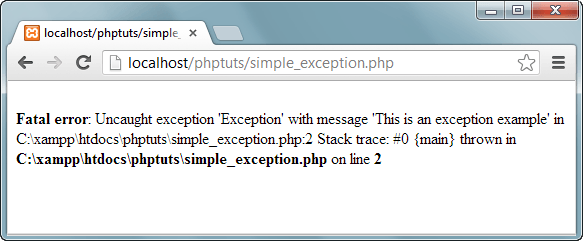
throw new Exception("This is an exception example");

?>

HERE,

* “throw” is the keyword used to throw the exception
* “new Exception(…)” creates an exception object and passes “This is an exception example “ string as the message parameter.

The above code outputs the following message.

[](https://www.guru99.com/images/2013/04/uncaught_exception.png)

We are now going to look at an example that implements the throw and catch exceptions.

We will modify the above example and include the try, throw and catch.

It has the following basic syntax.

<?php

try {

//code goes here that could potentially throw an exception

}

catch (Exception $e) {

//exception handling code goes here

}

?>

  HERE,

* “try{…}” is the block of code to be executed that could potentially raise an exception
* “catch(Exception $e){…}” is the block of code that catches the thrown exception and assigns the exception object to the variable $e.

The code below shows the basic exception example with the try, throw and catch exception implemented.

The program deliberately throws an exception which it then catches.

<?php

try {

$var\_msg = "This is an exception example";

throw new Exception($var\_msg);

}

catch (Exception $e) {

echo "Message: " . $e->getMessage();

echo "";

echo "getCode(): " . $e->getCode();

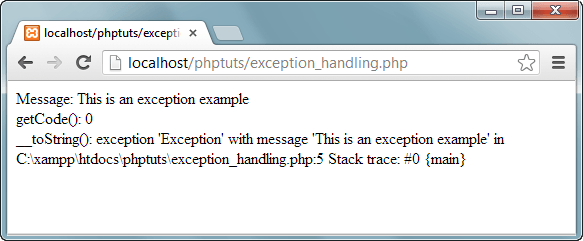
echo "";

echo "\_\_toString(): " . $e->\_\_toString();

}

?>

  Open the URL **http://localhost/phptuts/exception\_handling.php** You will get the following results.

[](https://www.guru99.com/images/2013/04/exception_info.png)

It’s also possible to create multiple exceptions for one php try statement depending on the type of exception thrown.

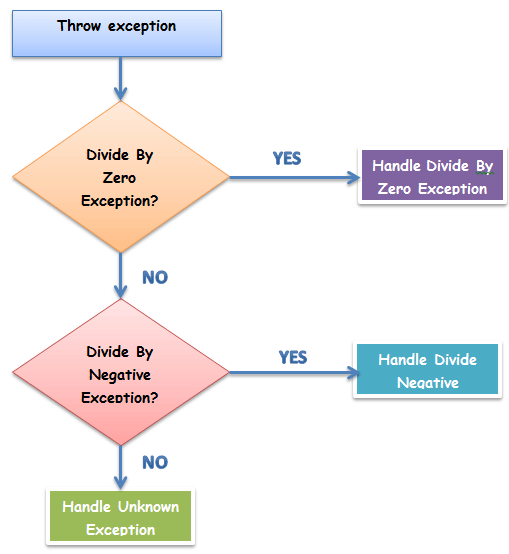
See the article on MySQL, PHP data access… for implementation examples of multiple exceptions

**Multiple Exceptions**

Multiple exception use multiple try catch blocks to handle the thrown exceptions. Multiple exceptions are useful when;

* You want to display a customized message depending on the exception thrown
* You want to perform a unique operation depending on the exception thrown

The flowchart below illustrates the how multiple exceptions work

[](https://www.guru99.com/images/2013/04/exception_flowchart.png)

Let’s look at an example that uses multiple exceptions.

We will modify the code that divides a number by the passed in denominator.

We expect two types of exceptions to occur;

* Division by zero
* Division by a negative number

For simplicity’s sake, we will only display the exception type in our catch blocks.

The PHP built in Exception class is used to throw exceptions.

We will create two classes that extend the exception class and use them to throw exceptions.

The code below shows the implementation.

<?php

class DivideByZeroException extends Exception {};

class DivideByNegativeException extends Exception {};

function process($denominator)

{

try

{

if ($denominator == 0)

{

throw new DivideByZeroException();

}

else if ($denominator < 0)

{

throw new DivideByNegativeException();

}

else

{

echo 25 / $denominator;

}

}

catch (DivideByZeroException $ex)

{

echo "DIVIDE BY ZERO EXCEPTION!";

}

catch (DivideByNegativeException $ex)

{

echo "DIVIDE BY NEGATIVE NUMBER EXCEPTION!";

}

catch (Exception $x)

{

echo "UNKNOWN EXCEPTION!";

}

}

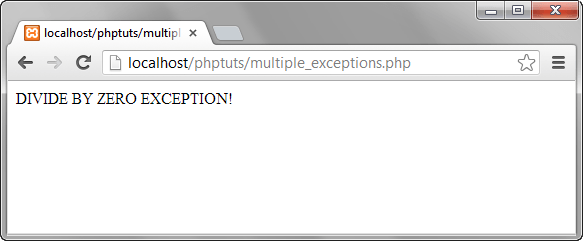
process(0);

?>

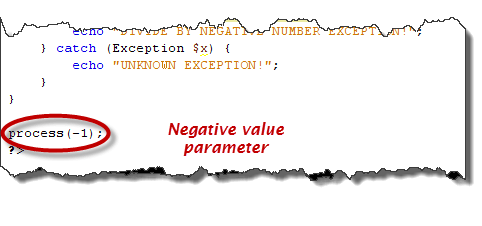
**Testing the code**

We will assume you saved multiple\_exceptions.php in phptuts folder.

Browse to the URL **http://localhost/phptuts/multiple\_exceptions.php**

[](https://www.guru99.com/images/2013/04/division_by_zero_exception.png)

Switch back to the PHP file and pass -1 as the parameter as shown in the following diagram.

[](https://www.guru99.com/images/2013/04/negative_value_exception.png)

Browse to the URL **http://localhost/phptuts/multiple\_exceptions.php**.

 What results do you get? Pass 3 as the parameter.

What results do you get?

**Summary**

* Errors are unexpected results produced by PHP code
* Error handling improves the application performance
* PHP has built in functions that can be used to customize the way PHP reports errors
* Exceptions are like errors, but they can be caught using the catch block when thrown.
* Displaying error messages that show error information is considered a bad security practice.