**Exception Handling**

An exception is an object that describes an error or unexpected behaviour of a PHP script. Exceptions are thrown by many PHP functions and classes. User defined functions and classes can also throw exceptions.

Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs.

### Syntax:

### new Exception(message, code, previous)

**i) message:** Optional. A string describing why the exception was thrown.

**ii) code:** Optional. An integer that can be used to easily distinguish this exception from others of the same type.

**iii) previous:** Optional. If this exception was thrown in a catch block of another exception, it is recommended to pass that exception into this parameter.

### Methods:

When catching an exception, the following methods that can be used to get information about the exception:

**getMessage():** Returns a string describing why the exception was thrown.

**getPrevious() :** If this exception was triggered by another one, this method returns the previous exception. If not, then it returns *null.*

**getCode():**Returns the exception code.

**getFile():**Returns the full path of the file in which the exception was thrown.

**getLine():**Returns the line number of the line of code which threw the exception.

**Key words used for Exception Handling:**

**i) try:** It represent block of code in which exception can arise.

**ii) catch:** The catch block is a block of code which indicates what type of exception should be caught and the name of the variable which can be used to access the exception.. It creates an object containing the exception information.

**iii) throw:** It is used to throw user defined function or method to throw an exception. It is also used to list the exceptions that a function throws, but doesn’t handle itself.

**iv) finally:** It is used in place of catch block or after catch block basically it is put for cleanup activity in PHP code. Code in the finally block will always run regardless of whether an exception was caught. If finally is present, the catch block is optional

**Program-1: (throw an exception without catching it)**

### <?php function division($a, $b)

### {   if($b == 0)

### {     throw new Exception("Division by zero");   }   return $a / $b; }  division(5, 0); ?>

If an exception is not caught, a fatal error will occur with an "Uncaught Exception" message.

## try...catch Statement

To avoid the error from the example above, we can use the try...catch statement to catch exceptions and continue the process.

Syntax:

try

{  
  code that can throw exceptions  
} catch(Exception $e)

{  
  code that runs when an exception is caught  
}

**Program-2: (try and catch)**

### <?php function division($a, $b)

### {   if($b == 0)

### {     throw new Exception("Division by zero");   }   return $a / $b; } try

### {    division(5, 0); } catch(Exception $e)

### {   echo "Unable to divide."; } ?>

### Syntax:

### try

### {   code that can throw exceptions } catch(Exception $e)

### {   code that runs when an exception is caught }

### finally

### {   code that always runs regardless of whether an exception was caught }

**Program-3: (try, catch and finally)**

<?php  
function division($a, $b)

### {   if($b == 0)

### {     throw new Exception("Division by zero");   }   return $a / $b; } try

### {   echo division(5, 0); } catch(Exception $e)

### {   echo "Unable to divide. "; } finally

### {   echo "Process complete."; } ?>

**Program-3: (use of methods)**

<?php  
function division($a, $b) {  
  if($b == 0) {  
    throw new Exception("Division by zero", 1);  
  }  
  return $a / $b;  
}  
  
try {  
   division(5, 0);  
} catch(Exception $x)

{  
  $c = $x->getCode();  
  $m = $x->getMessage();  
  $f = $x->getFile();  
  $l = $x->getLine();  
  echo "Exception thrown in $f on line $l: [Code $c] $m";

}  
?>