PHP Try...Catch

In PHP, the keywords **try, catch, throw** and **finally** are provided to deal with exceptions. Whereas an Error is an unexpected program result, which cannot be handled by the program itself and the program has to be terminated with die() or setting a custom error handler.

On the other hand, an exception refers to an unexpected situation which can be handled in such a way that the program may keep running after throwing the exception out of its normal flow.

An exception can be thrown, and caught with the catch keyword within PHP code. A code block which is potentially prone to exception is surrounded by a **try** block. Each **try** must have at least one corresponding catch or finally block.

Try, Throw, Catch, and Finally

The four exception related keywords have the following role to play –

- **Try** – A block of code where some exception is likely to occur is placed in "try" block. If exception is not triggered, the code continues execution. However, if exception does occur, it is "thrown". The execution is halted and PHP looks for matching "catch" block. If the exception is not caught, PHP issues a Fatal Error.
- **Throw** – Here is how you trigger an exception. Each "throw" must have at least one "catch" or "finally" block.
- **Catch** – a block that retrieves an exception and creates an object containing the exception information. Multiple catch blocks can be used to catch different exceptions.
- **Finally** – Code within a finally block is always executed after throw or catch block.

Example

Here is an example of exception handling technique. The code renders two text fields on the browser and asks the user to enter two numbers for their division to be performed. If the second number (denominator) is 0, an exception is thrown and the program enters the catch block and prints the exception message. Otherwise the result of division is displayed.

```
<html>
<body>
   <form action="<?php echo $_SERVER['PHP_SELF'];?>" method="post">
```

```
      <h3>First No: <input type="text" name="first"/></h3>
      <h3>Second No: <input type="text" name="second"/></h3>
      <input type="submit" value="Submit" />
   </form>

   <?php
      if ($_SERVER["REQUEST_METHOD"] == "POST") {
         $x = $_POST['first'];
         $y = $_POST['second'];
         echo "$x $y";
         try {
            if ($y == 0) {
               throw new Exception("Division by Zero");
            }
            $z = $x/$y;
            echo "<h3>x = $x y = $y Division = $z<br>";
         }
         catch (Exception $e) {
            echo "<h3> Exception: " . $e->getMessage();
         }
      }
   ?>
</body>
</html>
```

It will produce the following **output** −

```
Case 1: x = 10 y = 5 Division = 2

Case 2: x = 10 y = 0
Exception: Division by Zero
```

The Exception Class

PHP throws an object of **Exception class**. In PHP, Exception class is the base for user exceptions. It implements throwable interface.

This class defines the following methods −

getMessage()

This function returns the Exception message as a string −

```
final public Exception::getMessage(): string
```

getCode()

This function returns the exception code as **int** in Exception −

```
final public Exception::getCode(): int
```

Take a look at the following **example** −

```
try {
   throw new Exception("Some error message", 30);
}
catch(Exception $e) {
   echo "The exception code is: " . $e->getCode();
}
```

getFile()

This function returns the filename in which the exception was created −

```
final public Exception::getFile(): string
```

Take a look at the following **example** −

```
try {
   if ($y == 0) {
      throw new Exception("Division by Zero");
   }
   $z = $x/$y;
   echo "<h3>x = $x y = $y Division = $z<br>";
}
catch (Exception $e) {
   echo "<h3> Exception: " . $e->getMessage(). " in " . $e->getFile();
}
```

It will produce the following **output** −

```
Exception: Division by Zero in C:\xampp\htdocs\hello.php
```

getLine()

This function returns the line number where the exception was created −

```
final public Exception::getLine(): int
```

Example

Take a look at the following example −

```
<?php
   if ($_SERVER["REQUEST_METHOD"] == "POST") {
      $x = $_POST['first'];
      $y = $_POST['second'];
      echo "$x $y";
      try {
         if ($y == 0) {
            throw new Exception("Division by Zero");
         }
         $z = $x/$y;
         echo "<h3>x = $x y = $y Division = $z<br>";
      }
      catch (Exception $e) {
         echo "<h3> Exception: " . $e->getMessage(). " in " . $e->getLine() . " of
      }
   }
?>
```

It will produce the following **output** −

```
Exception: Division by Zero in 21 of C:\xampp\htdocs\hello.php
```


Multiple Catch Blocks

PHP allows a series of catch blocks following a try block to handle different exception cases. Multiple **catch** blocks may be employed to handle predefined exceptions and errors as well as user defined exceptions.

Example

The following example uses **catch** blocks to process DivisioByZeroError, TypeError, ArgumentCountError and InvalidArgumentException conditions. There is also a **catch** block to handle general Exception.

```
<?php
   declare(strict_types=1);
   function divide(int $a, int $b) : int {
      return $a / $b;
   }
   $a=10;
   $b=0;
   try {
   if (!$b) {
      throw new DivisionByZeroError('Division by zero.');
      if (is_int($a)==FALSE || is_int($b)==FALSE)
      throw new InvalidArgumentException("Invalid type of arguments");
      $result=divide($a, $b);
      echo $result;
   }

   // if argument types not matching
   catch (TypeError $x) {
      echo $x->getMessage();
   }

   // if denominator is 0
   catch (DivisionByZeroError $y) {
      echo $y->getMessage();
   }

   // if number of arguments not equal to 2
   catch (ArgumentCountError $z) {
      echo $z->getMessage();
   }

   // if argument types not matching
   catch (InvalidArgumentException $i) {
      echo $i->getMessage();
   }
```

```
   // any uncaught exception
   catch (Exception $ex) {
      echo $ex->getMessage();
   }
?>
```

To begin with, since denominator is 0, "divide by 0" error will be displayed −

```
Division by 0
```

Set **$b=3** which will cause TypeError because divide function is expected to return integer but division results in **float**.

```
divide(): Return value must be of type int, float returned
```

If just one variable is passed to divide function by changing **$res=divide($a);** this will result in an **ArgumentCountError** −

```
Too few arguments to function divide(), 1 passed in C:\xampp\htdocs\hello.php on line 1
```

If one of arguments is not integer, it is a case of **InvalidArgumentException**. Change **$b** to a string −

```
Invalid type of arguments
```

The Finally Block

A **finally** block may also be specified after or instead of **catch** blocks. Code within the **finally** block will always be executed after the **try** and **catch** blocks, regardless of whether an exception has been thrown, and before normal execution resumes.

```
try {
   if ($y == 0) {
      throw new Exception("Division by Zero");
   }
   $z = $x/$y;
   echo "<h3>x = $x y = $y Division = $z </h3><br>";
}
catch (Exception $e) {
```

```
   echo "<h3> Exception: " . $e->getMessage(). "</h3>";
}
finally {
   echo "<h3>End of try - catch - finally</h3>";
}
```

It will produce the following output –

Case 1 –

```
x = 10 y = 5 Division = 2
End of try - catch – finally
```

Case 2 –

```
X=10 y=0
Exception: Division by Zero
End of try - catch – finally
```

Finally With Return

There is a peculiar behaviour of finally block when either **try** block or **catch** block (or both) contain a **return** statement. Normally a **return** statement causes the control of program to go back to the calling position. However, in case of a function with try/catch block with **return**, the statements in **finally** block are executed first before returning.

Example

In the following example, the div() function has a "try-catch-finally" construct. The **try** block without exception returns result of division. In case of exception, the **catch** block returns an error message. However, in either case, the statement in the **finally** block is executed first.

</> Open Compiler

```
<?php
   function div($x, $y) {
      try {
         if ($y==0)
         throw new Exception("Division by 0");
         else
```

```
            $res=$x/$y;;
            return $res;
         }
         catch (Exception $e) {
            return $e->getMessage();
         }
         finally {
            echo "This block is always executed\n";
         }
      }
      $x=10;
      $y=0;
      echo div($x,$y);
   ?>
```

It will produce the following **output** –

```
This block is always executed
Division by 0
```