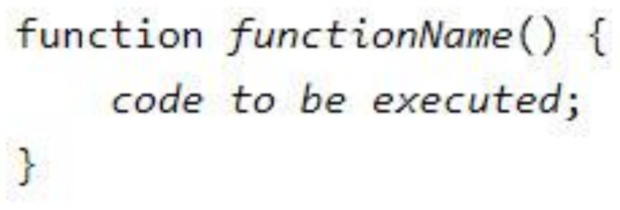
Function & Exception handling in PHP

Objectives

- Creating and calling functions in PHP
- Exception handling in PHP

PHP Functions

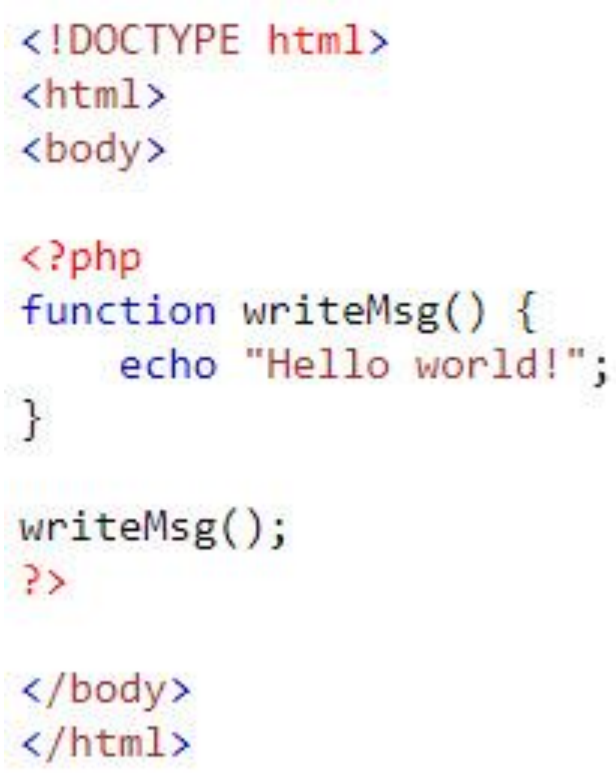
- Besides the built-in PHP functions, we can create our own functions.
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute immediately when a page loads.
- A function will be executed by a call to the function.
- A user-defined function declaration starts with the word function
- Syntax



```
function functionName() {
    code to be executed;
}
```

PHP Functions

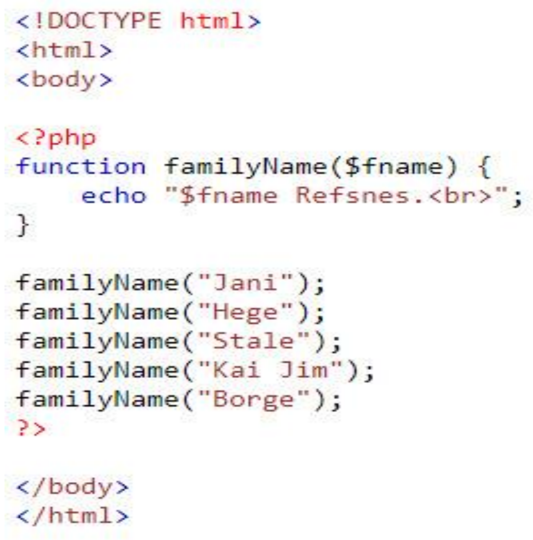
- **Note:** A function name can start with a letter or underscore (not a number).
- Example



```
<!DOCTYPE html>
<html>
<body>

<?php
function writeMsg() {
    echo "Hello world!";
}

writeMsg();
?>

</body>
</html>
```

Function Arguments

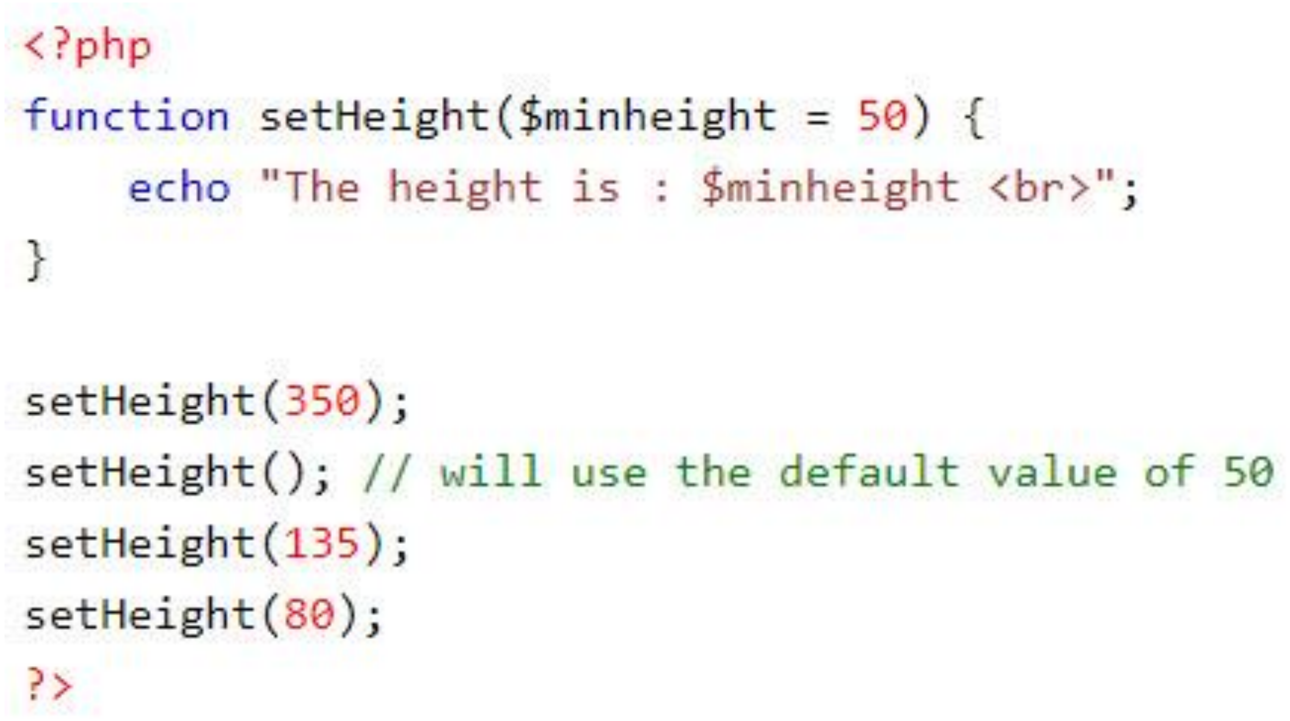
- Information can be passed to functions through arguments. An argument is just like a variable.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
- Example



```
<!DOCTYPE html>
<html>
<body>

<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>

</body>
</html>
```

Default Argument Value

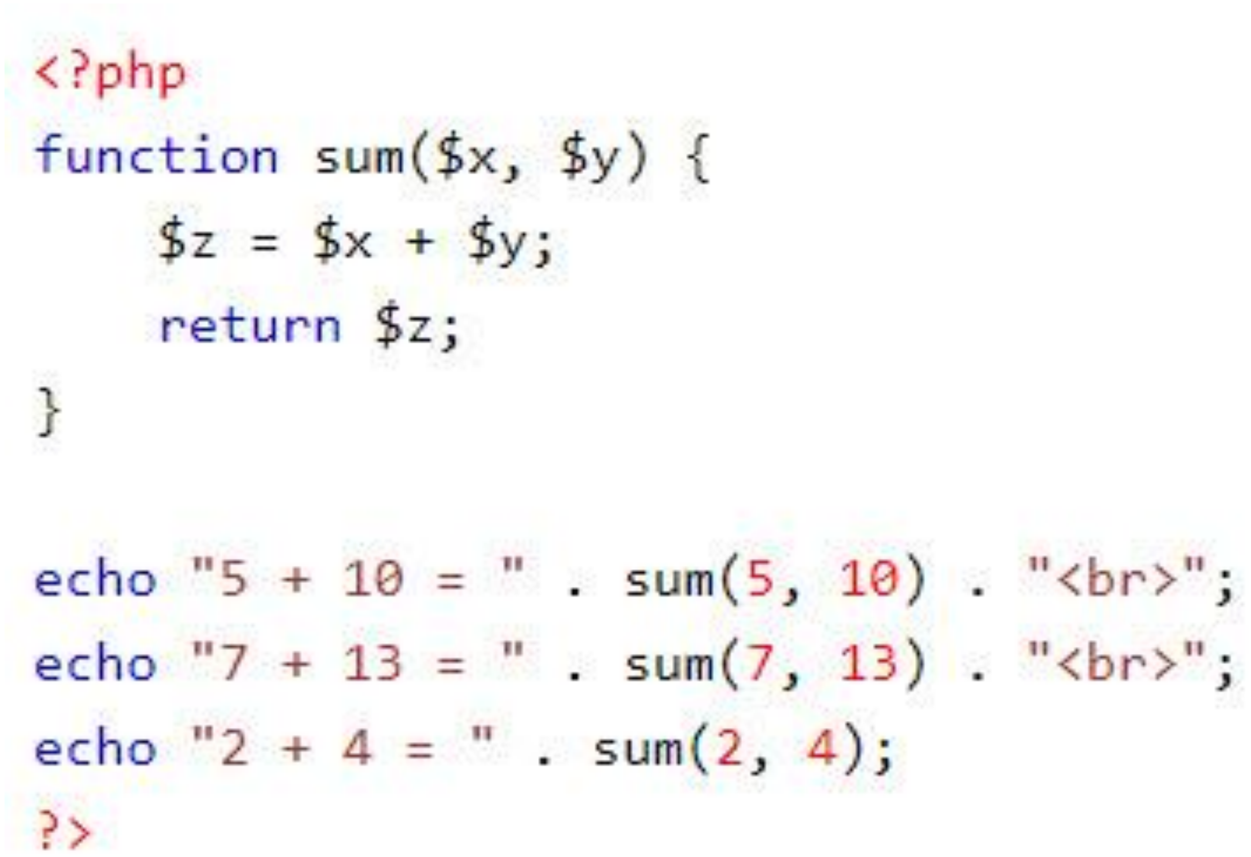
- Example



```php
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

Returning values

- To let a function return a value, use the return statement



```php
<?php
function sum($x, $y) {
    $z = $x + $y;
    return $z;
}

echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

Demo

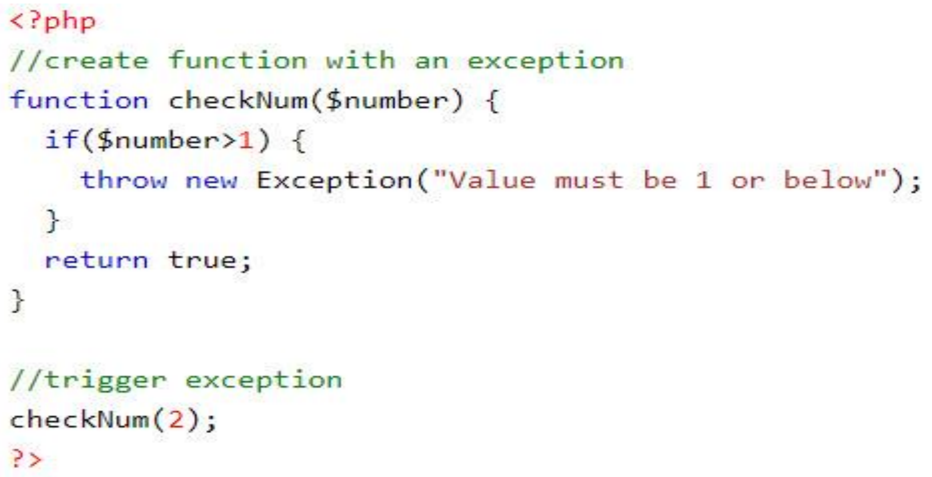
- Teacher demo about creating functions for students

Exception Handling

- Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.
- This is what normally happens when an exception is triggered:
  - The current code state is saved
  - The code execution will switch to a predefined (custom) exception handler function
  - Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code
- **Note:** Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point.

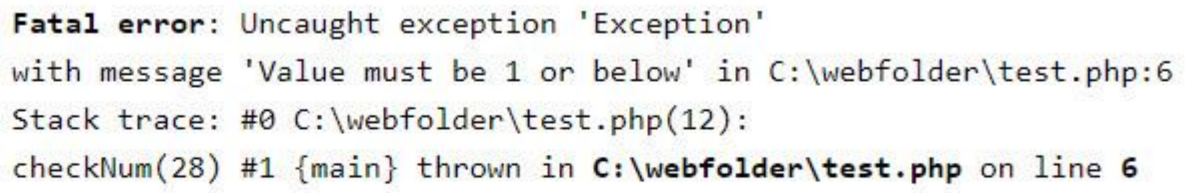
Use of Exceptions

- When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.
- If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.
- Example:



```php
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}

//trigger exception
checkNum(2);
?>
```

- Output:



```
Fatal error: Uncaught exception 'Exception'
with message 'Value must be 1 or below' in C:\webfolder\test.php:6
Stack trace: #0 C:\webfolder\test.php(12):
checkNum(28) #1 {main} thrown in C:\webfolder\test.php on line 6
```

Exception Handling

- Try, throw and catch
  - try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"
  - throw - This is how you trigger an exception. Each "throw" must have at least one "catch"
  - catch - A "catch" block retrieves an exception and creates an object containing the exception information

Exception Handling

- Example

```php
<?php
//create function with an exception
function checkNum($number) {
  if($number>1) {
    throw new Exception("Value must be 1 or below");
  }
  return true;
}

//trigger exception in a "try" block
try {
  checkNum(2);
  //If the exception is thrown, this text will not be shown
  echo 'If you see this, the number is 1 or below';
}

//catch exception
catch(Exception $e) {
  echo 'Message: ' .$e->getMessage();
}
?>
```

- Output:

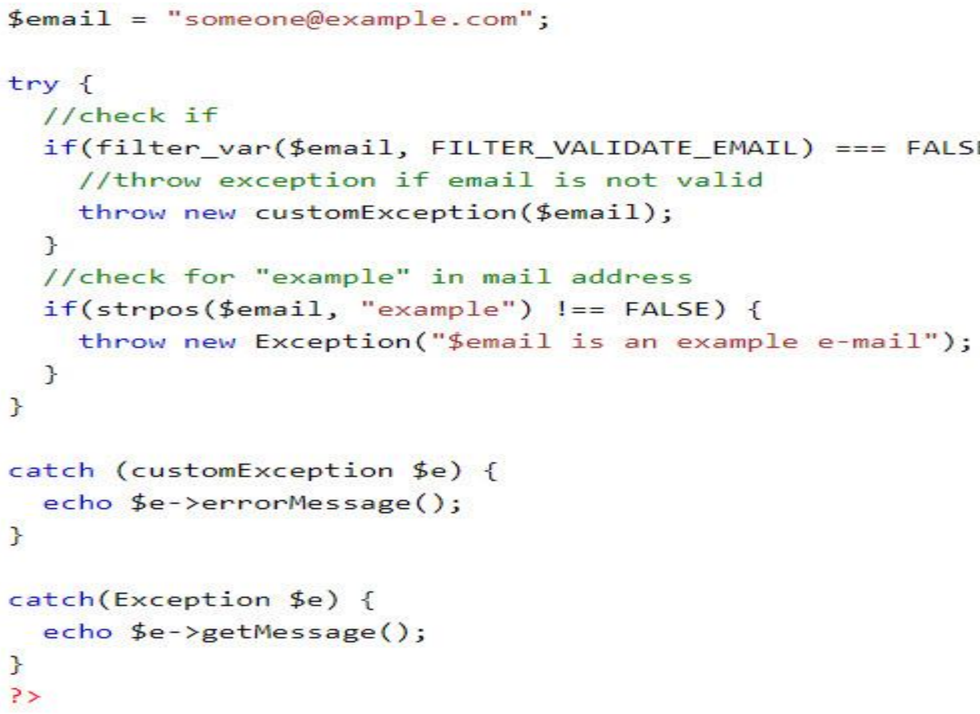
```
Message: Value must be 1 or below
```

Exception Handling

- Example explained:
  - The code above throws an exception and catches it:
  - The checkNum() function is created. It checks if a number is greater than 1. If it is, an exception is thrown
  - The checkNum() function is called in a "try" block
  - The exception within the checkNum() function is thrown
  - The "catch" block retrieves the exception and creates an object ($e) containing the exception information
  - The error message from the exception is echoed by calling $e->getMessage() from the exception object

Multiple Exceptions

- It is possible for a script to use multiple exceptions to check for multiple conditions.
- It is possible to use several if..else blocks, a switch, or nest multiple exceptions. These exceptions can use different exception classes and return different error messages
- Example:



```
$email = "someone@example.com";

try {
  //check if
  if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE) {
    //throw exception if email is not valid
    throw new customException($email);
  }
  //check for "example" in mail address
  if(strpos($email, "example") !== FALSE) {
    throw new Exception("$email is an example e-mail");
  }
}

catch (customException $e) {
  echo $e->errorMessage();
}

catch(Exception $e) {
  echo $e->getMessage();
}
?>
```

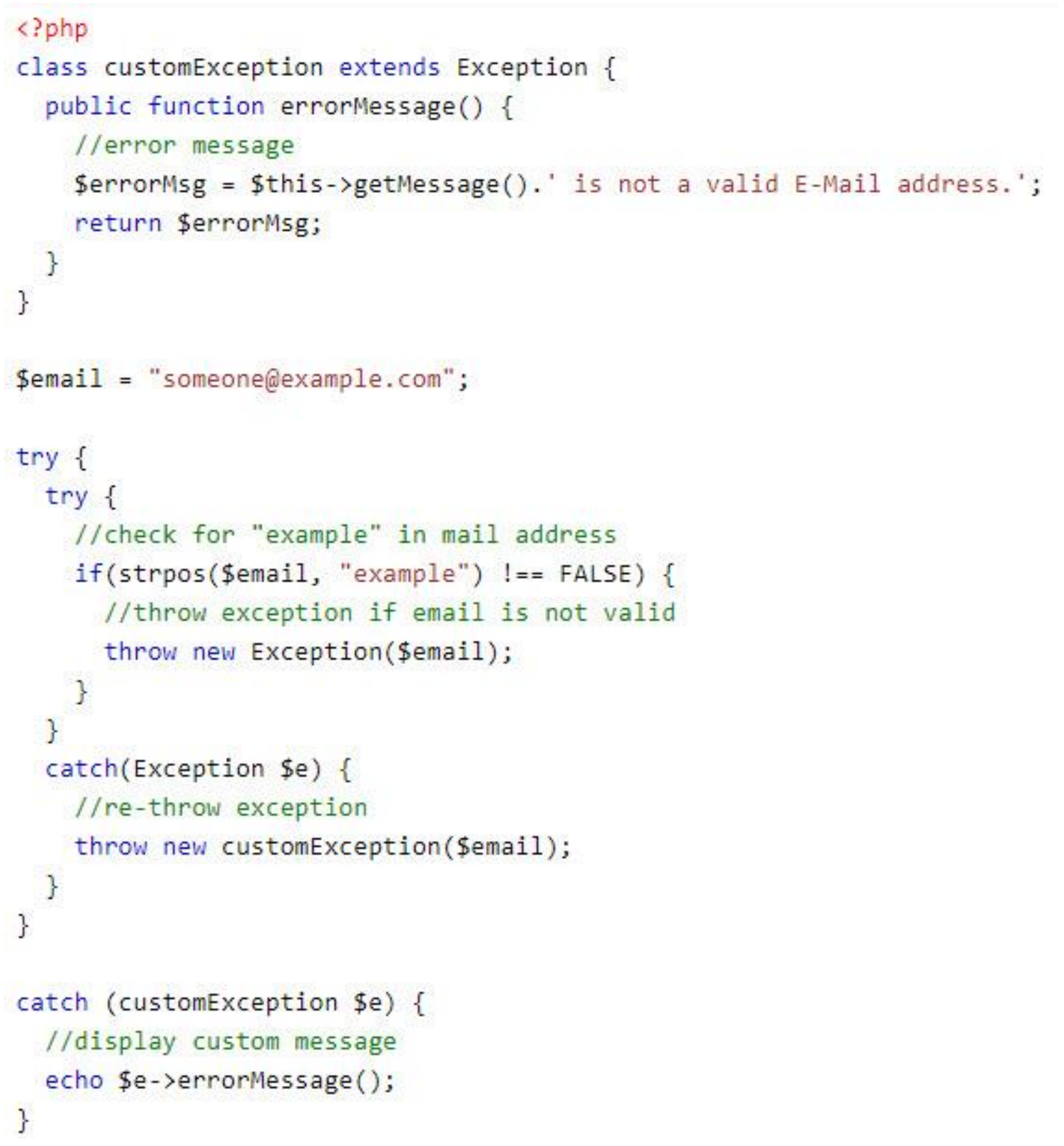
Exception Handling

- Rules for exceptions
  - Code may be surrounded in a try block, to help catch potential exceptions
  - Each try block or "throw" must have at least one corresponding catch block
  - Multiple catch blocks can be used to catch different classes of exceptions
  - Exceptions can be thrown (or re-thrown) in a catch block within a try block

Re-throwing Exceptions

- Sometimes, when an exception is thrown, you may wish to handle it differently than the standard way. It is possible to throw an exception a second time within a "catch" block.
- A script should hide system errors from users. System errors may be important for the coder, but are of no interest to the user.
- To make things easier for the user you can re-throw the exception with a user friendly message

Re-throwing Exceptions



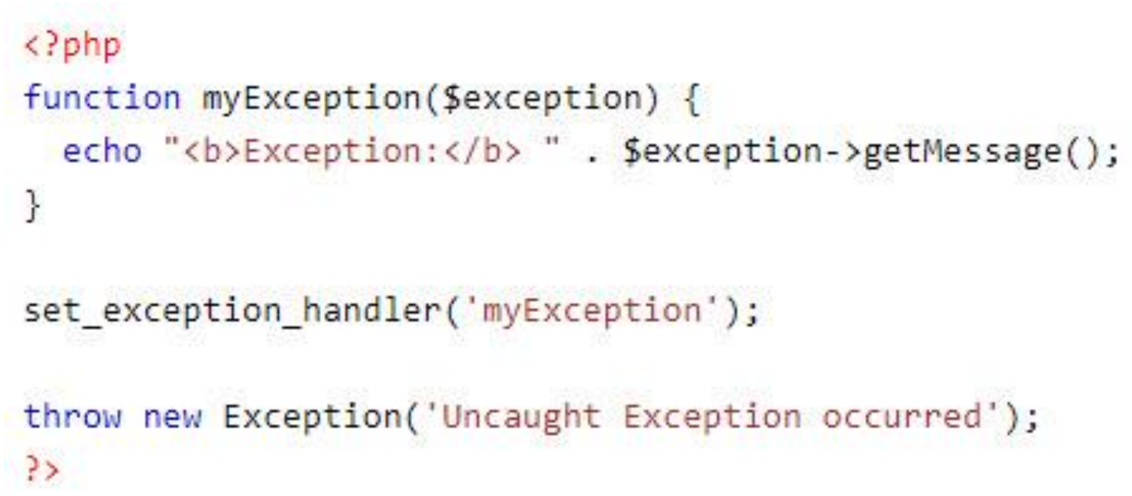
```php
<?php
class customException extends Exception {
  public function errorMessage() {
    //error message
    $errorMsg = $this->getMessage().' is not a valid E-Mail address.';
    return $errorMsg;
  }
}

$email = "someone@example.com";

try {
  try {
    //check for "example" in mail address
    if(strpos($email, "example") !== FALSE) {
      //throw exception if email is not valid
      throw new Exception($email);
    }
  }
  catch(Exception $e) {
    //re-throw exception
    throw new customException($email);
  }
}

catch (customException $e) {
  //display custom message
  echo $e->errorMessage();
}
```

Re-throwing Exceptions

- Example explained:
  - The code above tests if the email-address contains the string "example" in it, if it does, the exception is re-thrown:
    - The customException() class is created as an extension of the old exception class. This way it inherits all methods and properties from the old exception class
    - The errorMessage() function is created. This function returns an error message if an e-mail address is invalid
    - The $email variable is set to a string that is a valid e-mail address, but contains the string "example"
    - The "try" block contains another "try" block to make it possible to re-throw the exception
    - The exception is triggered since the e-mail contains the string "example"
    - The "catch" block catches the exception and re-throws a "customException"
    - The "customException" is caught and displays an error message
- If the exception is not caught in its current "try" block, it will search for a catch block on "higher levels".

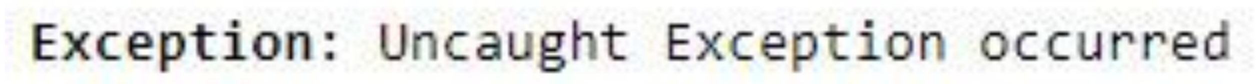
Set a Top Level Exception Handler

- The set_exception_handler() function sets a user-defined function to handle all uncaught exceptions



```php
<?php
function myException($exception) {
  echo "<b>Exception:</b> " . $exception->getMessage();
}

set_exception_handler('myException');

throw new Exception('Uncaught Exception occurred');
?>
```

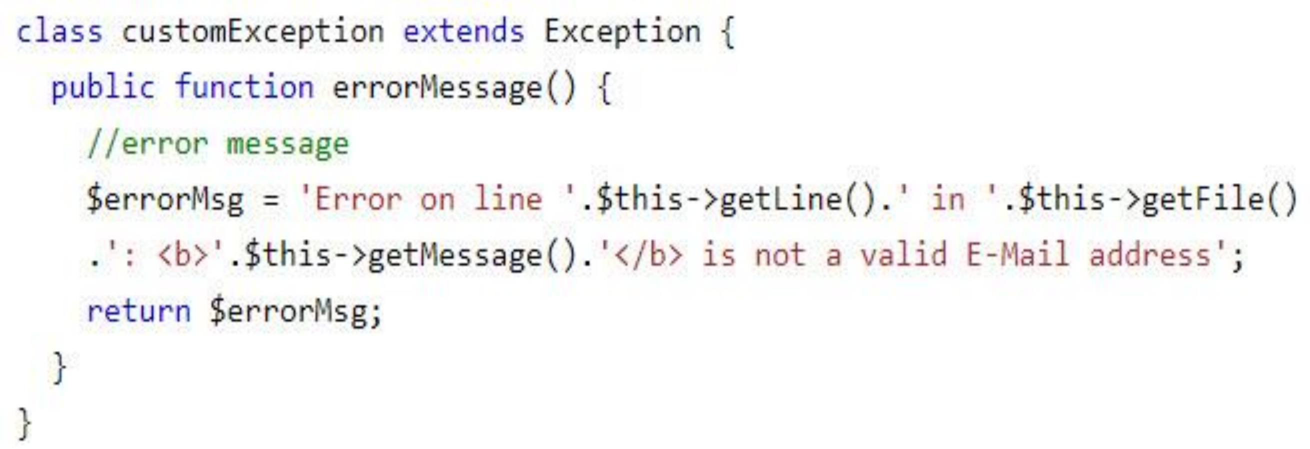
- Output:



```
Exception: Uncaught Exception occurred
```

Custom Exception Class

- To create a custom exception handler you must create a special class with functions that can be called when an exception occurs in PHP. The class must be an extension of the exception class.
- The custom exception class inherits the properties from PHP's exception class and you can add custom functions to it.
- Example



```
class customException extends Exception {
  public function errorMessage() {
    //error message
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';
    return $errorMsg;
  }
}
```

Demo

- Teacher demo code about exception handling for students.

Summary

- Creating and calling functions in PHP
- Exception handling in PHP