Web Development Using PHP

Unit-3

Semester-IV

The INSB IITMS BCA & PGDCA College, Idar

PHP OOPS CONCEPTS ANDADVANCE OOP

* **Introduction to Object Oriented Programming**
* Object-oriented programming (OOP) is a programming paradigm that revolves around the concept of objects. Objects are defined as collections of data and methods that act on that data, and they are used to create and maintain complex systems.
* OOP allows developers to create objects that can interact with each other and share data, making it easier to develop, maintain, and debug applications. OOP languages also provide features such as encapsulation, polymorphism, and inheritance, which allow developers to write code that is more organized, efficient, and secure.

*What is Class?

- Class: A class is a blueprint for creating objects. It defines the properties and methods that an object of that class will have. Classes can be thought of as data types that describe the structure of objects.
- Class is collection of data members and members functions.
- Class is prototype defined by user in which we define the properties & behavior of object.
- Class is group of similar objects.
- Class is logical entity.
- Class is declared using class keyword :- Class student{}
- Class is doesn’t allocated memory when it is created.

Defining a PHP Class:

Syntax: The syntax for defining a PHP class is as follows:

```
class ClassName {
// properties
// methods
}
```

Example: Consider the following example of a class for a car:

```
<?php
class Car
{          // member variables public $color; public $brand; public $model;
           // member functions
public function startEngine()
{
       // code to start the engine
}

public function accelerate()
{
        // code to accelerate the car
}
}
?>
```

*What is Object?

- Objects: An object is an instance of a class. It is created by using the "new" keyword and provides access to the properties and methods defined in the class.
- Object is an real world entity such as pen, table, laptop etc.
- Object is a physical entity.
- Object is created many time as per requirement.
- Object allocated memory when its created.
- Object is created through new keyword -> Student object= new student().

*Constructor

- Constructor functions are special type of functions which are called automatically whenever an object is created. So we take full advantages of this behavior, by initializing many things thought constructor functions.
- PHP provides a special function called _construct() to define a constructor. You can pass as many as arguments you like into the constructor function.
- Notice that the construct function starts with two underscores (__)!
- For ex. Create one constructor for blocks class and it will initialize name and color for the fruit at the time of object creation.

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function get_name() {
     $this->name=$name;
  }
  function get_color() {
    $this->color=$color;
  }
}

$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>
```

Destructor

- A destructor is called when the object is destructed or the script is stopped or exited.
- If you create a __destruct() function, PHP will automatically call this function at the end of the script.
- Notice that the destruct function starts with two underscores (__)!
- The example below has a __construct() function that is automatically called when you create an object from a class, and a __destruct() function that is automatically called at the end of the script

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function __destruct() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

$apple = new Fruit("Apple", "red");
?>
```

Access Modifiers

- Properties and methods can have access modifiers which control where they can be accessed.
- There are three access modifiers:
- public - the property or method can be accessed from everywhere. This is default
- protected - the property or method can be accessed within the class and by classes derived from that class
- private - the property or method can ONLY be accessed within the class
- In the following example we have added three different access modifiers to three properties (name, color, and weight). Here, if you try to set the name property it will work fine (because the name property is public, and can be accessed from everywhere). However, if you try to set the color or weight property it will result in a fatal error (because the color and weight property are protected and private.

* Inheritance

- Inheritance in OOP = When a class derives from another class.
- The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.
- An inherited class is defined by using the extends keyword.
- **Single Inheritance**:- PHP supports single inheritance, meaning a class can inherit from only one parent class at a time.
- **Multiple Inheritance:-**PHP does not support multiple inheritance where a class can inherit from multiple parent classes simultaneously.

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

// Strawberry is inherited from Fruit
class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

* The final keyword

- The final keyword can be used to prevent class inheritance or to prevent method overriding.

-
  ```
  <?php
      class myclass {
       Final  function companyname{
              echo “bca sem-4”;    }
      }
      class intro extends myclass {
            function companyname{
              echo “hello! bca sem-4”;    }
      }
      $bca=new intro();
      $bca->companyname();
      ?>
  ```

* Constants Class

- The **const** keyword is used to declare a class constant. A constant is unchangeable once it is declared. The constant class declared inside the class definition.
- The class constant is case-sensitive. However, it is recommended to name the constants in all uppercase letters.
- Constant differ from normal variables as no $ (dollar sign) is used. The default visibility of class constants is public. Class constants are useful when you need to declare some constant data (which does not change) within a class.
- **There are two ways to access class constant:**

1. **Outside the Class:** The class constant is accessed by using the class name followed by the scope **resolution operator** (::) followed by the constant name.

```
<?php
class code{
    const bca = "Welcome to BCA college idar";    }
echo code::bca              ?>
```

2.**Inside the Class:** It can be accessed by using the *self* keyword followed by the scope resolution operator(::) followed by the constant name.

```
<?php
class code{
    const bca = “Hello! BCA sem-4";
    public function welcome(){
        echo self::bca;  }        }
      $val = new code();
      $val->welcome();
?>
```

Abstract Classes & methods

- Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.
- An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.
- An abstract class or method is defined with the **abstract** keyword.
- Abstract classes are the classes in which at least one method is abstract. Unlike C++ abstract classes in PHP are declared with the help of abstract keyword. Use of abstract classes are that all base classes implementing this class should give implementation of abstract methods declared in parent class. An abstract class can contain abstract as well as non abstract methods.

```
<?php
abstract class Base
{
    abstract function printdata();
}
class Derived extends base {
    function printdata() {
        echo “Hello! BCA sem-4";
    }
}
$b1 = new Derived;
$b1->printdata();
?>
```

Interface

- An Interface allows the users to create programs, specifying the public methods that a class must implement, without involving the complexities and details of how the particular methods are implemented.
- It is generally referred to as the next level of abstraction. It resembles the abstract methods, resembling the abstract classes. An Interface is defined just like a class is defined but with the class keyword replaced by the **interface keyword** and just the function prototypes.
- The interface contains no data variables. The interface is helpful in a way that it ensures to maintain a sort of metadata for all the methods a programmer wishes to work on.
- Interfaces allow you to specify what methods a class should implement.

- Interfaces make it easy to use a variety of different classes in the same way. When one or more classes use the same interface, it is referred to as "polymorphism".
- Interfaces are declared with the interface keyword.
- To implement an interface, a class must use the implements keyword.
- A class that implements an interface must implement **all** of the interface's methods.
-
```
<?php
interface bca{ public function show();              }
class info implements bca {
  public function show() {
    echo “Hello BCA sem-4";      }    }
$bca = new info();
$bca->show();
?>
```

* Static Method & Static Properties

- Static methods can be called directly - without creating an instance of the class first.
- Static methods are declared with the static keyword.
- A class can have both static and non-static properties. A static property can be accessed from a method in the same class using the self keyword and double colon (::):
-

```
<?php
class info {
 public static $name="Hello Bca Sem-4";
 public static function bca() {
 echo "Hello World!";
   }
}
 // Call static method      echo info::$name;
 info::bca();
?>
```

* PHP File Handling

- File handling in PHP is used to you to create, open, read, write, delete, and manipulate files on a server. It is used when you need to store data persistently or handle files uploaded by users.
- File handling is the process of interacting with files on the server, such as reading file, writing to a file, creating new files, or deleting existing ones. File handling is essential for applications that require the storage and retrieval of data.

1. **fopen()** - Opens a file
2. **fclose()** - Closes a file
3. **fread()** - Reads data from a file
4. **fwrite()** - Writes data to a file
5. **file_exists()** - Checks if a file exists
6. **unlink()** - Deletes a file

❖File Modes in PHP

Files can be opened in any of the following modes.

1) **“w”** - Opens a file for writing only.
2) **“r”** - File is open for reading only.
3) **“a”** - File is open for writing only. File pointer points to end of file. Existing data in file is preserved.
4) **“w+”** - Opens file for reading and writing both. If file not exist then new file is created and if file already exists then contents of file is erased.
5) **“r+”** - File is open for reading and writing both.
6) **“a+”** - File is open for write/read. File pointer points to end of file. Existing data in file is preserved. If file is not there then new file is created.
7) **“x”** - New file is created for write only.

- **ReadFile Function:-** The readfile() function reads a file and writes it to the output buffer.
- The readfile() function is useful if all you want to do is open up a file and read its contents.

```
<?php
      echo readfile("webpage.txt");
 ?>
```

- **Opening a File:-** A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.
- **Closing a file:-** The fclose() function is used to close an open file. The fclose() requires the name of the file (or a variable that holds the filename) we want to close.
- **Check End-of-file:-** The feof() function checks if the "end-of-file" (EOF) has been reached. The feof() function is useful for looping through data of unknown length.

- **Reading a file line by line:-** The fgets() function is used to read a single line from a file.
- **Reading a file character by character:-** The fgetc() function is used to read a single character from a file.
- File system function:- readfile(); , file_exists(); ,copy(); rename(); mkdir(); rmdir(); delete(); unlink(); filesize(); filetype(); realpath(); pathinfo(); dirname(); basename();

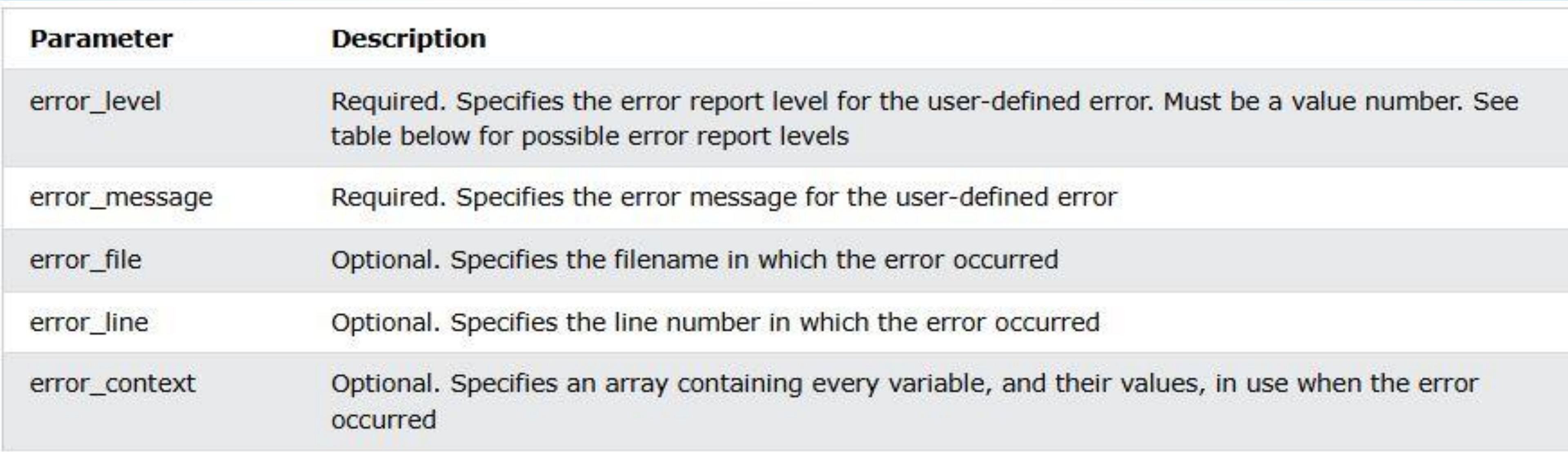
* PHP Error Handling

- Simple "die()" statements:- The die() method is used to throw an exception. **die()** is the same as **exit()**. A program’s result will be an empty screen. Use die() when there is an error and have to stop the execution.
- The die() function print a message and exit from current script
- Ex.

```
if(!file_exists("bca41.txt"))
        {     die("File is not present");     }
        else{
        $file = fopen("bca41.txt", "w");
       }
```

* PHP Custom Error

- Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP.
- This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context).
- Syntax:- error_function(error_level,error_message, error_file,error_line,error_context)



Parameter	Description
error_level	Required. Specifies the error report level for the user-defined error. Must be a value number. See table below for possible error report levels
error_message	Required. Specifies the error message for the user-defined error
error_file	Optional. Specifies the filename in which the error occurred
error_line	Optional. Specifies the line number in which the error occurred
error_context	Optional. Specifies an array containing every variable, and their values, in use when the error occurred

PHP Error Triggers

- In a script where users can input data it is useful to trigger errors when an illegal input occurs. In PHP, this is done by the trigger_error() function.
- For ex.

```
<?php
    $test=2;
    if ($test>=1) {
     trigger_error("Value must be 1 or below");
    }
  ?>
```

```
<?php
//error handler function
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
}

//set error handler
set_error_handler("customError",E_USER_WARNING);

//trigger error
$test=2;
if ($test>=1) {
  trigger_error("Value must be 1 or
below",E_USER_WARNING);
}
?>
```

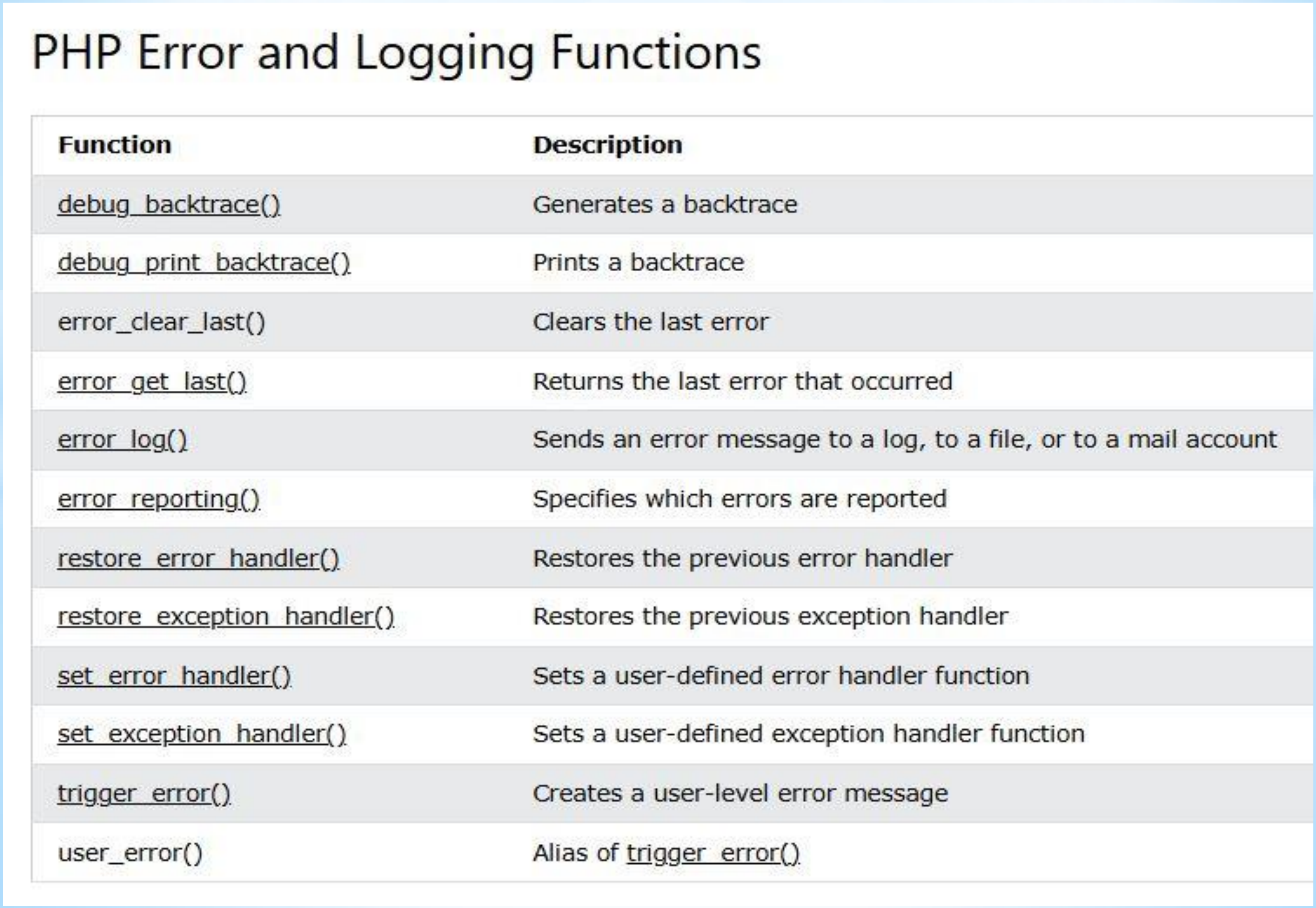
* Error Reporting

- The error_reporting() function specifies which errors are reported.
- To adjust error reporting in PHP, you can use the error_reporting() function to set the level of errors that are reported.
- You can also use the ini_set() function to adjust the display_errors and log_errors settings to control whether errors are displayed on the screen or logged to a file.
- **Error reporting and Logging**:- The error_log() function sends an error message to a log, to a file, or to a mail account.
- Syntax:- error_log(*message*, *type*, *destination*, *headers*);
- Message:- Required. Specifies the error message to log

* Error Reporting

- Type:- Optional. Specifies where the error message should go. 0 - Default. Message is sent to PHP's system logger, 1 - Message is sent by email to the address in the *destination* parameter. 3 - Message is appended to the file specified in *destination*
- *Destination:-* Optional. Specifies the destination of the error message. This value depends on the value of the *type* parameter.
- *Headers:-* Only used if the *type* parameter is set to 1. Specifies additional headers, like From, Cc, and Bcc. Multiple headers should be separated with a CRLF (\r\n).

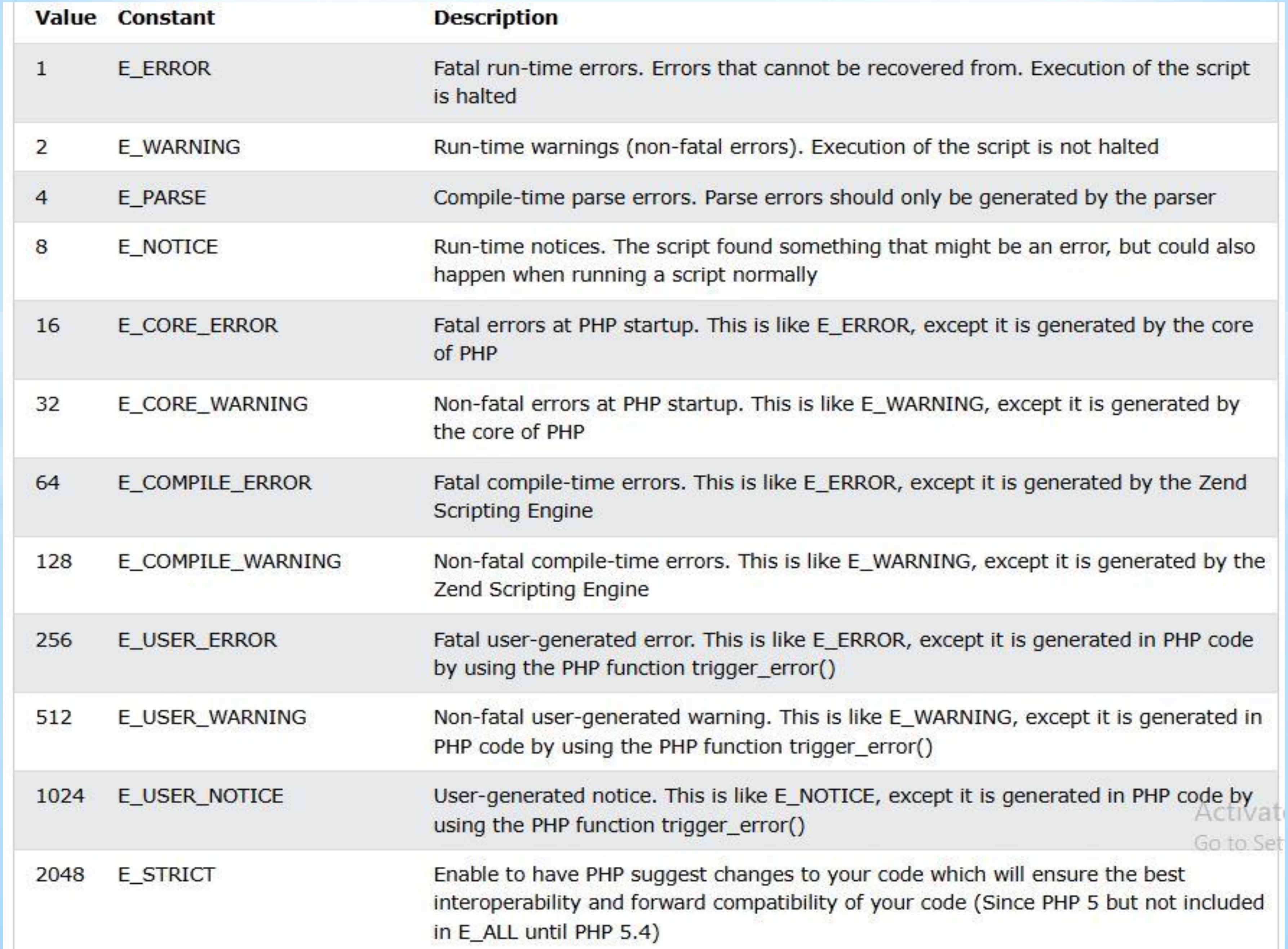
❖Error-Reporting Functions (Debugging Techniques):-



PHP Error and Logging Functions

Function	Description
debug_backtrace()	Generates a backtrace
debug_print_backtrace()	Prints a backtrace
error_clear_last()	Clears the last error
error_get_last()	Returns the last error that occurred
error_log()	Sends an error message to a log, to a file, or to a mail account
error_reporting()	Specifies which errors are reported
restore_error_handler()	Restores the previous error handler
restore_exception_handler()	Restores the previous exception handler
set_error_handler()	Sets a user-defined error handler function
set_exception_handler()	Sets a user-defined exception handler function
trigger_error()	Creates a user-level error message
user_error()	Alias of trigger_error()

❖Error-Reporting Functions (Debugging Techniques):-



Value	Constant	Description
1	E_ERROR	Fatal run-time errors. Errors that cannot be recovered from. Execution of the script is halted
2	E_WARNING	Run-time warnings (non-fatal errors). Execution of the script is not halted
4	E_PARSE	Compile-time parse errors. Parse errors should only be generated by the parser
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally
16	E_CORE_ERROR	Fatal errors at PHP startup. This is like E_ERROR, except it is generated by the core of PHP
32	E_CORE_WARNING	Non-fatal errors at PHP startup. This is like E_WARNING, except it is generated by the core of PHP
64	E_COMPILE_ERROR	Fatal compile-time errors. This is like E_ERROR, except it is generated by the Zend Scripting Engine
128	E_COMPILE_WARNING	Non-fatal compile-time errors. This is like E_WARNING, except it is generated by the Zend Scripting Engine
256	E_USER_ERROR	Fatal user-generated error. This is like E_ERROR, except it is generated in PHP code by using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like E_WARNING, except it is generated in PHP code by using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like E_NOTICE, except it is generated in PHP code by using the PHP function trigger_error()
2048	E_STRICT	Enable to have PHP suggest changes to your code which will ensure the best interoperability and forward compatibility of your code (Since PHP 5 but not included in E_ALL until PHP 5.4)

* PHP Exceptions

- An exception is an object that describes an error or unexpected behavior of a PHP script.
- Exceptions are thrown by many PHP functions and classes.
- User defined functions and classes can also throw exceptions.
- Exceptions are a good way to stop a function when it comes across data that it cannot use.

* Throwing an Exception

- The throw statement allows a user defined function or method to throw an exception. When an exception is thrown, the code following it will not be executed.
- If an exception is not caught, a fatal error will occur with an "Uncaught Exception" message.

❖ The try...catch..finally Statement

- To avoid the error from the example following, we can use the try...catch statement to catch exceptions and continue the process.
- 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
- The **finally** block is optional and is used to execute cleanup code that should always run, regardless of whether an exception is thrown or caught.
- Syntax:- try {

```
// Code that may throw exceptions
} catch (Exception $e) {

// Code to handle exceptions
} finally {

// Optional: Code that always executes
}
```

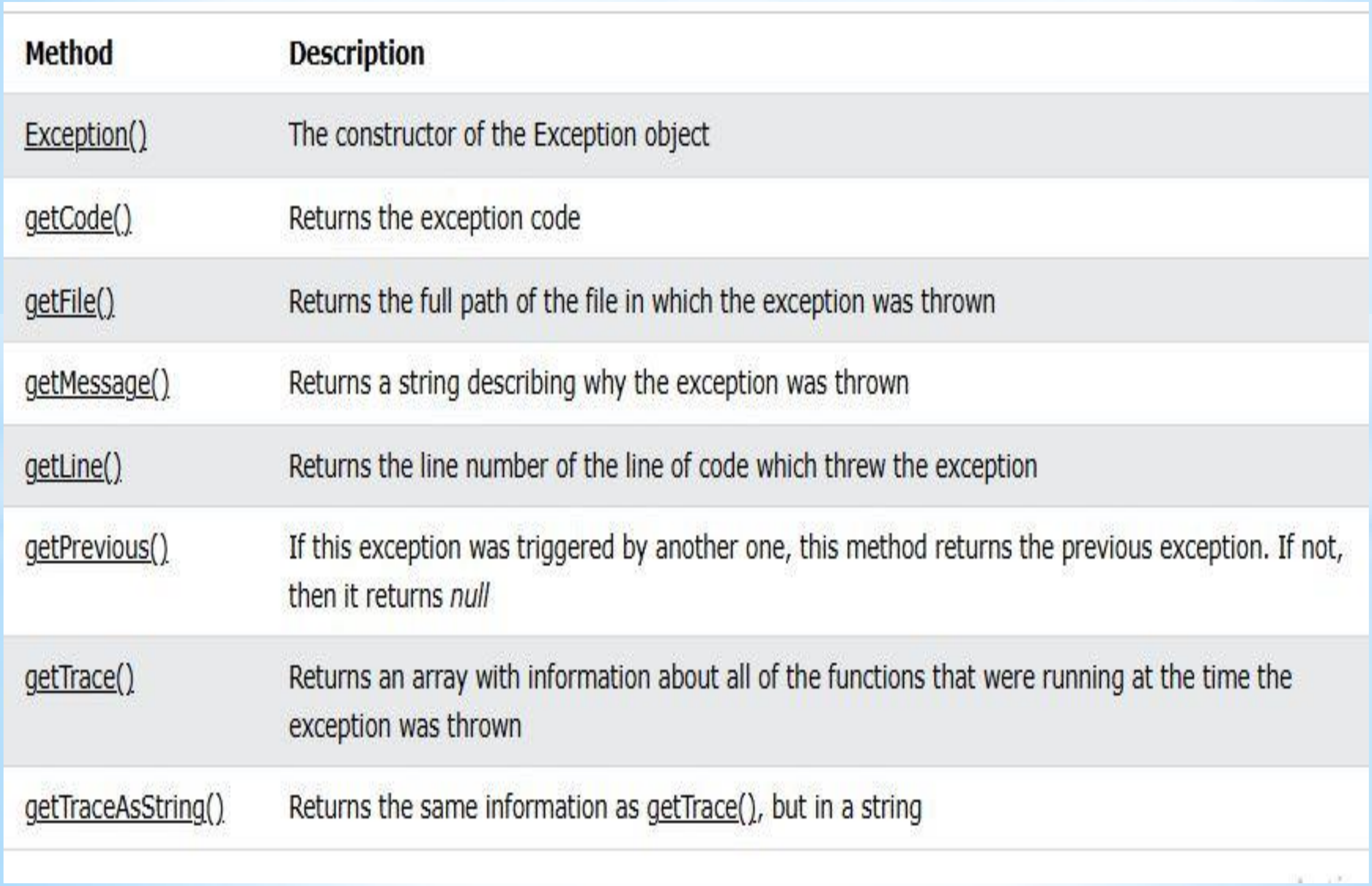
For Ex:-

```
      try {
           $result = 1 / 0;
           }
           catch (DivisionByZeroError $e)
           {
             echo "Error: ", $e -> getMessage(), "\n";
           }
          finally
           {
            echo "Executing finally block";
          }
```

❖ <u>PHP Exception Object</u>

- The Exception Object contains information about the error or unexpected behavior that the function encountered.
- Syntax:- new Exception(message, code, previous)
- Exceptions are used by functions and methods to send information about errors and unexpected behaviour.
- The Exception object has no public properties, but it has private and protected properties which can be written to or read from using the constructor and methods.

PHP Exception Object



Method	Description
Exception()	The constructor of the Exception object
getCode()	Returns the exception code
getFile()	Returns the full path of the file in which the exception was thrown
getMessage()	Returns a string describing why the exception was thrown
getLine()	Returns the line number of the line of code which threw the exception
getPrevious()	If this exception was triggered by another one, this method returns the previous exception. If not, then it returns *null*
getTrace()	Returns an array with information about all of the functions that were running at the time the exception was thrown
getTraceAsString()	Returns the same information as getTrace(), but in a string

```
For Ex:- <?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero", 1);
  }
  return $dividend / $divisor;
}
try {
  echo divide(5, 0);
} catch(Exception $ex) {
  $code = $ex->getCode();
  $message = $ex->getMessage();
  $file = $ex->getFile();
  $line = $ex->getLine();
  echo "Exception thrown in $file on line $line: [Code $code]
  $message";
}
?>
```