

# Web Design and Programming

Week 8

13 June 2024

Instructor: Dr. Peeraya Sripian

# Course schedule

Week	Date	Topic
1	4/18	Intro to WWW, Intro to HTML
2	4/25	CSS Fundamental
	5/2	Holiday (GW)
3	5/9	CSS and Bootstrap
4	5/16	Work on midterm project
5	COIL 22 MAY 18:00-19:30 (counted as 1 class, replacing 23 May)	
6	5/30	Midterm project presentation week
7	6/6	PHP fundamentals + Installation XAMPP
8	6/13	PHP fundamentals 2 + Intro of Final project
9	6/20	mySQL fundamentals
10	6/27	Assessing MySQL using PHP, MVC pattern
11	7/4	Cookies, sessions, and authentication + <b>Proposal of final project</b>
12	7/11	Javascript and PHP validation
13	7/18	Final project development
14	7/25	<b>Final project presentation</b>

# Today's topic

- String manipulation and Regular Expression
- Introduction to OOP in PHP
  - Classes, objects, and properties
  - Methods and inheritance
- Error handling and exceptions in PHP
  - Handling and logging errors
  - Exception handling and custom exceptions
- PHP frameworks
  - Laravel, Symfony, CodeIgniter
  - Overview of popular frameworks and their features

# String

- In PHP, a string is a data type that represents a sequence of characters. It is used to store and manipulate textual data. Strings can contain letters, numbers, symbols, and special characters.
- In PHP, strings can be defined using single quotes `'..'` or double quotes `"..."` For example:

```
$string1 = 'Hello World';  
$string2 = "I am a string";
```

- Strings can also be assigned to variables, making them dynamic. For example:

```
$name = "John";  
$greeting = "Hello, " . $name;
```

# String manipulation

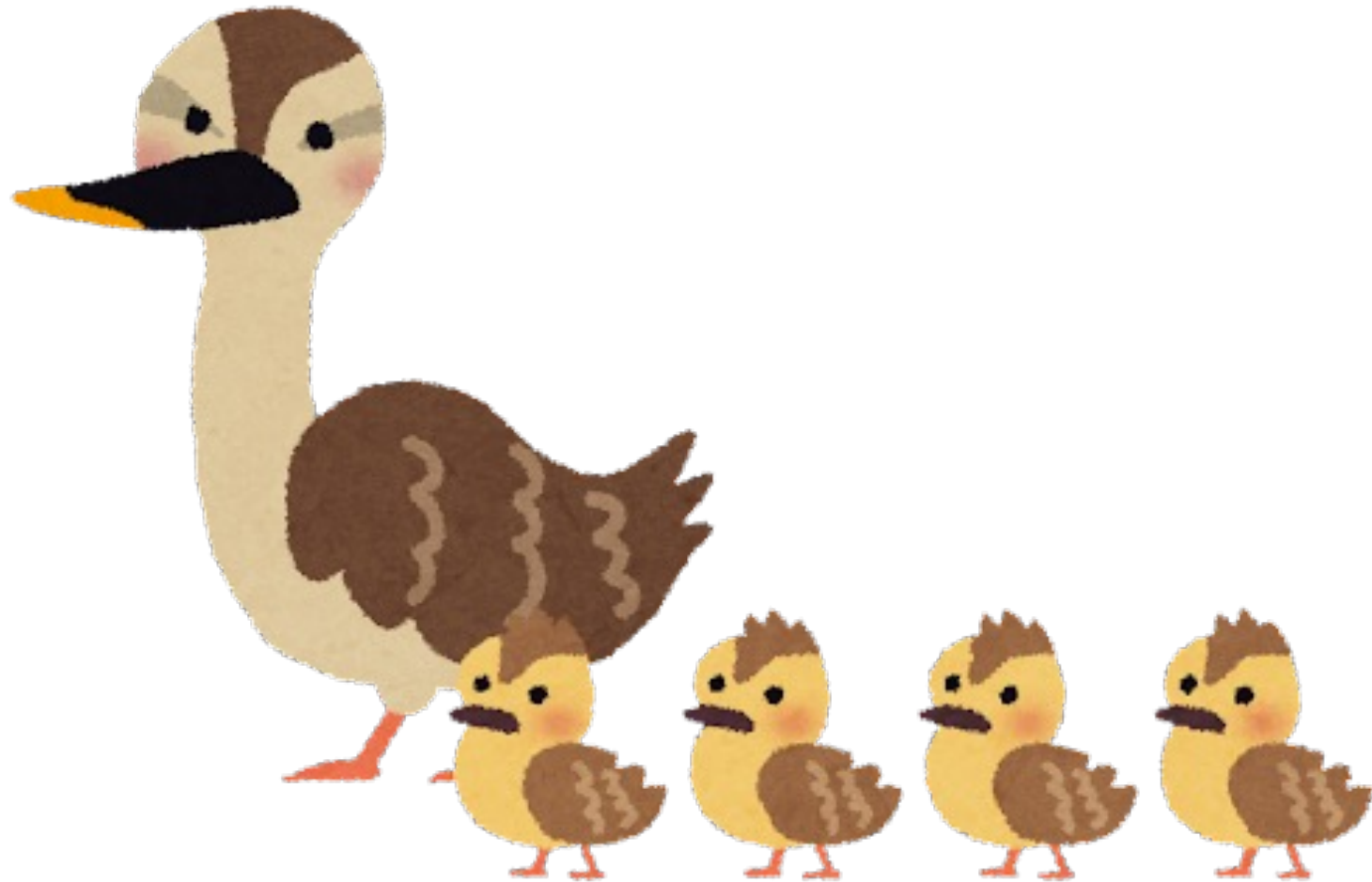
- **strlen()**: returns the length of a string.
- **substr()**: extracts a substring from a string.
- **str\_replace()**: replaces all occurrences of a search string with a replacement string.
- **strtolower()**, **strtoupper()**: convert a string to lowercase or uppercase.
- **trim()**: removes whitespace or specified characters from the beginning and end of a string.
- **explode()**, **implode()**: allow splitting a string into an array and joining an array into a string, respectively.

# Regular expression

- Regular expression (regex) is a sequence of characters that defines a search pattern, used for pattern matching and manipulating text data.
- Widely used in programming languages, including PHP.
- Used for tasks like validation, data extraction, and text manipulation.
- PHP supports regular expressions for more complex pattern matching.

# PCRE

- PCRE (Perl Compatible Regular Expressions) is a library and set of functions for pattern matching in PHP.
- PCRE regular expressions use a syntax and behavior similar to Perl regular expressions.
- PCRE regular expressions are enclosed in forward slashes ("/") in PHP.
- Example: `/pattern/` represents a PCRE regular expression pattern.
- PCRE functions in PHP allow matching, searching, and replacing strings using regular expressions.
- Example: `preg_match($pattern, $subject)` matches a string against a pattern using PCRE.
- PCRE supports a wide range of features, including character classes, quantifiers, anchors, modifiers, and more.
- Example: `/[0-9]+/` matches one or more digits in a string.
- PCRE offers powerful capabilities for pattern matching and manipulation tasks in PHP applications.



# OOP Concept



# Object-Oriented Programming

- A programming paradigm that involves creating objects that combine data and functions. In contrast to procedural programming, which focuses on procedures or functions that manipulate data, OOP offers several advantages:
  - **Efficiency**: OOP is faster and easier to execute
  - **Structure**: OOP provides a clear and organized structure for programs
  - **DRY (Don't Repeat Yourself) Principle**: OOP promotes code reusability by extracting common code into reusable components.
  - **Maintainability**: OOP simplifies code maintenance, modification, and debugging by enabling easier issue identification and isolation through its modular nature.
  - **Reusability**: OOP allows for the creation of reusable applications with less code and faster development. Leveraging objects and classes, developers can utilize existing components, minimizing the need for reinventing solutions.

Remember, adhering to the DRY principle means extracting common code and reusing it rather than repeating it throughout the application.

# Classes and Objects

- **Class:**

- Blueprint or template that defines properties and behaviors
- Represents a specific type of object
- Defines structure and behavior

- **Objects:**

- Created from a class
- Instance of a class
- Holds data and can perform actions
- Represents a concrete, specific thing

- **In summary:**

- Class: Blueprint or template defining properties and behaviors
- Object: Instance of a class, holds data and performs actions

```
<?
// Class definition
class Car
{
    // Properties
    public $brand;
    public $color;

    // Method
    public function startEngine()
    {
        echo "The {$this->brand} car with
color {$this->color} is starting the
engine.";
    }
}

// Object instantiation
$myCar = new Car();

// Object property assignment
$myCar->brand = "Toyota";
$myCar->color = "Red";

// Object method invocation
$myCar->startEngine();
```

# Visibility

- Visibility in PHP OOP refers to the accessibility of properties and methods within a class.
  - **Public** visibility allows properties and methods to be accessed from anywhere, both within the class and externally.
  - **Protected** visibility restricts access to properties and methods to the class itself and its subclasses (child classes).
  - **Private** visibility limits access to properties and methods only within the class where they are defined.
- By controlling visibility, you can enforce encapsulation and prevent direct access to sensitive data or implementation details.
- Visibility modifiers are declared using the keywords `public`, `protected`, or `private` preceding the properties or methods.

# Methods and Properties

- **Methods:**

- Functions defined within a class that perform specific actions or operations.
- Encapsulate behavior and can manipulate the properties of an object.
- Declared within a class using the function keyword.

- **Properties:**

- Variables that hold data within an object.
- Define the state or characteristics of an object.
- Declared within a class and can have different visibility modifiers (**public**, **protected**, or **private**).

```
<?
class Car {
    // Properties
    public $brand;
    private $color;

    // Method to set the color
    public function setColor($newColor) {
        $this->color = $newColor;
    }


    // Method to get the color
    public function getColor() {
        return $this->color;
    }
}

// Object instantiation
$myCar = new Car();

// Setting property value using a method
$myCar->setColor("Red");

// Getting property value using a method
echo "The color of the car is " . $myCar->getColor();
?>
```

# Magic Methods

- Methods which are called automatically when certain actions are undertaken
- Also can be manually called too
- Typically prefixed with two underscores 
- Magic methods provide flexibility and customization within PHP classes, allowing you to define special behaviors and handle various scenarios in a more controlled manner.

# Magic methods

- **\_\_construct**: called when an object is instantiated from a class. It is commonly used for initializing object properties or performing setup tasks.
- **\_\_destruct**: called when an object is no longer referenced or goes out of scope. It is typically used for performing cleanup tasks or releasing resources.
- **\_\_get** and **\_\_set**: called when getting or setting inaccessible or undefined properties of an object, respectively. They allow you to define custom logic for handling property access.
- **\_\_call** and **\_\_callStatic**: invoked when calling inaccessible or undefined methods of an object or a class, respectively. They provide the ability to define custom behavior for method invocation.
- **\_\_toString**: called automatically when an object is treated as a string, such as when using echo or print. It allows you to define how the object should be represented as a string.
- **\_\_isset** and **\_\_unset**: triggered when using **isset()** or **unset()** on inaccessible or undefined properties of an object. They enable you to define custom behavior for checking the existence or unsetting of properties.

# Working with objects

- In PHP, working with objects involves creating instances of classes and interacting with their properties and methods.

## 1. Creating Objects:

- Objects are instances of classes, created using the **new** keyword followed by the class name.
- The process of creating an object is called instantiation.
- Example: **`$object = new ClassName();`**

## 2. Accessing Object Properties:

- Object properties hold data specific to each object.
- Properties are accessed using the object instance followed by the arrow operator (**`->`**) and the property name.
- Example: **`$object->propertyName = "value";`**

# Working with objects

## 3. Calling Object Methods:

- Object methods are functions defined within a class that perform actions or operations.
  - Methods are invoked using the object instance followed by the arrow operator (->) and the method name, optionally with parentheses if the method accepts arguments.
  - Example: `$object->methodName();`
- 
- Objects encapsulate data and behavior, allowing you to create reusable and modular code.



# \$this

- In PHP, the **\$this** keyword is a special variable that refers to the current object instance within a class. It is commonly used to access properties and methods of the object.
- The **\$this** keyword allows you to refer to the current object instance within the class, enabling access to its properties and methods.

```
<?
class Example {
    private $name;

    public function setName($name) {
        $this->name = $name;
    }

    public function sayHello() {
        echo "Hello, my name is " . $this->name;
    }
}

// Create an object of the Example class
$example = new Example();

// Set the name using the setName() method
$example->setName("John");

// Call the sayHello() method
$example->sayHello(); // Output: Hello, my
name is John
?>
```

# Static

- In PHP, properties and methods can indeed be accessed from an uninstantiated class (not an object) if they are declared as static and accessed using the Scope Resolution Operator (::).

```
<?
class Example {
    public static $property = "Hello, I am a
static property.";

    public static function sayHello() {
        echo "Hello, I am a static method.";
    }
}

// Accessing static property
echo Example::$property; // Output: Hello,
I am a static property.

// Calling static method
Example::sayHello(); // Output: Hello, I am
a static method.

?>
```

# Class constants

- In PHP, class constants are values that are associated with a class and remain constant throughout the execution of the program.
- They are similar to class properties, but they cannot be changed once they are defined (unless you modify the PHP file directly).
- Constants provide a way to define values that should remain constant and unchangeable throughout the program's execution.
- They can be used to store configuration values, mathematical constants, error codes, or any other fixed value that is relevant to the class.
- Class constants offer clarity and self-documentation to your code, as they indicate that the value should not be modified.

# Declaring Class Constants

- Class constants are declared using the **const** keyword followed by the constant name and its value.
- Conventionally, constant names are written in uppercase.
- Constants are typically defined at the top of the class, outside of any methods.
- Example:

```
class Example {  
    const MAX_VALUE = 100;  
    const PI = 3.14159;  
}
```

# Accessing Class Constants

- Class constants are accessed using the Scope Resolution Operator (::) followed by the constant name.
- Unlike class properties, constants do not require an object instance to be accessed.
- Example:

```
echo Example::MAX_VALUE; // Output: 100
echo Example::PI;        // Output: 3.14159
```

# Interfaces and implements

- An interface is like a blueprint or a set of rules that a class agrees to follow.
- It defines a list of method declarations (without implementations) that classes implementing the interface must provide.
- Think of an interface as a contract that specifies what methods a class must have, but not how they should be implemented.
- Example: An **Animal** interface may declare methods like **eat()**, **sleep()**, and **makeSound()**.
- The implements keyword is used in a class declaration to specify that the class will adhere to the methods defined in an interface.
- When a class implements an interface, it promises to provide implementations for all the methods declared in the interface.
- By implementing an interface, a class can fulfill multiple contracts and share common behaviors across different classes.
- Example: A **Cat** class may implement the **Animal** interface by providing its own implementations for **eat()**, **sleep()**, and **makeSound()** methods.

# Extends

- when a class extends another, it inherits properties, methods, and constants from the parent class.
- Visibility settings play a crucial role in determining how these inherited elements can be accessed and overridden.
- Public and protected elements can be overridden in the child class, provided the names and method signatures match.
- This allows child classes to customize and provide their own implementations while leveraging the shared functionality inherited from the parent class.

# Extends

- **Inheritance:**

- A class (child class) can inherit properties, methods, and constants from another class (parent class).
- The child class is created using the **extends** keyword followed by the name of the parent class.
- Example: **class ChildClass extends ParentClass { ... }**

- **Inherited Elements:**

- When a class extends another, it inherits all the public and protected properties, methods, and constants from the parent class.
- Private properties and methods are not inherited (accessible only in parent).
- The child class can use the inherited elements as if they were defined within the child class itself.



# Overriding Properties and Methods

- In the child class, properties and methods inherited from the parent class can be overridden if they have the same name.
- Overriding allows the child class to provide its own implementation of the property or method.
- The method signature (name and number of parameters) must match in order to override a method.

# Example

```
<?
class ParentClass {
    protected $property = "Parent Property";

    protected function method() {
        echo "Parent Method";
    }
}

class ChildClass extends ParentClass {
    public $property = "Child Property"; // Overrides parent property

    public function method() { // Overrides parent method
        echo "Child Method";
    }
}

$child = new ChildClass();
echo $child->property; // Output: Child Property
$child->method();      // Output: Child Method
?>
```

- We have a **ParentClass** with a protected property and a protected method.
- The **ChildClass** extends the **ParentClass** and overrides the property and method with its own implementations.
- We create an object of the **ChildClass** and access the overridden property and method, which now reflect the child class implementations.

# Parent

- The **parent** keyword can be used to access and invoke the overridden method from the parent class.
- It allows the child class to selectively utilize and extend the functionality provided by the parent class while providing its own customized implementations.

```
public function callParentMethod() {  
    parent::method(); // Using parent  
    keyword to call the overridden method from  
    parent class  
}
```

```
$child->callParentMethod(); // Output: Parent  
Method
```

- The **ChildClass** now includes a new public method called **callParentMethod()**.
- Inside this method, we use the **parent** keyword followed by the Scope Resolution Operator (::) to explicitly access and call the overridden method() from the parent class.

# Abstract class

- An abstract class in PHP allows us to define both abstract methods that must be implemented by child classes and regular methods and properties that can be extended by child classes.
- Abstract classes cannot be directly instantiated, but they serve as a foundation for creating child classes that provide the required implementations.
- This approach allows for code reusability, enforcing method contracts, and providing extensibility.

# Abstract class

- An abstract class is a class that cannot be instantiated directly. It serves as a blueprint for other classes to extend from.
- It is defined using the abstract keyword before the class declaration.
- Example:

```
abstract class AbstractClass { ... }
```

# Abstract methods

- Abstract methods are declared within an abstract class, but they do not contain an implementation.
- Child classes extending the abstract class must provide the implementations for all the abstract methods defined in the parent abstract class.

```
abstract class AbstractClass
{
    abstract public function abstractMethod();
}
```

- In addition to abstract methods, an abstract class can contain regular methods and properties with implementations.
- These methods and properties can be inherited and extended by child classes.
- Child classes can override these methods or use them as they are, depending on their needs.

# Encapsulation

- Encapsulation in object-oriented programming hides the internal representation of an object from outside access.
- Typically, only the object itself is allowed to directly access and modify its properties.
- This approach gives the programmer greater control over how properties are modified.
- For example, by encapsulating the email address property and using a setter method, we can enforce checks and restrictions on user input, ensuring data integrity.
- This allows us to perform validation or apply specific rules before accepting the user's input as a valid email address.
- Encapsulation helps maintain data integrity and enables us to enforce rules and restrictions on how properties are accessed and modified.

# Polymorphism

- Polymorphism is a fundamental concept in object-oriented programming (OOP).
- It refers to the ability of objects of different classes to be treated as objects of a common parent class.
- In PHP, polymorphism is achieved through method overriding and method overloading.
- Method overriding allows a subclass to provide its own implementation of a method that is already defined in its parent class.
- Method overloading, on the other hand, enables a class to have multiple methods with the same name but different parameters.
- Polymorphism allows us to write code that can work with objects of different classes, as long as they share a common interface or parent class.
- This flexibility allows for code reusability, extensibility, and the ability to write more generic and flexible code.
- Polymorphism helps in achieving loosely coupled code, where objects can be interchanged easily without affecting the overall functionality of the program.
- By utilizing polymorphism, we can write code that is more adaptable and can handle a wider range of scenarios.

DEMO



# Error handling and exceptions in PHP

---



# Error and error handling

- An error refers to something that has been done incorrectly or wrongly, or something that should not have been done.
- Error handling is the process of detecting and managing errors that occur in our program, taking appropriate action to prevent unforeseen consequences.

# Error handling in PHP

- Identifying and capturing errors that arise during the execution of a PHP script.
- Taking necessary measures to handle those errors and prevent them from causing unexpected issues or disruptions.
- This can include displaying informative error messages to the user, logging errors for debugging purposes, or gracefully recovering from errors to ensure the program continues running smoothly.
- Effective error handling is essential for creating robust and reliable PHP applications, as it helps in identifying and resolving issues promptly, improving user experience, and preventing potential problems from escalating.

# Errors in PHP

- PHP errors occur during the execution of a PHP script and can be caused by syntax errors, runtime issues, or logical mistakes.
  - Syntax errors are violations of PHP's syntax rules and can include missing semicolons, incorrect function names, or mismatched brackets.
  - Runtime errors occur during script execution and can be caused by accessing undefined variables, calling undefined functions, or division by zero.
  - Logic errors are design or implementation mistakes that lead to incorrect program behavior.
  - Fatal errors are severe errors that halt the script execution, such as memory exhaustion or calling undefined functions.
  - Warnings and notices are non-fatal errors that indicate potential issues or provide informational messages.
- Proper error handling techniques, such as try-catch blocks, error reporting settings, or logging errors, help identify and resolve issues in PHP applications.

# Die() function

- The **die()** function is equivalent to the **exit()** function, and they both serve the same purpose.

```
$number = 10;

if ($number > 5) {
    echo "Number is greater than 5.";
} else {
    echo "Number is not greater than 5.";
    die(); // Terminate the script
}

// The code below will not be executed
echo "This line will not be displayed.";
```

- In the example, if **\$number > 5**, it displays **"Number is greater than 5."**
- Otherwise, it shows **"Number is not greater than 5."**
- The **die()** function is used to immediately end the script if the condition is false, preventing the execution of subsequent code.
- However, **die()** or **exit()** functions should be used carefully for error handling or exceptional situations, as they may hinder proper cleanup or graceful handling.

# Defining your own custom error handling

- `set_error_handler()`
  - Can specify a callback function that will be invoked whenever an error occurs.
1. **Define a custom error handling function:**
    - Create a function that will handle the errors according to your requirements.
    - The custom error handling function should have a specific format that accepts the error details as parameters.
  2. **Register the custom error handling function:**
    - Use the `set_error_handler()` function to register your custom error handling function.
    - This informs PHP to invoke your custom function whenever an error occurs.

# Error parameter

- In PHP 7 and 8, the error parameter refers to an integer value that represents the type of error that has occurred.
- It is used in error handling functions to identify and handle different types of errors.
- The error parameter can take on various predefined error constants or user-defined error codes.

# Error parameter

- **E\_ERROR**: Represents fatal errors that can cause script termination.
  - **E\_WARNING**: Indicates non-fatal warnings that may affect script execution.
  - **E\_NOTICE**: Represents informational errors, often related to uninitialized variables or undefined array keys.
  - **E\_PARSE**: Signifies parse errors that occur during script compilation.
  - **E\_EXCEPTION**: Indicates exceptions that are thrown when an error condition is encountered.
- 
- These are some of the frequently encountered error parameters in PHP websites. Handling these errors appropriately is crucial for maintaining the stability and functionality of the website.



# Displaying PHP errors

## 1. Modify PHP Configuration:

- Locate the php.ini file used by your PHP installation.
- Find the ``error_reporting`` directive and set it to the desired error reporting level. For example, ``error_reporting = E_ALL`` displays all types of errors.
- Set the ``display_errors`` directive to On: ``display_errors = On``. This allows errors to be displayed on the screen.
- Save the php.ini file and restart the web server for the changes to take effect.

# Displaying PHP errors

## 2. Use PHP code:

- Add the following lines at the beginning of your PHP script to enable error reporting and display errors on the screen:

```
error_reporting(E_ALL);  
ini_set('display_errors', 1);
```

### Testing Display Errors

**Warning:** Undefined variable \$var in `/var/www/html/gift/class7/13_displayerror.php` on line 16

**Warning:** foreach() argument must be of type array|object, null given in `/var/www/html/gift/class7/13_displayerror.php` on line 16

**Fatal error:** Uncaught DivisionByZeroError: Division by zero in `/var/www/html/gift/class7/13_displayerror.php:17` Stack trace: #0 {main} thrown in `/var/www/html/gift/class7/13_displayerror.php` on line 17

**Note:** `display_errors` will only works so long as that script runs (it cannot have any parse errors). You need to enable `display_errors` in PHP config file to always see the errors

# Exceptions handling

- Exceptions handling in PHP allows you to handle and manage errors or exceptional situations in a more controlled and structured manner.
- **try**: The try block is used to enclose the code that may potentially throw an exception. It is followed by one or more catch blocks or a finally block.
- **throw**: The throw statement is used to explicitly throw an exception within the try block or within a custom function or method. It allows you to specify and raise custom exceptions or use predefined exception classes.
- **catch**: The catch block is used to catch and handle specific exceptions that are thrown within the try block. It allows you to define code that will be executed when a particular exception occurs. You can have multiple catch blocks to handle different types of exceptions.

# Example

```
<?
try {
    // Code that may throw an exception
    $file = fopen("nonexistent_file.txt", "r");

    if (!$file) {
        throw new Exception("File not found!");
    }

    // Other code to be executed if the file is successfully opened
    // ...

    // Close the file
    fclose($file);
} catch (Exception $e) {
    // Exception handling
    echo "Exception: " . $e->getMessage();
}

?>
```

# Creating custom exception handler

- You can create your own custom exception handler by defining a function and registering it as the default exception handler using the `set_exception_handler()` function.
- This allows you to customize how exceptions are handled in your application.



## PHP frameworks

# PHP frameworks

- PHP frameworks are pre-built software libraries that provide a foundation and structure for developing web applications in PHP.
- They offer a set of tools, components, and conventions to streamline the development process and promote code organization, reusability, and maintainability.
- Using a PHP framework can greatly simplify the development process, provide a solid foundation, and enable you to build web applications efficiently and effectively.

# Benefits of using PHP frameworks

- **Increased productivity:** Frameworks offer built-in features, functionalities, and tools, saving development time.
- **Code organization:** Frameworks encourage a structured approach, separating concerns and improving code maintainability.
- **Reusability:** They provide reusable components and modules, reducing the need for repetitive code.
- **Security:** Frameworks often include security measures and protections against common vulnerabilities.
- **Community and documentation:** Many frameworks have active communities and extensive documentation, making it easier to find support and resources.





# Popular PHP frameworks:

- **Laravel:** A robust and modern framework known for its elegant syntax and rich feature set. <https://laravel.com/>
- **Symfony:** A flexible and scalable framework that emphasizes reusable components and follows industry standards. <https://symfony.com/>
- **CodeIgniter:** A lightweight and beginner-friendly framework with a small footprint, suitable for small to medium-sized projects. <https://codeigniter.com/>
- **Zend Framework:** A powerful framework that offers a high degree of flexibility and follows a component-based architecture.
- **Yii:** A high-performance framework with strong caching support and excellent security features.

# End of the topic

---

Advanced PHP

# Next

1. Final project group decision
2. Final project explanation
3. In class activity (download doc and code from Material folder in Github organization)