University of Tissemsilt
Faculty of Science & Technology
Departement of Math and Computer Science

Application Web Development

Introduction to PHP

2024-04-08

Lecturer

Dr. HAMDANI M

Speciality : Computer Science (ISIL)
Semester: S4

Plan

Resources

Resources for individuals interested in mastering PHP programming:

- https://www.w3schools.com/php/php_syntax.asp
- https://www.php.net/manual/en/

What is PHP

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.n
- PHP is easy to learn and runs efficiently on the server side

PHP Installation

- Visual Studio Code (VS Code)
  - **Using XAMP**:
    https://www.youtube.com/watch?v=3xHrMwy-Y5A
  - **Using PHP Server**
    https://www.youtube.com/watch?v=-VcnwkR6QuQ
- Netbeans :
  https://www.youtube.com/watch?v=HvVRfHORHSg

Recommendation

XAMPP provides a convenient platform for running web servers locally, facilitating the setup and testing of web-based projects like WordPress sites.

PHP Formatter - VSCODE

- Install ***intelephense*** extension,
- Open **settings.json**
- add the value :

```
"[php]": {
    "editor.defaultFormatter":
            "bmewburn.vscode-intelephense-client"
}
```

Opening and Closing Tags

PHP code is enclosed within **<?php** and **?>** tags.

```php
<?php

// PHP code goes here

?>
```

The default file extension for PHP files is "**.php**"

Example

A simple **.php** file with both HTML code and PHP code:

```
<!DOCTYPE html>
<html>
<body>
  <h1>My first PHP page</h1>
```

```
  <?php
    echo "Hello World!";
  ?>
```

```
</body>
</html>
```

Comments

- Single-line comments: //
- Multi-line comments: /* */

```php
<?php
    // Single-line comment

    /*
      Multi-line
      comment
    */
?>
```

Variables

- Start with $ followed by the variable name
- Case-sensitive

```php
<?php
  $name = "Ahmed";
  $age = 12;
?>
```

Echoing Output

Print:

- Can only take one argument at a time
- Has a return value of 1 so it can be used in expressions

Echo:

- Can take multiple arguments (although such usage is rare)
- Generally faster than print
- Has no return value

```php
<?php
   echo "Hello, world!";   // no parentheses required
   print("Hello, world!"); // parentheses required
?>
```

Data Types

Strings, integers, floats, booleans, arrays, objects

```php
<?php
    $string = "Hello, world!";
    $integer = 42;
    $float = 3.14;
    $boolean = true;
    $array = array("apple", "banana", "cherry");
```

To verify the type of any object in PHP, use the **var_dump()**

Concatenation

Concatenate strings using the dot (.) operator

```php
<?php
  $greeting = "Hello";
  $name = "Ahmed";
  echo $greeting . " " . $name;
  // Outputs: Hello Ahmed
?>
```

Double or Single Quotes?

Double Quotes ("):

- Variables within double quotes are parsed and their values are inserted into the string.
- Escape sequences like \n, \t, etc., are interpreted as newline, tab, etc

Single Quotes ('):

- Returns the string like it was written (literally)

```php
<?php
  $name = "Ahmed";
  echo "Hello, $name!"; // Output: Hello, Ahmed!

  $name = "Ahmed";
  echo 'Hello, $name!'; // Output: Hello, $name!
```

String Functions

- **strlen()**: Returns the length of a string
- **str_word_count()**: counts the number of words in a string
- **strpos()**: Finds the position of the first occurrence of a substring
- **substr()**: Returns a part of a string
- **str_replace()**: Replaces all occurrences of a substring with another substring
- **strtolower()** / **strtoupper()**: Converts a string to lowercase / uppercase.
- **trim()**: Removes whitespace from the beginning and end of a string
- **explode()**: Splits a string into an array by a specified delimiter.
- **implode()** / **join()**: Joins array elements with a string.

PHP Numbers

- **Integers**: Whole numbers (e.g., 42).
  - PHP_INT_MAX - The largest integer supported
  - PHP_INT_MIN - The smallest integer supported
  - PHP_INT_SIZE - The size of an integer in bytes
  - is_int($var): Checks for true integer (int data type).
- **Floats**: Decimals (e.g., 3.14) or scientific notation.
  - PHP_FLOAT_MAX - The largest floating number
  - PHP_FLOAT_MIN - The smallest floating number
  - PHP_FLOAT_DIG - The number of decimal digits that can be rounded into a float and back without precision loss
  - PHP_FLOAT_EPSILON - The smallest representable positive number x, so that x + 1.0 != 1.0
- **is_numeric($var)**: Checks for any numeric value (number or numeric string).

Change Data Type

- **(string)** - Converts to data type String
- **(int)** - Converts to data type Integer
- **(float)** - Converts to data type Float
- **(bool)** - Converts to data type Boolean
- **(array)** - Converts to data type Array
- **(object)** - Converts to data type Object
- **(unset)** - Converts to data type NULL

Example - Casting

```php
<?php
  $floatNum = 3.14;
  $stringNum = "42";
  $boolValue = true;
  $castToInt = (int)$floatNum; // Cast float to integer
  $castToIntFromString = (int)$stringNum; //string to integer
  $castToFloat = (float)$stringNum; // Cast string to float
  $castToString = (string)$boolValue; // Cast boolean to string
  echo "Original Float: $floatNum, Cast to Integer:
        $castToInt<br>";
  echo "Original String: $stringNum, Cast to Integer:
        $castToIntFromString<br>";
  echo "Original String: $stringNum, Cast to Float:
        $castToFloat<br>";
  echo "Original Boolean: $boolValue, Cast to String:
        $castToString<br>";
```

PHP Operators

PHP offers a rich set of operators, categorized based on their function:

- Arithmetic operators: Perform mathematical calculations (+, -, *, /, %)
- Assignment operators: Assign values to variables (=, +=, -=, *=, /=).
- Comparison operators: Compare values (==, !=, <, >, <=, >=)
- Logical operators: Perform logical operations (&&, ||, !)
- Increment/decrement operators: Increase/decrease variable values(++, - -).
- String operators: PHP has two operators(Concatenation, Concatenation assignment)
- Array operators : are used to compare arrays
- Conditional assignment operators: set a value depending on conditions

Arithmetic Operators in PHP

Op.	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

PHP Assignment Operators

Op.	Name	Example	Description
=	Assignment	$x = $y	Assigns the value of $y to $x.
+=	Addition ass.	$x += $y	Adds $y to $x and stores the result in $x.
-=	Subtraction ass.	$x -= $y	Subtracts $y from $x and stores the result in $x.
*=	Multiplication ass.	$x *= $y	Multiplies $x by $y and stores the result in $x.
/=	Division ass.	$x /= $y	Divides $x by $y and stores the result in $x.
%=	Modulus ass.	$x %= $y	Computes the modulus of $x divided by $y and stores the result in $x.
**=	Exponentiation ass.	$x **= $y	Raises $x to the power of $y and stores the result in $x.
.=	Concatenation ass.	$x .= $y	Concatenates $y to $x and stores the result in $x.

PHP Comparison Operators

Op.	Name	Example	Description
==	Equal	$x == $y	Returns true if $x is equal to $y.
!=	Not equal	$x != $y	Returns true if $x is not equal to $y.
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type.
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type.
>	Greater than	$x > $y	Returns true if $x is greater than $y.
<	Less than	$x < $y	Returns true if $x is less than $y.
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y.
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y.
<=>	Spaceship	$x <=> $y	Returns -1 if $x is less than $y, 0 if they are equal, and 1 if $x is greater than $y. Introduced in PHP 7.

PHP Increment/Decrement Operators

Op.	Same as...	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

Op.	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP String Operators

Op.	Name	Example	Description
.	Concatenation	$x . $y	Concatenates $x and $y
.=	Concatenation assignment	$x .= $y	Appends $y to $x

PHP Array Operators

Op.	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Ret. true if $x is not equal to $y
<>	Inequality	$x <> $y	Ret. true if $x is not equal to $y
!==	Non-identity	$x !== $y	Ret. true if $x is not identical to $y

PHP Conditional Assignment Operators

Op.	Name	Example	Result
?:	Ternary	$x = expr1 ? expr2 : expr3	Returns the value of $x. The value of $x is expr2 if expr1 is TRUE. The value of $x is expr3 if expr1 is FALSE.
??	Null coalescing	$x = expr1 ?? expr2	Returns the value of $x. The value of $x is expr1 if expr1 exists and is not NULL. If expr1 does not exist or is NULL, the value of $x is expr2. Introduced in PHP 7.

Conditional Statements - if

```php
<?php
  if (condition) {
    // Code to execute if condition is true
  } elseif (another_condition) {
    // Code to execute if another_condition is true
  } else {
    // Code to execute if none of the above conditions are
          true
  }
?>
```

Conditional Statements - Switch

```php
<?php
  switch (expression) {
    case value1:
        // Code to execute if expression equals value1
       break;
    case value2:
       // Code to execute if expression equals value2
       break;
    default:
     // Code to execute if expression doesn't match any case
  }
?>
```

While Loop

While Loop :

```php
<?php
    while (condition) {
      // Code to execute while condition is true
    }
?>
```

Do-While Loop

```php
<?php
   do {
  //Code to execute at least once, then while condition is true
   } while (condition);
?>
```

For Loop

For Loop

```php
<?php
  for ($i = 0; $i < count($array); $i++) {
    // Code to execute for each iteration
  }
?>
```

Foreach Loop

```php
<?php
  foreach ($array as $key => $value) {
    // Code to execute for each element in the array
  }
```

example - foreach

```php
<?php
  // Define an array of student names
  $students = array("John", "Alice", "Bob", "Emily");

  // Iterate over the array using foreach loop
  echo "<h2>List of Students:</h2>";
  echo "<ul>";
      foreach ($students as $student) {
        echo "<li>$student</li>";
      }
  echo "</ul>";
```

Break Statement

- Used to exit a loop prematurely.
- Terminates the current loop iteration and resumes execution at the next statement after the loop.
- Useful for exiting a loop when a certain condition is met.

```php
<?php
 for ($i = 1; $i <= 10; $i++) {
   if ($i == 5) {
     break; // Exit the loop when $i equals 5
   }
   echo $i . " ";
 }
 //output : 1 2 3 4
?>
```

Continue Statement

- Used to skip the current iteration of a loop.
- Skips the remaining code inside the loop for the current iteration and continues with the next iteration.
- Useful for skipping certain iterations based on a condition.

```php
<?php
   for ($i = 1; $i <= 10; $i++) {
     if ($i % 2 == 0) {
       continue; // Skip even numbers
     }
     echo $i . " ";
   }
   //Output : 1 3 5 7 9
```

HTML Form Structure

- **HTML <form> tag**: Defines the form to collect user input.
- **Action attribute**: Specifies the URL of the PHP script that will handle the form data.
- **Method attribute**: Defines how form data should be transmitted to the server (GET or POST)

```
<form action="process.php" method="post">
            ....
</form>
```

Method: Get and Post

- GET method: Sends form data in the URL query string.
- POST method: Sends form data in the HTTP request body.
- Use POST for sensitive or large data, as it's more secure and has no size limitations.

GET vs. POST Methods

Feature	GET	POST
Data Placement	URL (after ?)	HTTP request body
Security	Less secure (visible in URL)	More secure (hidden)
Data Size Limit	Smaller (limited by URL length)	Larger
Example Use Cases	• Search queries • Pagination • Sorting data	• Form submissions • Login credentials • File uploads

Collect form data

- **$_GET** is an array of variables passed to the current script via the URL parameters.
- **$_POST** is an array of variables passed to the current script via the HTTP POST method.

```
<form action="process.php" method="post">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username">
  <button type="submit">Submit</button>
</form>
```

```
<?php    // file : process.php
    $username = $_POST['username'];
     // Process $username...
```

Example

```
<form action="process.php" method="post">
   <label for="username">Username:</label>
   <input type="text" id="username" name="username" required>
   <label for="password">Password:</label>
   <input type="password" id="password" name="password" required>
   <button type="submit">Login</button>
</form>
```

```
<?php     // file : process.php
  if ($_SERVER["REQUEST_METHOD"] == "GET") {
     $username = $_POST['username'];
     $password = $_POST['password'];
     if ($username === "isil" && $password === "isil") {
       header("Location: /profile.html");// Redirect to profile.html
       exit; // Make sure to exit after redirection
     } else
       { echo "Invalid username or password. Please try again.";}
  }
```

Common $_SERVER Variables

- **$_SERVER["SERVER_ADDR"]**: IP address of the server hosting the script.
- **$_SERVER["SERVER_SOFTWARE"]**: Server software name and version (e.g., Apache, Nginx).
- **$_SERVER["SERVER_NAME"]**: Hostname of the server
- **$_SERVER["DOCUMENT_ROOT"]**: Root directory of the document web server.
- **$_SERVER["REQUEST_METHOD"]**: HTTP method used to submit the request (e.g., GET, POST).
- **$_SERVER["REQUEST_URI"]**: URI path of the requested resource.
- **$_SERVER["REMOTE_ADDR"]**: IP address of the user's machine.
- **$_SERVER["HTTP_REFERER"]**: URL of the referring page (if the user came from another page).
- **$_SERVER["SCRIPT_FILENAME"]**: Absolute path to the currently executing script.

$_SERVER["PHP_SELF"] variable

- **$_SERVER["PHP_SELF"]**: a super global variable that returns the filename of the currently executing script.
- **Warning** : The $_SERVER["PHP_SELF"] variable can be used by hackers!
- **Solution** : $_SERVER["PHP_SELF"] exploits can be avoided by using the ***htmlspecialchars()*** function :

```
htmlspecialchars($_SERVER["PHP_SELF"])
```

- **For more details**: https://www.w3schools.com/php/php_form_validation.asp

Information

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications.

XSS enables attackers to inject client-side script into Web pages viewed by other users

Example of hacking a page

- Create file : test_form.php

```
 <form action="<?php echo $_SERVER["PHP_SELF"]; ?>" method="post">
  <label for="name">Name:</label>
  <input type="text" name="name" id="name">
  <button type="submit">Submit</button>
</form>
```

```
<?php
 if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $name = $_POST["name"];
   echo "<p>Hello, " . $name . "!</p>";
 }
?>
```

- Click on submit, then add to the URL:
  /"><script>alert("XSS Attack!");</script>
- Modify test_form.php, and try to hack the page:
  htmlspecialchars($_SERVER["PHP_SELF"]);

Validate Form Data With PHP

- Pass all variables through PHP's htmlspecialchars()
- Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
- Remove backslashes \ from the user input data (with the PHP stripslashes() function)
- Create a function that will do all the checking :

```php
<?php
    function test_input($data) {
        $data = trim($data);
        $data = stripslashes($data);
        $data = htmlspecialchars($data);
        return $data;
    }
?>
```

Example - Validate Form Data

```php
<?php
  // define variables and set to empty values
  $name = $email = $gender = $comment = $website = "";
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = test_input($_POST["name"]);
    $email = test_input($_POST["email"]);
    $website = test_input($_POST["website"]);
    $comment = test_input($_POST["comment"]);
    $gender = test_input($_POST["gender"]);
  }
  function test_input($data) {
    $data = trim($data);
    $data = stripslashes($data);
    $data = htmlspecialchars($data);
    return $data;
  }
?>
```

Common PHP Form Vulnerabilities (1)

- **XSS (Cross-Site Scripting)**: Unsanitized user input leading to execution of malicious scripts.
- **SQL Injection**: Improperly sanitized form inputs allowing unauthorized SQL commands.
- **CSRF (Cross-Site Request Forgery)**: Lack of CSRF tokens enabling actions on behalf of users.
- **File Upload Vulnerabilities**: Inadequate validation of uploaded files permitting execution of malicious scripts.

Common PHP Form Vulnerabilities (2)

- **Form Spoofing**: Deceptive creation of fake forms to capture sensitive information.
- **Parameter Tampering**: Manipulation of form parameters leading to unauthorized access or data manipulation.
- **Session Fixation**: Poor session management enabling hijacking of user sessions.
- **Denial of Service (DoS)**: Lack of input validation or rate limiting making the server vulnerable to flooding attacks.

Homework Assignment: PHP Form Validation

Objective:

- Make input fields required in your PHP form.
- Create error messages if required fields are left empty.

Instructions:

- Use the `required` attribute in HTML input fields
- Implement PHP validation to check for empty fields
- Display error messages next to the input fields if they are left empty
- Ensure to implement validation and sanitization to avoid vulnerabilities
- Use CSS to style the error messages for better visibility.

Helpful Resource:

- https://www.w3schools.com/php/php_form_required.asp

Classes and Objects

- Classes and objects are the two main aspects of object-oriented programming
- A class is a template for objects, and an object is an instance of a class
- When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties

Define a Class

```php
<?php
  class ClassName {
    // Properties (attributes)
    // Methods (functions)
  }
?>
```

Example Class Definition

```php
<?php
  class Person {
    private $name;
    private $age;

    public function __construct($name, $age) { // Constructor
      $this->name = $name;
      $this->age = $age;
    }
    public function getName() {// Getter for name
      return $this->name;
    }
    public function setName($name) {// Setter for name
      $this->name = $name;
    }
    public function getAge() {// Getter for age
      return $this->age;
    }
    public function setAge($age) { // Setter for age
      $this->age = $age;
    }
  }
//....
```

Constructors

- A constructor is a special method in a PHP class.
- It's automatically called when you create a new object
- Its purpose is to initialize the object's state (properties)
- It can accept arguments to customize the initial state.
- The constructor name must match the class name.

```php
<?php
  class Fruit {
   public $name;
   public $color;
   function __construct($name, $color) {
     $this->name = $name;
     $this->color = $color;
   }
  }
  $apple = new Fruit("Apple", "red");
```

Destructors

- A destructor is a special method named `__destruct` in a PHP class.
- It's automatically called when an object is destroyed (goes out of scope).
- Its purpose is to perform any cleanup tasks before the object is removed from memory.
- This might involve closing database connections, releasing resources, or logging information
- If you create a `__destruct()` function, PHP will automatically call this function at the end of the script

Destructor Example

```php
<?php
 class DatabaseConnection {
   private $connection;
   function __construct($host, $username, $password, $database) {
     $this->connection = new mysqli($host, $username, $password, $database);
     if ($this->connection->connect_error)
       {die("Connection failed: " . $this->connection->connect_error);}
   }
   function __destruct() {
     if (isset($this->connection)) {
       $this->connection->close();
       echo "Database connection closed.\n";
     }
   }
 }
 $db = new DatabaseConnection("localhost", "myuser", "mypassword",
         "mydb");
 // When $db goes out of scope or is explicitly destroyed, the destructor
         closes the connection
 //isset() :  check if a variable is set and is not null
 //die() :  immediately terminate script execution and display an optional
         message
```

Access Modifiers in PHP

- **public**: Accessible everywhere.
- **protected**: Accessible within class and subclasses.
- **private**: Accessible only within the class.

Creating Objects in PHP

Syntax for creating objects in PHP

```php
<?php
    $objectName = new ClassName();
 ?>
```

Example : Create an object of Person

```php
<?php
    $person1 = new Person("Alice", 30);
    echo $person1->getAge();
?>
```

Inheritance

- Inheritance is a way to create a new class from an existing class
- The new class inherits properties and methods from the existing class
- It is a way to reuse code and establish a hierarchical relationship

Why Use Inheritance?

- Code Reusability
- Reduced Redundancy
- Improved Maintainability
- Hierarchical Relationships

```php
<?php
class ClassName {
    // Class properties and methods go here
}

class ChildClass extends ParentClass {
  // Additional or overridden properties and methods
}
```

Example - Class Person

```php
<?php
  class Person {
   public $name;
   public $age;

   public function __construct($name, $age) {
     $this->name = $name;
     $this->age = $age;
   }

   public function displayInfo() {
     echo "Name: " . $this->name . ", Age: " . $this->age;
   }
  }
```

Example - Class Employee

```php
<?php
class Employee extends Person {
   public $salary;
     public $department;

  public function __construct($name, $age, $salary, $dprtmnt){
    parent::__construct($name, $age);
    $this->salary = $salary;
    $this->department = $dprtmnt;
  }
  public function displayInfo($showSalary = false) {
    parent::displayInfo();
    if ($showSalary) {
      echo ", Salary: " . $this->salary;
    }
    echo ", Department: " . $this->dprtmnt;
  }
 }
```

Example - Create Object

```php
<?php
$person = new Person("John Doe", 30);

$person->displayInfo();
// Output: Name: John Doe, Age: 30

$employee = new Employee("Jane Smith", 35, 50000, "Marketing");

$employee->displayInfo();
/* Output: Name: Jane Smith, Age: 35, Department: Marketing*/

$employee->displayInfo(true);
/* Output: Name: Jane Smith, Age: 35, Salary: 50000,
      Department: Marketing  */
```

Common Error Types in PHP

- **Syntax Errors:** Typos, missing semicolons, mismatched parentheses (e.g., `$variable = 5` instead of `$variable = 5;`)
- **Runtime Errors:** Occur during execution (e.g., division by zero, file not found)
- **Logic Errors:** Flaws in program logic (e.g., incorrect calculation, infinite loop)
- **Database Errors:** Issues with database connections, queries (e.g., invalid SQL syntax)

Basic Error Handling: Using the die() function

- The die() function displays a message and immediately terminates script execution.
- Use die() for critical errors where continued execution is pointless.

```php
<?php
  $connection = mysqli_connect("host", "user", "password",
        "database");
  if (!$connection) {
    die("Connection failed: " . mysqli_connect_error());
  }
  echo "Connected successfully";
?>
```

However, simply stopping the script is not always the right way to go

Creating a Custom Error Handler

- Create a special function that can be called when an error occurs

```
error_function(error_level,error_message,
               error_file,error_line,error_context)
```

Parameters :

- **error_level**: Required. The level of the error raised.
- **error_message**: Required. The error message.
- **error_file**: Optional. The filename in which the error was raised.
- **error_line**: Optional. The line number in which the error was raised.
- **error_context**: Optional. Specifies an array containing every variable, and their values, in use when the error occurred

Example - Custom Error Handler

```php
<?php
function myErrorHandler($severity, $message, $file, $line) {
  throw new ErrorException($message, 0, $severity, $file,
        $line);
}

set_error_handler("myErrorHandler");
```

Essentials of Exception Handling in PHP

Proper exception code should include:

- **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

```php
<?php
try {
  // Code that may throw an error or exception
} catch (Throwable $t) {
  // Handle error or exception
}
```

Example - Exception

```php
<?php

try {
  // Risky code (might throw an error)
  $result = 10 / 0; //  DivisionByZeroError
} catch (Throwable $t) {
  echo "Something went wrong: " . $t->getMessage();
}

echo "Script continues...";
?>
```

File management in PHP

```php
<?php
$file = fopen("example.txt", "r"); // Opening a File

//Reading a File
$content = fread($file, filesize("example.txt"));

fwrite($file, "Hello, World!");//Writing to a File

fclose($file); //Closing a File
```

PHP also provides functions to copy (**copy()**), move (**rename()**), delete (**unlink()**) files, and check if a file exists (**file_exists()**).

The opendir() function can be used to open a directory, and readdir() to read its content

Remote Files

Ensure the **allow_url_fopen** directive is set to **On** in your php.ini file:

php.ini

```
allow_url_fopen = On
```

Reading Remote Files

using **file_get_contents()**:

```
    <?php
$content =
      file_get_contents('http://www.example.com/somefile.txt');
echo $content;
```

Or with **fopen()**

```
    <?php
    $file = fopen("http://www.example.com/somefile.txt", "r");
    if ($file) {
      while (!feof($file)) {
        $line = fgets($file);
        echo $line;
      }
      fclose($file);
    }
```

Writing to Remote Files

Using **cURL** to Send Data

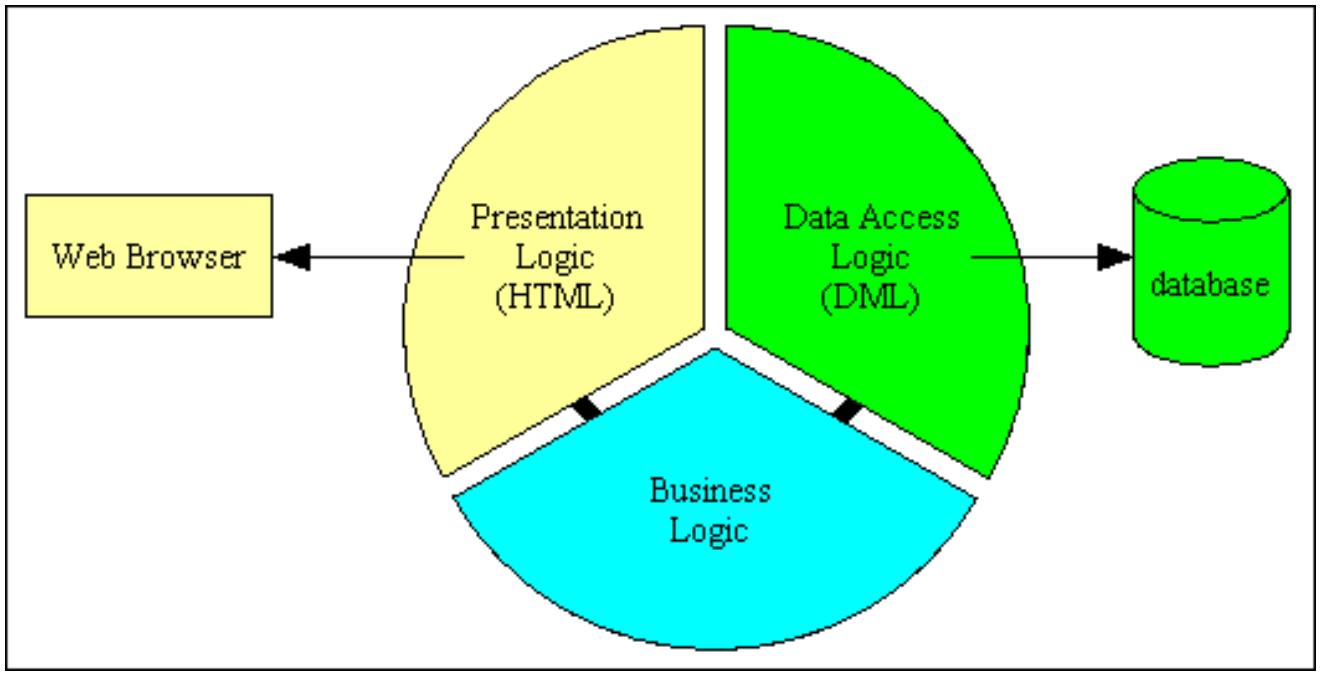
```php
    <?php
$ch = curl_init();

$data = array('key' => 'value');
curl_setopt($ch, CURLOPT_URL,
      "http://www.example.com/receive.php");
curl_setopt($ch, CURLOPT_POST, true);
curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_query($data));

$response = curl_exec($ch);
curl_close($ch);
```

Security: When enabling **allow_url_fopen** and handling remote files:

Web 3-tier Architecture in PHP

Web 3-tier architecture is a design pattern for developing scalable and maintainable web applications. It divides the application into three layers:

- Presentation Layer
- Business Logic Layer
- Data Access Layer



Presentation Layer

The front-end part of the application. Concerned with how the application looks and interacts with users

- Receives HTTP requests
- Handles user interactions
- Implemented using HTML, CSS, JavaScript, PHP
- Components: web pages, forms, UI

Business Logic Layer

The heart of the application

- Classes and methods that handle the processing of data, such as user authentication, data validation, and business rules.
- Communicates with the presentation layer to send data to be displayed and interacts with the data layer to retrieve, manipulate, and store data

Data Access Layer

This bottom layer manages the storage and retrieval of data

- Interacts with data storage
- Handles database operations
- Implemented using PDO, MySQLi
- Components: DAOs(Data Access Objects), repositories

Questions ?