University of Anbar

College of Computer Sciences and

Information Technology

Web Development

Dr. Alaa Abdalqahar Jihad

Fourth Stage – Second Course

Lecture 5: Functions and Code Reusability

Objectives:

- Understand the purpose and advantages of using functions in PHP.
- Define and invoke functions.
- Work with parameters (including default values and passing by reference).
- Return values from functions.
- Recognize the concept of variable scope (local vs. global).
- Understand and use anonymous functions (closures) when necessary.

Introduction to Functions

- **What is a Function?**
    - A block of reusable code designed to perform a specific task.
    - Helps in reducing code redundancy and enhances readability.
- **Advantages of Using Functions:**
    - **Modularity:** Breaks down complex tasks into smaller, manageable parts.
    - **Reusability:** Write once, use many times across the application.
    - **Maintainability:** Easier to debug and update code.

Defining and Calling Functions

- **Syntax of a PHP Function:**

```
<?php
function greet($name) {
   return "Hello, " . $name . "!";
}
echo greet("Ali"); // Outputs: Hello, Ali!
?>
```

- **Understanding Function Parameters and Return Values:**
    - **Parameters:** Variables passed into the function.
    - **Return Values:** What the function sends back to the caller.

Function Scope and Variable Management

- **Local vs. Global Variables:**
    - Variables declared inside a function are local.
    - Global variables need to be declared using the global keyword if accessed within a function.
- **Example:**

```
<?php
$message = "Hello from global!";
function showMessage() {
   global $message;
   echo $message;
}
showMessage(); // Outputs: Hello from global!
?>
```

Advanced Function Features

- **Default Parameters:** Provide default values to parameters so the function can be called with fewer arguments.

```
<?php
function greet($name = "Guest") {
   return "Hello, " . $name . "!";
}
echo greet(); // Outputs: Hello, Guest!
?>
```

- **Passing Parameters by Reference:** Allows the function to modify the original variable.

```
<?php
function addFive(&$number) {
   $number += 5;
}
$value = 10;
addFive($value);
echo $value; // Outputs: 15
?>
```

- **Anonymous Functions (Closures):** Useful for creating inline functions or callback functions.

```
<?php
$numbers = [1, 2, 3, 4];
$squared = array_map(function($n) {
   return $n * $n;
}, $numbers);
print_r($squared); // Outputs squared values of numbers
?>
```

Built-In Functions and Code Reusability

- **Overview of PHP Built-In Functions:** Emphasize the vast array of functions available (e.g., strlen(), substr(), array_merge(), etc.).

Practical Example:

Task: Create a PHP script that:

1. Defines a function to calculate the factorial of a number.
2. Uses a loop or recursion inside the function.
3. Calls the function with a sample number (e.g., 5) and prints the result.

Solution:

```
<?php
function factorial($n) {
    if ($n <= 1) {
        return 1;
    } else {
        return $n * factorial($n - 1);
    }
}
echo "Factorial of 5 is: " . factorial(5); // Expected output: Factorial of 5 is: 120
?>
```

Summary

- **Functions** allow you to encapsulate code into reusable blocks.
- **Parameters** and **return values** facilitate dynamic behavior in functions.
- Understanding **scope** is essential for managing variable access.
- Advanced features like **default parameters**, **passing by reference**, and **anonymous functions** empower you to write flexible and maintainable code.

Homework Assignment

Create a PHP script that:

1. Defines several functions:
    - One to convert temperatures from Celsius to Fahrenheit.
    - Another to calculate the area of a rectangle.
2. Demonstrates the use of default parameters and passing by reference.
3. Calls these functions with different values and displays the results.

Lecture 6: Forms and Data Handling

Objectives:

- Understand the purpose and structure of HTML forms in web development.
- Identify various form elements (text fields, radio buttons, checkboxes, dropdowns, etc.).
- Differentiate between the GET and POST methods.
- Use PHP superglobals ($_GET and $_POST) to capture and process form data.
- Validate and sanitize user inputs to improve security.
- Create a simple contact or registration form and handle its data on the server side.

Introduction to HTML Forms

- **Purpose of Forms:**
    - Collect user input, such as contact information, survey responses, or login credentials.
    - Enable interaction between users and web applications.
- **HTML Form Structure:**
    - Basic tags: <form>, <input>, <textarea>, <select>, <button>.
    - Attributes: action, method (GET vs. POST), and name for inputs.

- **Example: Simple HTML Form Structure:**

```
<form action="process.php" method="POST">
   <label for="name">Name:</label>
   <input type="text" id="name" name="name"><br><br>
   <label for="email">Email:</label>
   <input type="email" id="email" name="email"><br><br>
   <input type="submit" value="Submit">
</form>
```

Understanding GET vs. POST Methods

- **GET Method:**
    - Appends data to the URL.
    - Suitable for non-sensitive data and simple queries.
    - Limitations: URL length constraints and visibility of data.
- **POST Method:**
    - Sends data in the request body.
    - Better for sensitive information (e.g., passwords, personal data).
    - No size limits like GET.

Retrieving and Processing Form Data in PHP

- **PHP Superglobals:**
    - $_GET for data sent via the GET method.
    - $_POST for data sent via the POST method.

- **Example Code: (The name of the file must be the process.php)**

```
<?php
// process.php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST['name'] ?? '';
    $email = $_POST['email'] ?? '';

    // Basic output
    echo "Name: " . htmlspecialchars($name) . "<br>";
    echo "Email: " . htmlspecialchars($email);
}
?>
```

- **Key Points:**
    - Use conditional checks to ensure data is available.
    - Introduce htmlspecialchars() to prevent Cross-Site Scripting (XSS).

Data Validation and Sanitization

- **Importance of Validation:**
    - Ensuring data meets expected formats (e.g., valid email, required fields).
- **Sanitization Techniques:**
    - Using PHP functions like filter_var() to validate inputs.
    - Example: Validate email format

```php
<?php
$email = $_POST['email'] ?? '';
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
   echo "Invalid email format.";
} else {
   echo "Valid email!";
}
?>
```

- **Error Handling:**
    - Providing user feedback when validation fails.
    - Implementing measures to avoid SQL Injection (if data will be stored in a database later).

Security Considerations

- **Common Vulnerabilities:**
    - XSS (Cross-Site Scripting): Use htmlspecialchars() when outputting data.
    - CSRF (Cross-Site Request Forgery): Consider using tokens for sensitive operations.
- **Notes:**
    - Always validate and sanitize user input.
    - Use prepared statements if integrating with a database.
    - Keep error messages generic to avoid leaking sensitive information.

Example

Task: Create a PHP application that:

1. Displays a form asking for a user's name, email, and a short message.
2. Processes the form data in a separate PHP file.
3. Validates the inputs (e.g., check if fields are not empty, validate email format).
4. Sanitizes the data before displaying it on the screen.

Example Code:

- **HTML Form (index.html):**

```
<form action="process.php" method="POST">
    <input type="text" name="name" placeholder="Your Name">
    <input type="email" name="email" placeholder="Your Email">
    <textarea name="message" placeholder="Your Message"></textarea>
    <input type="submit" value="Send">
</form>
```

- **PHP Processing (process.php):**

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = trim($_POST['name'] ?? '');
    $email = trim($_POST['email'] ?? '');
    $message = trim($_POST['message'] ?? '');
    // Validate inputs
    if (empty($name) || empty($email) || empty($message)) {
```

```
        echo "All fields are required.";
        exit;
    }

    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
        echo "Invalid email format.";
        exit;
    }
    // Sanitize outputs
    $name = htmlspecialchars($name);
    $email = htmlspecialchars($email);
    $message = htmlspecialchars($message);
    echo "Thank you, $name!<br>";
    echo "We have received your message: $message<br>";
    echo "We will contact you at: $email";
}
?>
```

Summary

- The structure and purpose of HTML forms.
- Using GET vs. POST methods.
- Capturing, validating, and sanitizing form data in PHP.

Homework Assignment:

Create a PHP-based feedback form that:

1. Collects user feedback (name, email, feedback message).
2. Validates that no field is left empty and the email is in the proper format.
3. Sanitizes all user inputs before displaying or storing them.
4. Provides user-friendly error messages for invalid inputs.