Department of Information Technology - State Polytechnic of Malang

**Jobsheet-7: PHP – Form Processing Web Design and Programming Courses** Web Design and Programming Teaching Team October 2024

**Topic**

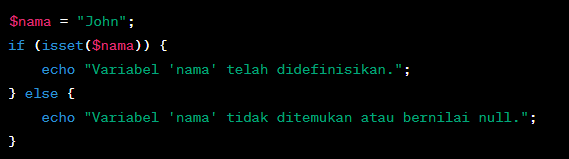
- Form Proccessing Concept with PHP and Jquery

**Objective**

Students are expected to:

1. Students are able to create forms using PHP
2. Students are able to create forms using jQuery

**Introduction**

**Function isset isset()** is a function in PHP that is used to check whether a variable has been defined (exists) or not. This function returns **true** if the variable has been defined and has a value, and **false** if the variable does not exist or has a null value. Here's an example script and a minimal explanation of **isset():**

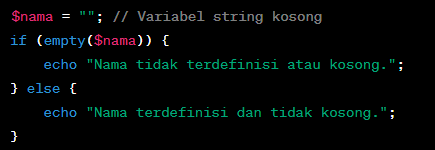
The above code checks if **$nama** variable has been defined. If yes, then the message "Variable 'name' has been defined." will be displayed, if not, then the message "Variable 'name' not found or null value." will be displayed.

# Practical Section 1. Function isset()

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | Create a new folder **week7** and name it **isset.php** |
| 2 | Type into the **isset.php** file the code below. |
| 3 |  |
| 4 | Save the file, then open a browser and run **localhost/week7/isset.php**. |

|  |  |
| --- | --- |
|  | What do you understand from using the isset on the file? (Question No. 1)    The isset function in PHP checks whether a variable is set and not null. It's commonly used to ensure that variables are defined before accessing them. |
| 5 | Add the contents of the **isset.php** file with the code below. |
| 6 | Save the file, then open the browser and run **localhost/week7/isset.php**. Ensure that the output does not appear in a single line; the result from the **echo** should be displayed separately. Explain what you understand from the use of **isset()** in that file. Write your understanding below. (Question No. 2)    When you add more code and check the behavior of isset, it demonstrates how the function evaluates the defined state of variables. Each echo result displayed separately illustrates clarity in debugging. |

**Function empty()** The **empty()** function in PHP is used to check whether a variable is empty or undefined. This function returns **true** if the variable is empty or undefined, and **false** if the variable has a value or has been defined. Here is an explanation and example of using **empty():**



The **empty()** function can be used to check whether a string is empty or not.

**Practical Section 2. Function** empty()

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | Create a new file named **empty.php** inside **week7** folder. Write this code into **empty.php** file |

|  |  |
| --- | --- |
| 2 |  |
| 3 | Save the file, then open a browser and run **localhost/week7/empty.php**  What do you understand from the use of empty on the file? Write your understanding below. (Question No. 3)    The empty function checks if a variable is either undefined or has a value considered empty, such as "", 0, NULL, or an empty array. |
| 4 | Add the contents of the **empty.php** file with the code below |
| 5 | Save the file, then open the browser and run **localhost/week7/empty.php**. Ensure that the output does not appear in a single line; the result from the **echo** should be displayed separately. Explain what you understand from the use of **empty()** in that file. Write your understanding below. (Question No. 4)    By testing scenarios with non-empty and empty values, you see how empty distinguishes between the two, helping validate input or conditions. |

# Practical Section 3. PHP Input Form

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | Create a new file named **form.php**. Write this code below. |

|  |  |
| --- | --- |
| 2 |  |
| 3 | Create a new file named proses\_form.php. Type the code below inside proses\_form.php |
| 4 | Save the file, then open a browser and run localhost/week7/proses\_form.php. Explain what happened and write your understanding below.  Then run localhost/week7/form.php. Explain what happened and write your understanding below.  (Question No. 5)  ‘  form.php is the input interface, while proses\_form.php processes the submitted data. This demonstrates form submission workflows. |
| 5 | Create a new file named form\_self.php. Type the code below. |

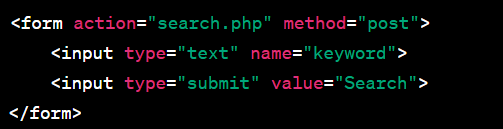
|  |  |
| --- | --- |
|  |  |
| 6 | Save the file, then open a browser and run localhost/week7/ form\_self.php. What do you understand from the use of forms in the file? Write your understanding below. (Question No. 6)    This shows how a form can process its data on the same page, reducing navigation and improving user experience. |

**HTML Injection** HTML injection (also known as an "HTML injection attack" or "client-side injection") is a security attack that occurs when an attacker injects malicious HTML or JavaScript code into the input received by a web application. The malicious code will be executed by the user's browser viewing the affected web page, which could result in illegal access to data, changes in page view, or other attacks.

An attacker can try to inject malicious HTML or JavaScript code into the input received by the web application. If the web application does not properly sanitize or escape input, then the malicious code will be executed by the user's browser, which can cause security issues.

# Example HTML Injection:

For example, we have a simple search form on a website that searches for keywords among user reviews:



Now, if a web application doesn't validate the input correctly, an attacker can enter malicious input like this:

If the web application does not avoid or clean up this input before displaying it on the search results page, then the result will look like this:



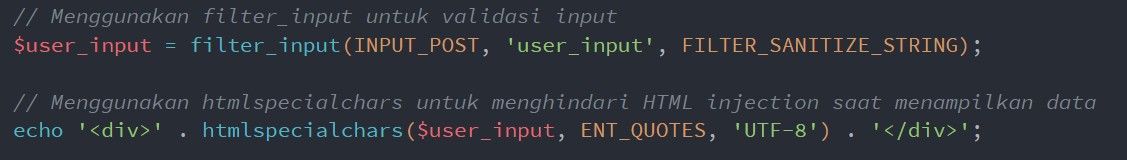
The malicious JavaScript code will be executed by the user's browser and will bring up a "You have been hacked!" warning box. This is a simple example of HTML injection. As a web developer, it is important to always sanitize and validate input from users, as well as avoid displaying user input directly on web pages without proper escaping or sanitation to avoid these kinds of attacks.

# How to Handle HTML Injection

The way to handle HTML injection is to validate, filter, and avoid displaying user input without proper escaping. Here are some steps to protect your app from HTML injection:

1. **Input Validation**: Validate input from users to ensure that only valid data is received. You can use functions such as **filter\_input()** or **filter\_var()** to perform validation.
2. **Filter Input**: Filter user input to remove or replace potentially harmful characters, such as **<**, **>**, **&**, and more. You can use functions like **htmlspecialchars()** or **strip\_tags()** for this.
3. **Parameterized Statements (Query):** If you generate SQL queries with user input, use parameterized statements or prepared statements to prevent SQL injection, which can be a form of attack similar to HTML injection.
4. **Content Security Policy (CSP):** Apply a Content Security Policy (CSP) to your HTTP headers to control the resources that can be used within your web pages.
5. **Escape Output**: When you display data on a web page, make sure you avoid JavaScript injection by using **htmlspecialchars()** or similar methods.

Here's a simple example in PHP to solve HTML injection:



By using these steps, you can mitigate the risk of HTML injection in your web application. It's always important to validate inputs, clean incoming data, and avoid displaying user data without proper escaping.

# Practical Section 4. HTML Injection

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | Create a new file named **html\_safe.php** inside **week7** folder. Type the code below. |
| 2 |  |
| 3 | Create a file with the name **form1.php**, fill it with code to create a form with 1 text input to be processed on the **html\_safe.php**. |
| 4 | Save the file, open the browser and run **localhost/week7/form1.php**  Fill with input that contains the html tag. |
| 5 | Next, delete the code **$input = htmlspecialchars($input, ENT\_QUOTES, 'UTF-8');**  Save the file, open the browser and run /refresh **localhost/week7/form1.php**  Refill with input that contains the html tag. |
| 6 | Record here what you observed, give your explanation. (Question No. 7)    Removing htmlspecialchars exposes the page to HTML injection, where raw HTML tags in input are rendered on the page, potentially causing vulnerabilities. |

|  |  |
| --- | --- |
|  |  |
| 7 | Type the code below on the **html\_safe.php** |
| 8 | A screen shot of a computer program  Description automatically generated |
| 9 | Complete the program code in the if and else sections so that it can run correctly. |
| 10 | In **form1.php**, add code to create an email input that will be processed on **html\_safe.php**. Save the file, open the browser and run /refresh **localhost/week7/form1.php** |
| 11 | Note here what you observe from the addition of the program code above. (Question No. 8)    This expands functionality to process and sanitize email inputs, reinforcing input validation best practices. |

**Regular Expression (Regex)**

Regex is a powerful tool for searching, matching, or manipulating text based on specific patterns. You can use it for various purposes such as input validation, text search, text replacement, and more.

Basic Regex Patterns:

* **Single Character**: Any character will match itself in Regex, except for special characters that must be avoided with escape characters.
  + Example: The **/a/** pattern will match the letter "a" in the text.
* **Specific Characters**: You can match specific characters by mentioning them in a pattern.
  + Example: **The /hello/** pattern will match the text "hello" in the text.
* **Special Characters**: Some special characters in Regex should be avoided with escape characters (**\**) if you want to match them literally. Examples of special characters: **.**, **\***, **+**, **?**, **|**, **[**, **]**, **(**, **)** , **{**, **}**, **^**,

**$**, **\**.

* **Character Set**: You can match characters from a set of characters by using **[ ]**. For example,

**/[aeiou]/** will match one of the vowels.

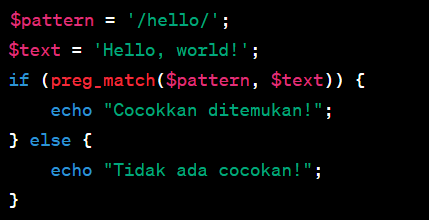
* **Character Range**: You can use **-** in a character set to specify a character range. For example, **/[a- z]/** will match any lowercase letters.
* **Quantifier**: You can set the number of times a character or group of characters should appear beforehand. Example quantifier:
  + **\***: 0 or more

 **+**: 1 or more

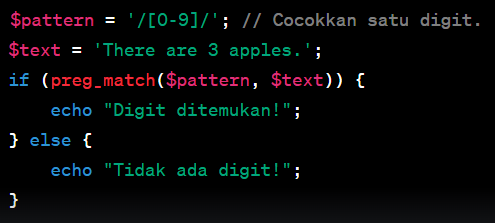
* + **?**: 0 or 1 time
  + **{n}**: Exactly **n** times
  + **{n,}**: At least **n** times
  + **{n,m}**: At least **n** times, maximum **m** times

# Examples of Regex Usage:

1. Matching patterns



1. Match set characters



# Practical Section 5 : The Use of Regex in PHP

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | Create a new file named **regex.php** inside **week7** folder. Type the code in step 2 |
| 2 |  |
| 3 | Save the file, then open a browser and run/refresh **localhost/week7/regex.php** |
| 4 | Note here what you observe from the addition of the program code above. (Question No. 9)    Regex efficiently matches or processes strings based on patterns. Observations include how it validates specific formats or replaces characters. |
| 5 | Type the code below inside the **regex.php** |

|  |  |
| --- | --- |
| 6 |  |
| 7 | Complete the program code above so that the result is neat. Save the file, then open a browser and run /refresh **localhost/week7/regex.php** |
| 8 | Note here what you observe from the addition of the program code above. (Question No. 10)    Proper formatting improves readability and understanding of regex operations. |
| 9 | Type the additional code below inside the **regex.php** |
| 10 |  |
| 11 | Complete the program code above so that the result is neat. Save the file, then open a browser and run /refresh **localhost/week7/regex.php** |
| 12 | Note here what you observe from the addition of the program code above. (Question No. 11)    The \* quantifier matches 0 or more occurrences, while + matches 1 or more. |
| 13 | Type the additional code in step 14 inside the **regex.php** |
| 14 |  |
| 15 | Complete the program code above so that the result is neat. Save the file, then open a browser and run /refresh **localhost/week7/regex.php** |
| 16 | Note here what you observe from the addition of the program code above. (Question No. 12)    Using quantifiers like {n,m} allows more control over matches. |

|  |  |
| --- | --- |
|  |  |
| 17 | In the script in step 14, change the variable pattern from '\*' to '?'.  Save the file, then open a browser and run /refresh **localhost/week7/regex.php**  Note here what you observe from the addition of the program code above. (Question No. 13)    This matches 0 or 1 occurrence of the preceding element. |
| 18 | In the script in step 14, change the variable pattern to **'/[o]{1,3}/'**.  Save the file, then open a browser and run /refresh **localhost/week7/regex.php**  Note here what you observe from the addition of the program code above. (Question No 14)    Using patterns like /[o]{1,3}/ matches the letter "o" between 1 to 3 times. |

**Practical Section 6 : Advanced Form**

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | Create a new file named **form\_lanjut.php** inside **week7** folder. Type the code in step 2 inside the **form\_lanjut.php** |
| 2 |  |
| 3 | Create a new file named **proses\_lanjut.php** inside the **week7**. Type the following code |

|  |  |
| --- | --- |
|  |  |
| 4 | Save the file, then open a browser and run /refresh **localhost/week7/form\_lanjut.php** |
| 5 | Note here what you observe from the program code above. (Question No. 15)      Adding more fields and controls improves usability and demonstrates complex form handling. |
| 6 | Create a new file named **form\_ajax.php.** Type the following code |

|  |  |
| --- | --- |
|  |  |
| 7 | Save the file, then open a browser and run /refresh **localhost/week7/form\_ajax.php**.  Note here what you observe from the addition of the program code above. (Question No. 16)    If the backend script (proses\_ajax.php) doesn't sanitize input, it may become vulnerable to **XSS (Cross-Site Scripting)**. Always use functions like htmlspecialchars to escape user input. |

# Practical Section 7 : Form Validation

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | Create a new file named **form\_validation.php** inside **week7** folder. Type the code below. |
| 2 | Create a new file named **proses\_validasi.php** inside **week7** folder. Type the code below. |
| 3 | Save the file, then open a browser and run /refresh **localhost/week7/form\_validation.php** |

|  |  |
| --- | --- |
| 4 | Note here what you observe from the addition of the program code above. (Question No. 17)      **Pros**: Secure, ensures the form cannot bypass validation by disabling JavaScript.  **Cons**: Feedback isn't instant; the page reloads after form submission. |
| 5 | Change form\_validation.php file. Type the code below |
| 6 |  |

|  |  |
| --- | --- |
| 7 | Save the file, then open a browser and run /refresh **localhost/week7/form\_validation.php** |
| 8 | Note here what you observe from the addition of the program code above. (Question No. 18)  **Dynamic Feedback**: The validation using jQuery provides immediate feedback to the user without requiring a page reload. If the password is less than 8 characters, the user is notified instantly.  **Prevents Invalid Submissions**: The e.preventDefault() function stops the form from being submitted if the password doesn't meet the required conditions, ensuring only valid data reaches the server.  **Improved User Experience**: The user can correct errors in real-time rather than waiting for server-side validation, which reduces frustration and improves usability.  **Potential Limitation**: This approach relies on JavaScript, meaning users with disabled JavaScript can bypass validation. Pairing with server-side validation is necessary for robust security. |
| 9 | Create a script for step 6 using ajax. Screen shoot the code and wrote here what you observe from the addition of the program code.  (Question No. 19)    **Asynchronous Submission**: The form is submitted using AJAX, allowing the page to remain static while the server processes the request. This improves the overall user experience by avoiding unnecessary page reloads.  **Real-Time Error Handling**: Combining AJAX with jQuery validation ensures that users only send valid data to the server. Invalid inputs are caught in the browser, and error messages are displayed dynamically.  **Security Benefits**: Even with client-side validation, the server (via validate\_password.php) ensures the password meets the required conditions. This dual-layer approach prevents bypassing validation by disabling JavaScript.  **Efficiency**: By validating inputs on the client-side first, fewer invalid requests are sent to the server, reducing server load. |
| 10 | Add code for password validation with a minimum of 8 characters using jQuery and PHP.  Screen shoot the code and note here what you observe from the addition of the program code. (Question No. 20)    **Advanced Rules**: The addition of regex patterns in jQuery allows more sophisticated password validation. For example, checking for a combination of letters, numbers, and special characters.  **User-Friendly Messages**: The feedback mechanism explains exactly why the password is invalid, such as lacking a number or being too short. This guides users in fixing their input.  **Real-Time Corrections**: The form dynamically validates the password as the user types. Once the conditions are met, the feedback updates to indicate the password is valid.  **Scalability**: This approach can be expanded to include more complex requirements like enforcing uppercase letters, disallowing common words, or blacklisting weak passwords. |

Nama : Bayu Putra L

Class : 2I

Number : 06

Link GitHub : <https://github.com/putra-afk/Web-Programming/tree/main/week7>