

Jobsheet-7: PHP – Form Processing

Kevin Bramasta Arvyto Wardhana

2341720233 / 13

Practical Section 1. Function isset()

Code:

```
<?php

$age;

if (isset($age) && $age >= 10) {
    echo "You are an adult";
} else {
    echo "You are not an adult or the 'age' variable does not exist";
}
```

Output:

```
You are not an adult or the 'age' variable does not exist
```

What do you understand from using the isset on the file?

(Question No. 1)

the isset() function is used to check if the variable \$age has been set or declared. If the variable \$age has been set and its value is greater than or equal to 10, then the message "You are an adult" is echoed. Otherwise, if the variable \$age is not set or if its value is less than 10, the message "You are not an adult or the 'age' variable does not exist" is echoed.

Modification 1

Code:

```
<?php

$data = array(
    "name" => "John",
    "age" => 30
);

if (isset($data["name"])) {
    echo "Name: ". $data["name"];
} else {
    echo "Variable 'name' not found in the array.";
}
```

Output:

Name: John

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)

In the updated code snippet, the `isset()` function is used to check if the key "name" exists in the associative array `$data`. If the key "name" exists in the array, it will output the value associated with that key, which in this case is "John", preceded by the text "Name: ". If the key "name" does not exist in the array, it will output the message "Variable 'name' not found in the array."

This usage of `isset()` ensures that the script checks for the existence of a specific key in the array before attempting to access its value. It helps prevent errors that may occur when trying to access a non-existent key in an array.

Practical Section 2. Function `empty()`

Code:

```
<?php

$myArray = array();

if (empty($myArray)) {
    echo "Array is not defined or empty.";
} else {
    echo "Array is defined and not empty.";
}
```

Output:

```
Array is not defined or empty.
```

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 is used to check if the array \$myArray is empty or not. If the array is either not defined or has no elements (i.e., it is empty), then the message "Array is not defined or empty." is echoed. On the other hand, if the array is defined and contains one or more elements, the message "Array is defined and not empty." is echoed.

Modification 1

Code:

```
<?php

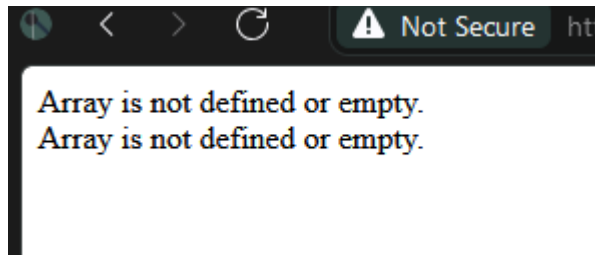
$myArray = array();

if (empty($myArray)) {
    echo "Array is not defined or empty.";
} else {
    echo "Array is defined and not empty.";
}

echo "\n";

if (empty($nonExistentArray)) {
    echo "Array is not defined or empty.";
} else {
    echo "Array is defined and not empty.";
}
```

Output:



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)

the code attempts to check the \$nonExistentArray using empty(). Since \$nonExistentArray has not been defined or initialized, it is considered empty, and the message "Array is not defined or empty." is echoed.

Practical Section 3. PHP Input Form

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h2>Form Input PHP</h2>
  <form action="proses_form.php" method="post">
    <label for="nama">Nama:</label>
    <input type="text" name="nama" id="nama" required><br><br>

    <label for="email">Email:</label>
    <input type="email" name="email" id="email" required><br><br>

    <input type="submit" value="Submit" name="submit">
  </form>
</body>
</html>
```

```
<?php

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST["name"];
    $email = $_POST["email"];
    echo "Name: $name<br>";
    echo "Email: $email<br>";
}
```

Output:

Form Input PHP

Name:

Email:

Name: kevin
Email: kevinbramastaa@gmail.com

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)

When running form.php, the HTML form titled "Form Input PHP" will be displayed in the browser. Users can input their name and email in the respective fields and click the "Submit" button. Upon submission, the form data will be sent to process_form.php for processing. The PHP script in process_form.php will then extract the name and email values from the form data and display them on the page.

Next

Code:

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>

<body>
  <h2>Form Input PHP</h2>
  <?php
    $nameErr = "";
    $name = "";

    if ($_SERVER["REQUEST_METHOD"] == "POST") {
      if (empty($_POST["name"])) {
        $nameErr = "Name is required";
      } else {
        $name = $_POST["name"];
        echo "Data Successfully Submitted";
      }
    }
  ?>

  <form action="<?php echo htmlspecialchars(string: $_SERVER["PHP_SELF"]); ?>" method="post">
    <label for="name">Name:</label>
    <input type="text" name="name" id="name" value="<?php echo $name; ?>">
    <span <?php echo $nameErr; ?></span><br><br>

    <input type="submit" value="Submit" name="submit">
  </form>
</body>

</html>

```

Output:

Form Input PHP

Name: Name is required

Form Input PHP

Data Successfully Submitted

Name:

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)

Upon submitting the form, the PHP script embedded in the same file will check if the form has been submitted using the `$_SERVER["REQUEST_METHOD"]` variable.

If the form has been submitted via POST method, it will check if the "name" field is empty. If it is empty, an error message "Name is required" will be displayed.

If the "name" field is not empty, the submitted name will be stored in the variable `$name`, and the message "Data Successfully Submitted" will be echoed.

The form will be displayed again with the previously submitted name pre-filled in the input field, and any error message related to the name field will be displayed.

Practical Section 4. HTML Injection

Code:

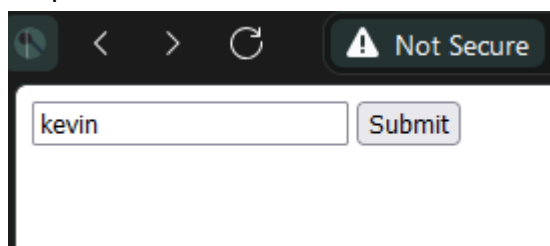
```
<?php

$input = $_POST['input'];
$input = htmlspecialchars(string: $input, flags: ENT_QUOTES, encoding: 'UTF-8');

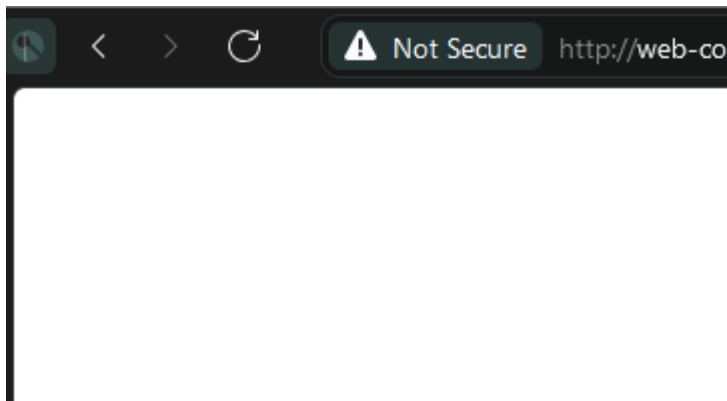
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form action="html_safe.php" method="post">
        <input type="text" name="input" id="input">

        <input type="submit" value="Submit" name="submit">
    </form>
</body>
</html>
```

Output:



The screenshot shows a web browser window with a dark theme. The address bar displays a warning icon and the text "Not Secure". Below the address bar, there is a text input field containing the name "kevin" and a "Submit" button next to it.

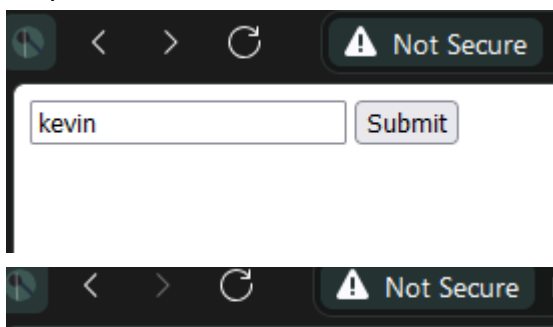


Modification 1

Code:

```
<?php  
  
$input = $_POST['input'];
```

Output:



Record here what you observed, give your explanation.

(Question No. 7)

\$_POST is a PHP superglobal variable that is used to collect form data after submitting an HTML form with the method set to POST.

In this code, \$_POST['input'] is used to retrieve the value of the 'input' field from the form that was submitted using the POST method.

The retrieved value is then stored in the variable input for further processing or usage in the PHP script.

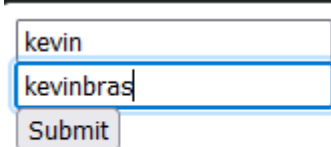
Modification 1:

Code:

```
if (filter_var(value: $email, filter: FILTER_VALIDATE_EMAIL)) {  
    echo "Email is valid";  
} else {  
    echo "Email is not valid";  
}
```

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1.0">  
    <title>Document</title>  
</head>  
<body>  
    <form action="html_safe.php" method="post">  
        <input type="text" name="input" id="input"><br>  
        <input type="email" name="email" id="email"><br>  
        <input type="submit" value="Submit" name="submit">  
    </form>  
</body>  
</html>
```

Output:



A screenshot of a web form. It contains two input fields. The first field has the text "kevin" entered. The second field has the text "kevinbras" entered. Below the second field is a "Submit" button.



Note here what you observe from the addition of the program code above.
(Question No. 8)

The code now includes validation for the email address provided in the form input.

By using `filter_var()` with the `FILTER_VALIDATE_EMAIL` filter, the code checks if the email address is in a valid format.

If the email address is valid, it outputs "Email is valid"; otherwise, it outputs "Email is not valid".

Practical Section 5: The Use of Regex in PHP

Code:

```
<?php

$pattern = '/[a-z]/';
$text = 'This is a Sample Text.';

if (preg_match(pattern: $pattern, subject: $text)) {
    echo 'Small letter found.';
} else {
    echo 'Small letter not found.';
}
```

Output:

A screenshot of a web browser window showing the output of the PHP code. The text "Small letter found." is displayed in a blue font on a white background.

Note here what you observe from the addition of the program code above.
(Question No. 9)

The code snippet uses a regular expression pattern `[a-z]` to check if any lowercase letters are present in the text variable. If a lowercase letter is found, it outputs "Small letter found."; otherwise, it outputs "Small letter not found.".

Modification 1:

Code:

```

<?php

$pattern = '/[a-z]/';
$text = 'This is a Sample Text.';

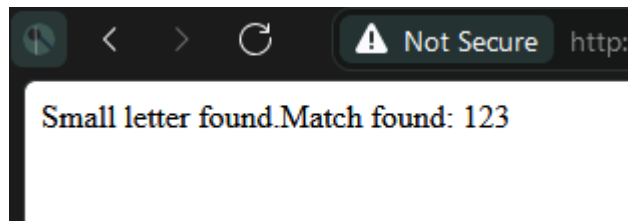
if (preg_match(pattern: $pattern, subject: $text)) {
    echo 'Small letter found.';
} else {
    echo 'Small letter not found.';
}

$pattern = '/[0-9]+/';
$text = 'There are 123 apples.';

if (preg_match(pattern: $pattern, subject: $text, matches: &$mat)) {
    echo "Match found: " . $mat[0];
} else {
    echo "Match not found.";
}

```

Output:



Note here what you observe from the addition of the program code above.
(Question No. 10)

The new code snippet extends the usage of regular expressions by introducing a pattern `[0-9]+` to match one or more digits in the text.

By using `preg_match()` with the updated pattern and text, the code determines if any digits are present in the text.

If a match is found, it outputs the matched digits using the `$mat` variable which stores the matched results.

Modification 2:

Code:

```
$pattern = '/apple/';
$replacement = 'banana';
$text = 'I like apple pie.';
$new_text= preg_filter(pattern: $pattern, replacement: $replacement, subject: $text);
echo $new_text;
```

Output:

!3I like banana pie.

Note here what you observe from the addition of the program code above.
(Question No. 11)

The addition of `preg_filter()` demonstrates how to perform pattern-based replacements in a text string using regular expressions.

The pattern `/apple/` is used to search for the word "apple" in the text.

The replacement `'banana'` is specified to replace any occurrences of `'apple'` with `'banana'`.

The `preg_filter()` function then applies this replacement operation to the text and stores the modified text in the variable `$new_text`.

Finally, the updated text is echoed to the screen to show the result of the replacement operation.

Modification 3:

Code:

```
$pattern = '/go*d/';
$text = 'god is good';

if (preg_match(pattern: $pattern, subject: $text, matches: &$matches)) {
    echo "Match found: ". $matches[0];
} else {
    echo "Match not found.";
}
```

Output:

Match found: god

Note here what you observe from the addition of the program code above.
(Question No. 12)

The new code snippet showcases the usage of a regular expression pattern `'go*d/'` to match the word `'god'` with zero or more occurrences of the letter `'o'` in between.

By applying `preg_match()` with this pattern and the text, the code determines if the specified pattern is present in the text.

If a match is found, it outputs the matched pattern using the `$matches` variable which stores the matched results.

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)

In this updated code snippet, the regular expression pattern `'go?d'` is used. This pattern will match either 'god' or 'good' in the input text 'god is good'. The '?' in the pattern makes the character 'o' optional, allowing it to match both 'god' and 'good'.

By making this change, the script will now output "Match found: good" when executed with the provided input text 'god is good'

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)

With this change, the regular expression pattern `'/[o]{1,3}'` will match the substring '[o]' repeated 1 to 3 times in the input text 'god is good'. The backslashes before the square brackets escape them, treating them as literal characters to match in the text.

When executed with the provided input text, this script will output "Match found: o" as the pattern '[o]' appears once in the text 'god is good'.

Practical Section 6 : Advanced Form

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h2>Sample Form</h2>
  <form method="POST" action="process_next.php">
    <label for="fruit">Choose a Fruit:</label>
    <select name="fruit" id="fruit">
      <option value="apple">Apple</option>
      <option value="banana">Banana</option>
      <option value="mango">Mango</option>
      <option value="orange">Orange</option>
    </select>

    <br><br>

    <label>Select Favorite Color:</label><br>
    <input type="checkbox" name="color[]" value="red"> Red<br>
    <input type="checkbox" name="color[]" value="blue"> Blue<br>
    <input type="checkbox" name="color[]" value="green"> Green<br>

    <br>

    <label>Select Gender:</label><br>
    <input type="radio" name="gender" value="Male"> Male<br>
    <input type="radio" name="gender" value="Female"> Female<br>

    <br>

    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

```

<?php

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $selectedFruit = $_POST['fruit'];

    if (isset($_POST['color'])) {
        $selectedColor = $_POST['color'];
    } else {
        $selectedColor = [];
    }

    $selectedGender = $_POST['gender'];

    echo "You selected fruit: " . $selectedFruit . "<br>";

    if (!empty($selectedColor)) {
        echo "Your favorite color(s): " . implode(separator: ", ", array: $selectedColor) . "<br>";
    } else {
        echo "You did not select any favorite color.<br>";
    }

    echo "Your gender: " . $selectedGender;
}

```

Output:

Sample Form

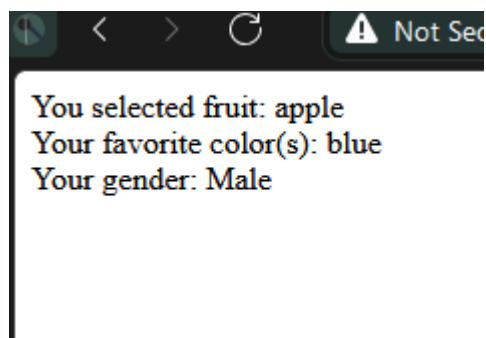
Choose a Fruit:

Select Favorite Color:

- ☐ Red
☒ Blue
☐ Green

Select Gender:

- ☒ Male
☐ Female



Note here what you observe from the program code above.
(Question No. 15)

Form Data Handling:

The script checks if the request method is POST before processing the form data. This ensures that the form data is only processed when the form is submitted. It retrieves the selected fruit, color(s), and gender from the `$_POST` superglobal array.

Conditional Handling:

It checks if the 'color' field is set in the form data. If set, it assigns the selected color(s) to the `$selectedColor` variable; otherwise, it initializes an empty array.

The script then checks if any color(s) were selected. If colors are selected, it outputs them using `implode` to join them with a comma. If no colors are selected, it informs the user that no color was selected.

Output:

The script outputs the selected fruit, favorite color(s), and gender to the user.

Modification 1:

Code:


```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7      <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
8  </head>
9  <body>
10     <h2>Sample Form</h2>
11     <form id="myForm">
12         <label for="fruit">Choose a Fruit:</label>
13         <select name="fruit" id="fruit">
14             <option value="apple">Apple</option>
15             <option value="banana">Banana</option>
16             <option value="mango">Mango</option>
17             <option value="orange">Orange</option>
18         </select>
19
20         <br>
21
22         <label>Select Favorite Color:</label><br>
23         <input type="checkbox" name="color[]" value="red"> Red<br>
24         <input type="checkbox" name="color[]" value="blue"> Blue<br>
25         <input type="checkbox" name="color[]" value="green"> Green<br>
26
27         <br>
28
29         <label>Select Gender:</label><br>
30         <input type="radio" name="gender" value="Male"> Male<br>
31         <input type="radio" name="gender" value="Female"> Female<br>
32
33         <br>
34
35         <input type="submit" value="Submit">
36     </form>
37
38     <div id="result">
39         <!-- Result will be displayed here -->
40     </div>
41
42     <script>
43     $(document).ready(function () {
44         $("#myForm").submit(function (e) {
45             e.preventDefault(); // Prevent default form submission
46
47             // Collect form data
48             var formData = $("#myForm").serialize();
49
50             // Send data to PHP server
51             $.ajax({
52                 url: "process_next.php", // Change to the appropriate PHP file name
53                 type: "POST",
54                 data: formData,
55                 success: function (response) {
56                     // Display server response in the "result" div
57                     $("#result").html(response);
58                 }
59             });
60         });
61     });
62     </script>
63 </body>
64 </html>
```

Output:

Sample Form

Choose a Fruit: Apple ▾

Select Favorite Color:

☐ Red

☐ Blue

☒ Green

Select Gender:

☒ Male

☐ Female

Submit

Sample Form

Choose a Fruit: Apple ▾

Select Favorite Color:

☐ Red

☐ Blue

☒ Green

Select Gender:

☒ Male

☐ Female

Submit

You selected fruit: apple

Your favorite color(s): green

Your gender: Male

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)

Form Structure:

The HTML form includes fields to select a fruit, favorite color(s), and gender.
It also contains a submit button to send the form data.

AJAX Form Submission:

The JavaScript code using jQuery intercepts the form submission event and prevents the default form submission behavior using `e.preventDefault()`.

It serializes the form data into a format that can be sent via AJAX.

An AJAX POST request is made to a PHP server-side script (process_next.php) with the serialized form data.

Upon successful submission, the response from the server is displayed in the "result" div without reloading the page.

Expected Behavior:

When you open the form_ajax.php file in a browser and submit the form, the form data will be sent asynchronously to the process_next.php script.

The server-side script should process the form data and send a response back.

The response from the server will be displayed in the "result" div on the page without a full page refresh.

Practical Section 7: Form Validation

Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Form Input with Validation</h1>
  <form action="process_validation.php" method="post">
    <label for="name">Name:</label>
    <input type="text" name="name"> <br>
    <label for="email">Email:</label>
    <input type="email" name="email">
    <br>

    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

```

<?php

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST["name"];
    $email = $_POST["email"];
    $errors = array();

    if (empty($name)) {
        $errors[] = "Name is required";
    }

    if (empty($email)) {
        $errors[] = "Email is required";
    } elseif (filter_var(value: $email, filter: FILTER_VALIDATE_EMAIL)) {
        $errors[] = "Invalid email format";
    }

    if (!empty($errors)) {
        foreach($errors as $error) {
            echo $error . "<br>";
        }
    } else {
        echo "Data submitted successfully: Name: " . $name . ", Email: " . $email;
    }
}

```

Output:

Form Input with Validation

Name:

Email:

Data submitted successfully: Name: kev, Email: kevin@sus.com

Note here what you observe from the addition of the program code above.
(Question No. 17)

Form Data Validation:

The script retrieves the 'name' and 'email' fields from the \$_POST superglobal array. It initializes an empty array \$errors to store validation error messages.

Validation Rules:

It checks if the 'name' field is empty and adds a corresponding error message to the \$errors array if it is.

For the 'email' field, it checks if it is empty and adds an error message if so. Additionally, it validates the email format using FILTER_VALIDATE_EMAIL and adds an error message if the format is invalid.

Error Handling:

If there are validation errors (stored in the \$errors array), the script iterates over the errors and outputs each error message.

If there are no errors, it displays a success message showing the submitted 'name' and 'email' values.

Modification 1:

Code:

```

<body>
  <h1>Form Input with Validation</h1>
  <form id="myForm" action="process_validation.php" method="POST">
    <label for="name">Name:</label>
    <input type="text" name="name" id="name"> <br>
    <span id="name-error" style="color: red;"></span>
    <label for="email">Email:</label>
    <input type="email" name="email" id="email">
    <span id="email-error" style="color: red;"></span>
    <br>

    <input type="submit" value="Submit">
  </form>
  <script>
    $(document).submit(function(event) {
      $("#myForm").submit(function(event) {
        var name = $("#name").val();
        var email = $("#email").val();
        var valid = true;

        if (name === "") {
          $("#name-error").text("Name is required");
          valid = false;
        } else {
          $("#name-error").text("");
        }

        if (email === "") {
          $("#email-error").text("Email is required");
          valid = false;
        } else {
          $("#email-error").text("");
        }

        if(!valid) {
          event.preventDefault();
        }
      });
    });
  </script>

```

Output:

Form Input with Validation

Name:

Name is required Email:

Note here what you observe from the addition of the program code above.
(Question No. 18)

When the form is submitted, the script checks if the name and email fields are empty. If either field is empty, an error message is displayed next to the respective input field. If any of the fields are empty, the form submission is prevented by calling `event.preventDefault()`. This ensures that the form is not submitted if there are validation errors.

The error messages are displayed in red color next to the input fields when the corresponding field is empty.

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.

Code:

```

</div>
<script>
    $(document).ready(function() {
        $("#myForm").submit(function(event) {
            event.preventDefault(); // Prevent default form submission

            var name = $("#name").val();
            var email = $("#email").val();
            var valid = true;

            if (name === "") {
                $("#name-error").text("Name is required");
                valid = false;
            } else {
                $("#name-error").text("");
            }

            if (email === "") {
                $("#email-error").text("Email is required");
                valid = false;
            } else {
                $("#email-error").text("");
            }

            if (valid) {
                $.ajax({
                    type: "POST",
                    url: "process_validation.php",
                    data: $("#myForm").serialize(), // Serialize form data
                    success: function(response) {
                        // Handle the response from the server
                        $("#result").html(response);
                    },
                });
            }
        });
    });
</script>

```

Output:

Form Input with Validation

Name:

Email:

Data submitted successfully: Name: test, Email: test@sus.com

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)

Code:

```
</div><div>Form Input with Validation</div>
<form id="myForm" method="post">
  <label for="name">Name:</label>
  <input type="text" name="name" id="name">
  <span id="name-error" style="color: red;"></span><br>
  <label for="email">Email:</label>
  <input type="email" name="email" id="email">
  <span id="email-error" style="color: red;"></span><br>
  <label for="password">Password:</label>
  <input type="password" name="password" id="password">
  <span id="password-error" style="color: red;"></span><br>

  <input type="submit" value="Submit">
</form>
```

```

<script>
    $(document).ready(function() {
        $("#myForm").submit(function(event) {
            event.preventDefault(); // Prevent default form submission

            var name = $("#name").val();
            var email = $("#email").val();
            var password = $("#password").val();
            var valid = true;

            if (name === "") {
                $("#name-error").text("Name is required");
                valid = false;
            } else {
                $("#name-error").text("");
            }

            if (email === "") {
                $("#email-error").text("Email is required");
                valid = false;
            } else {
                $("#email-error").text("");
            }

            if (password === "" || password.length < 8) {
                $("#password-error").text("Password must be at least 8 characters");
                valid = false;
            } else {
                $("#password-error").text("");
            }

            if (valid) {
                $.ajax({
                    type: "POST",
                    url: "process_validation.php",
                    data: $("#myForm").serialize(), // Serialize form data
                    success: function(response) {
                        // Handle the response from the server
                        $("#result").html(response);
                    },
                });
            }
        });
    });
</script>

```

Output:

Form Input with Validation

Name:

Email:

Password: Password must be at least 8 characters

<https://github.com/justKevv/Uni-stuff/tree/main/web-courses/week7>