TASK

PHP -

Form Validation

Visit our website

Introduction

CLEANING DIRTY INPUT

You saw in the last task how to get user input using HTML forms. Often users make typos, or misinterpret field labels and thereby enter incorrect information. In this task, we'll discuss ways to pick up on bad user input and gracefully reject it to have the user re-enter their details.

SINGLE PAGE FORM HANDLING

In the previous task, we used two pages to facilitate user input: one for the HTML form and one for parsing the form data. In order to let the user know when they made an input error and give them a chance to resubmit, it's easier to use a single page for form handling. We'll facilitate this by creating a PHP script (`form.php`) and, in it, checking what type of request has been made. Note that the completed script that we're about to write is available for your perusal in the task folder.

```php
<?php
   if ($_SERVER['REQUEST_METHOD'] === 'POST'){
       // todo process data
   } else {
       // todo just show form
   }
?>
```

The `$_SERVER` superglobal holds information about the request and about the PHP server. If the request is a POST, then it means form data has been sent. If it is a GET request then it's just a normal browser request.

If the user's input data is faulty, we'll be sending back the same form with the data in it as they input, so they can easily make the appropriate changes. Let's start by making a function to create the form. Declare this above the conditional.

```php
function make_form(){
   echo '<h2>Please fill in the following details:</h2>';

   echo '<form action="form.php" method="post">';
   echo '<label for="email">Email address: </label>';
   echo '<input type="text" id="email" name="email" required>';
   echo '<br/><br/>';
```

```
    echo '<label for="age">Age: </label>';
    echo '<input type="number" id="age" name="age" required>';
    echo '<br/><br/>';

    echo '<button type="submit">Submit</button>';
    echo '</form>';
}
```

Call this function in the else-block of the request method conditional. Load the script in your browser to ensure you can GET it.

If you click “Submit” you should see an empty page. This is because our request method’s conditional is set to do nothing if data is POSTed. Let’s add some logic there now, starting with getting the appropriate variables.

```
$email = $_POST['email'];
$age = (int) $_POST['age'];
```

By default, everything in `$_POST` is a string. If you know that a value is an integer, it’s good practice to cast it as such. This helps you (and your IDE) remember which variables are of what type.

Now let’s add some error checks and a success case:

```
if (strlen($email) < 2){
    echo '<p>Email is too short!</p>';
}else if ($age < 1){
    echo '<p>You are too young!</p>';
}else{
    echo '<h1>Thank you for submitting your information.</h1>';
}
```

Run the script to confirm that bad input is being identified as expected. You may notice that only a single line of text is outputted after submission. To have the form redisplay after a bad submission, we must first alter the `make_form` function to take the email and age as parameters to display them optionally.

```
function make_form($email=null, $age=null){
    echo '<h2>Please fill in the following details:</h2>';

    echo '<form action="form.php" method="post">';
    echo '<label for="email">Email address: </label>';
    if ($email === null)
        echo '<input type="text" id="email" name="email" required>';
    else
        echo '<input type="text" id="email" name="email"' .
             ' required value="'.$email.'">';
    echo '<br/><br/>';

    echo '<label for="age">Age: </label>';
    if ($age === null)
        echo '<input type="number" id="age" name="age" required>';
    else
        echo '<input type="number" id="age" name="age"' .
             ' required value="'.$age.'">';
    echo '<br/><br/>';

    echo '<button type="submit">Submit</button>';
    echo '</form>';
}
```

When you place an equals sign after a parameter in a function's definition (like with `make_form($email=null)`), you are specifying that parameter's default value. I.e. if you call `make_form()` without any arguments, it will assume `$email` is `null`. This allows us to use `make_form()` to make a blank form and, in a different place, use it with arguments to make a filled out form.

Let's do that now by adding the appropriate calls in the form validation logic.

```
if (strlen($email) < 2){
    echo '<p>Email is too short!</p>';
    make_form($email, $age);
}else if ($age < 1){
    echo '<p>You are too young!</p>';
    make_form($email, $age);
}else{
    echo '<h1>Thank you for submitting your information.</h1>';
}
```

Run it in your browser to make sure filled-in forms are being rendered when you send bad input. The net effect should be like an error message popping up above the form.

ARBITRARY TEXT FIELD VALIDATION

You should be able to use your programming logic knowledge built up so far together with your knowledge of PHP syntax to create arbitrary string checking functions. E.g. validating passwords, checking that a variable is one of a certain set (e.g. male/female/other), and checking that an email address contains an “@” symbol in the right place.

Here are some tips.

Check that a string contains a certain character

```
strpos("haystack", "y"); // 2
strpos("haystack", "x"); // false
strpos("haystack", "y") !== false; // true
strpos("haystack", "x") !== false; // false
```

Iterate through a string

```
for($i = 0; $i < strlen($str); $i++){
   echo $str[$i];
}
```

Check that a character is a lowercase letter

```
ord($char) >= ord('a') && ord($char) <= ord('z')
```

`ord` is a function that gets the ASCII code for a character. ASCII is one of the world’s oldest and most widely-used text encodings. It has the wonderful feature of having letters adjacent in the alphabet also adjacent in their ordinal codes. So if a letter’s ASCII code is x, then the ASCII character represented by x+1 will be the next letter in the alphabet. This helps a lot with string checking. ASCII table available **here**.

Check that a variable is one of a set

```
$options = ['a', 'b', 'c'];
in_array('x', $options); // false
in_array('b', $options); // true
```

You can, of course, also string together a bunch of equality comparisons, but this gets impractical after about three options.

```
if($var==='a' || $var==='b' || $var==='c'){
```

If you want to see what else is available, you can find a list of built-in PHP string functions **here**.

Compulsory Task 1

Create a multi-page form-handling PHP script that parses an HTML form with the following user data and constraints. The validator should consist of two files: an HTML file with the form and a PHP file with the validation code.

- Email (valid email address)
- Username (6-10 characters; alphanumeric only)
- Password (at least one lowercase letter, one uppercase letter, and one number)
- Date of birth (after 1900, and before 2020)
- Gender (either "male", "female", or "other")
- Address (multiple lines)

Your script should display an appropriate error message if the user enters bad input. It should not re-render the form. You may use a combination of HTML form constraints (mentioned in the previous task) and PHP form validation.

Compulsory Task 2

- Extend Compulsory Task 1 to be a single page form-handling script.
- If the user enters bad input, your script should display an error message *and* repopulate the form so that the user can change their entered data.
- Note that the Password field should be exempt from repopulation. It should always be blank on every form render. This is standard practice in web development and ensures users that their passwords aren't being stored unnecessarily.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we’ve done a good job?

Click here to share your thoughts anonymously.