Learn PHP

Bobby Tables

Collecting input from users is necessary, but nefarious users can use this as an opportunity to mess with things.

Just like the parents of [Bobby Tables](https://xkcd.com/327/).

We’ve made a form for a user to create a profile on our site. We want to collect the user’s name, character, email and birth year. Our users have been having a field day with this and we need to clean up our form.

Let’s begin!

**Tasks**

14/15 Complete

Mark the tasks as complete by checking them off

**What's in a name?**

1.

Try submitting some HTML in the name input box, like <h1>HELLO</h1>.

Can you see how this could cause problems if another user was browsing this profile?

The HTML is being interpreted and displayed. A nefarious user could use this to change behavior for others viewing their profile.

2.

We want to allow users to use HTML characters in their names, but we don’t want to interpret them.

Where we are assigning $name to the input value, use a built-in function to transform any HTML characters in the input into HTML entities.

$name = htmlspecialchars($\_POST["name"]);

3.

Let’s also remove any leading or trailing whitespace from the name.

Use a built-in function for this.

$name = trim(htmlspecialchars($\_POST["name"]));

4.

We also want to prevent people impersonating other users. To do this, we need to check the name from the input against the contents of the array $existing\_users.

Instead of assigning to the variable $name directly, assign the transformed value of the input to $raw\_name.

$raw\_name = trim(htmlspecialchars($\_POST["name"]));

5.

Check if $raw\_name is in $existing\_users, using the [built-in function in\_array](https://www.php.net/manual/en/function.in-array.php).

If it is, concatenate "This name is taken. <br>"; to $validation\_error.

If it is a new name, assign $raw\_name to $name.

$raw\_name = trim(htmlspecialchars($\_POST["name"]));  
if (in\_array($raw\_name, $existing\_users)) {  
  $validation\_error .= "This name is taken. <br>";  
} else {  
  $name = $raw\_name;  
}

**Strange characters**

6.

We intend for users to be a wizard, mage, or orc. Even though the character is a radio button selection, users can set themselves to something besides these options!

Try this yourself by inspecting one of the options from the embedded browser (in Chrome, right click on one of the options, like “Wizard”, and inspect).

You can change the “value” for the option to a different string, like "puppy". Select that option and submit the form to see what happens.

Remember, the front-end code can be edited by the user. We need to validate their submission on the back-end.

7.

Instead of assigning the character input to $character, save it to a new variable $raw\_character so we can validate it.

$raw\_character = $\_POST["character"];

8.

Use the [built-in function in\_array](https://www.php.net/manual/en/function.in-array.php) to check if $raw\_character is in the array ["wizard", "mage", "orc"].

If it is, assign $raw\_character to $character.

If it isn’t, concatenate "You must pick a wizard, mage, or orc. <br>" to $validation\_error.

$raw\_character = $\_POST["character"];  
if (in\_array($raw\_character, ["wizard", "mage", "orc"])) {  
  $character = $raw\_character;  
} else {  
  $validation\_error .= "You must pick a wizard, mage, or orc. <br>";  
}

**Emails that aren't emails**

9.

We want to prevent users from entering email addresses that aren’t…. well… email addresses.

Begin by assigning the email input to a new variable $raw\_email, so we can validate it.

$raw\_email = $\_POST["email"];

10.

Use a built-in PHP function to check if $raw\_email is an email address.

If it is, assign $raw\_email to $email.

If it isn’t, concatenate "Invalid email. <br>" to $validation\_error.

if (filter\_var($raw\_email, FILTER\_VALIDATE\_EMAIL)){  
  $email = $raw\_email;  
} else {  
  $validation\_error .= "Invalid email. <br>";  
}

**How old???**

11.

We’re using the user’s birth year to display an age in the user’s profile. Try entering a birth year way in the future or past and see what happens.

By submitting invalid birth years, we can generate negative or very unrealistic ages.

12.

Instead of assigning the birth year input to $birth\_year directly, assign it to a new variable $raw\_birth\_year.

$raw\_birth\_year = $\_POST["birth\_year"];

13.

We’re going to use filter\_var with some options to limit the range of years that users can submit.

Create an $options variable to set the minimum and maximum values to realistic years.

You can get the current year using date("Y").

$options = ["options" => ["min\_range" => 1900, "max\_range" => date("Y")]];

14.

Add a check to ensure that $raw\_birth\_year is an integer in the range defined by $options.

If it is, assign $raw\_birth\_year to $birth\_year.

If it isn’t, concatenate "That can't be your birth year. <br>" to $validation\_error.

if (filter\_var($raw\_birth\_year, FILTER\_VALIDATE\_INT, $options)) {  
  $birth\_year = $raw\_birth\_year;  
} else {  
  $validation\_error .= "That can't be your birth year. <br>";  
}

**Great!**

15.

Congratulations, you should now have a form that is mostly impervious to ill intentioned users!

If you’d like some more practice, consider implementing the following:

* Mages have to be born on leap years
* Names must all start with lowercase characters
* Emails must end in .com