Web forms

What is a form ??

A form is actually an HTML tag containing graphic elements, such as text boxes, choice of options, checkboxes, and so on.
When you sign in to a website or your mailbox, you use a form.
The forms are used to obtain the user's data and send it to the web server for processing.
Here is a simple example with 1 text box (nickname) and a button:

```
      <! DOCTYPE html>
<html>
<head>
   <title> My first form </ title>
</ head>
<body>

   <form name = "form1 "method =" "action =" ">
            <input type =" text "value =" nickname ">
            <input type =" submit "name =" submit1 "
    value =" login ">

   </ form>

</ body>
< / html>
```

In our example, if a user comes to our site and needs to log in, we need to get the data from the text boxes. First, let's look at the HTML METHOD, ACTION, and SUBMIT attributes.

The METHOD attribute

Let's take the form from the previous example.
If we look at the first line of our form, we see a METHOD attribute:

```
<form name = "form1" method = "" action = "">
```

- GET Method

The Method attribute is used to tell the browser how the form information should be sent. The two most common methods you can use are GET and POST.

In our example, our method is empty. Change the code:
<form name = "form1" method = "GET" action = "">

Try to validate the form by pressing the button.
What do you notice ?

What you should notice is in the address bar of the browser. After the name of our 'forms.php' file, we have this
? Submit1 = Login

This is due to the use of the GET method. The form data is found in the address bar. You will see a question mark, followed by the form data. 'submit1' is the name of the button and 'login' is the text value of the button. This is what is returned by the GET method.

You use the GET method when the data you want to return is not crucial information that needs to be protected.

- POST method

In place of the GET method, we can use the POST method.
So change the code with POST:

<form name = "form1" method = "POST" action = "">

Leave the page and reopen it in your browser. Validate the form by pressing the button.

What do you notice ?

The '*? Submit1 = Login*' part has now disappeared! This is because we used POST as a method. The use of POST means that the data of the form will not be added to the address in the address bar at the sight of all.

In our course, we will use both POST and GET but beware: if the data is not sensitive, use GET, otherwise use POST.

Example of use:

- A good example of using POST is the submission of connection information to the server.
- GET is ideal for search engines because it allows users to 'mark' (favorites) the results.

The ACTION attribute The ACTION

attribute means: "Where do you want the form to be sent?".

You can send form data to another PHP script, the same PHP script, an email address, or any other form of script.
If you do not specify it, your form will not be sent anywhere.

In PHP, a popular technique is to send the data from the form to the same page as the form, that is, to send it to itself.

At first, we will use this technique but we will see both techniques in action.

So modify your code:

<form name = "form1" method = "GET" action = "forms.php">

'*forms.php*' is the name of my PHP file.
If you refresh your page, you will not notice any difference. No mistake but nothing special either.

Now that an ACTION has been defined. We can 'submit' it (SUBMIT).

/! \ The METHOD attribute tells you how the form data is sent and the ACTION attribute tells you where it is sent. /! \

The SUBMIT button

The HTML Submit button is used to submit form data to the script mentioned in the ACTION attribute.

Example:

<input type = "submit" name = "submit1" value = "login">

You do not need to do anything special with a 'Submit button' - the data is submitted in secret.
As long as SUBMIT has a defined ACTION, your data will be sent somewhere. But the NAME attribute of the submit buttons is very useful. You can use this name to check if the form was actually sent or if the user simply refreshed the page. This is important when the PHP script is on the same page as the HTML form.

Our 'Submit button' is called 'Submit1', but you can call it as you wish.

Retrieve and Process Data

Now that we've seen the METHOD, ACTION, and SUBMIT attributes, we'll see how to retrieve and process the data that the user enters.

To get the text entered by a user in a text box, it requires a NAME attribute. You then tell PHP the name of the text box you want to work with.

Our text box does not have a NAME yet, so change your HTML code to:

```
<input type = "text" name = "nickname" value = "nickname">
```

Then put this piece of PHP code in your script (before the form!):

```
<? php
    $ nick = $ _POST ['nickname'];
        echo $ username;
    ?>
```

Reload your page and try to validate the form.
Then try changing the nickname and validate again. The new value of your text should appear.
On the other hand, the textbox text itself returns to '*pseudo*' each time. This is because the text box is reset when the data is returned to the browser.

Let's go back to the syntax of the code we just added:

```
$ _ POST [ 'username'];
```

We start with the dollar sign ($), an underscore (_) and then come the METHOD used: POST or GET. Finally, we open the square brackets and we type the NAME of our HTML element.
In our case the NAME is 'pseudo'.

What is returned is the value of our HTML element.
We can then assign it to a variable:

```
$ nickname = $ _POST ['nickname'];
```

PHP will fetch an HTML form element with the NAME 'nickname'. It then looks at the VALUE attribute for this form element. It returns this value to use and manipulate.

For now, we're just showing what the user is typing.

But we can, for example, check the username.
Change your code:

```
<? Php
    $ nick = $ _POST ['nickname'];
            if ($ username == 'simon') {
      echo "Welcome, my friend";
   } else {
      echo "You're not a member of this website";
   }
    ?>
```

Refresh your page and check your code.
The first time you arrive on the page, the message "*You're not a member of this website*" is displayed. The reason the text appears when you first load the page is that the script runs, whether the user clicks the button or not. This is the problem that you encounter when a PHP script is on the same page as the HTML and is subject to itself in the ACTION attribute.

Check if the 'submit' button has been clicked

To check if a 'SUBMIT' button has been clicked, use this little piece of code:

```
if (isset ($ _POST ['Submit1'])) {}
```

A little barbaric but if we break down, we only have 3 elements: the IF, isset (), and $ _POST ['submit1'].
The isset function checks whether a variable has been defined or not.

Try this piece of code:

```
<? Php

 if (isset ($ _ POST ['pseudo'])) {

    $ nick = $ _POST ['nickname'];

    if ($ username == 'simon') {
        echo "Welcome, my friend";
    } else {
        echo "You're not a member of this website";
    }

 }

 ?>
```

Keep the data entered by the user

Our problem is that each time we click on the button, the value of our textbox is reset (thanks to the VALUE by default).

To return the data in the form and thus preserve the data already entered by the user, you can use this:

```
value = "<? php echo $ username;?>"
```

Now the VALUE attribute is PHP code!

On the other hand a problem arises: during the first visit on your page, no name was typed by the user. For that, you can modify your PHP code to add a default value:

```
<? Php

    if (isset ($ _ POST ['pseudo'])) {

        $ nickname = $ _POST ['nickname'];

        if ($ username == 'simon') {
            echo "Welcome, my friend";
        } else {
            echo "You're not a member of this website";
        }

    } else {
        $ username = "username";
    }

?>
```

The ACTION attribute on another page

For the moment, the data was sent on the same page as the form itself. We will now see how to submit the form data on another page.

To do this create a 'submitForm.php' file and copy the PHP code from the previous example in:

```
<? Php
if (isset ($ _ POST ['nickname'])) {

    $ nick = $ _POST ['nickname' ];

    if ($ username == 'simon') {
        echo "Welcome, my friend";
    } else {
        echo "You're not a member of this website";
    }

} else {
    $ username = "username";
}
?>
```

So, your initial file 'forms.php' should look like this:

```
<! DOCTYPE html>
<html>
<head>
    <title> My first form </ title>
</ head>
<body>
    <form name = "form1" method = "POST" action =
"submitForm.php">

        <input type = "text" name = "nickname" value = "nickname">
        <input type = "submit" name = "submit1" value = "login">

    </ form>

</ body>
</ html>
```

Sending form data in a different PHP script is a way to keep the HTML and PHP separate. However, you may have noticed that the script is running on a new page.