Lecture / Instructor: Leigh Cotnoir

PHP : GETTING INFO FROM USERS (A)

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

- URLs and links

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

- URLs and links
- Forms with user input

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

- URLs and links
- Forms with user input
- Cookies

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

- URLs and links
- Forms with user input
- Cookies
- Sessions

How We Get Info From Web Users

Before digging into how to build pages, let's look at how we get information from users.

- URLs and links : $_GET
- Forms with user input : $_POST
- Cookies : $_COOKIE
- Sessions : $_SESSION

*These are some of the predefined PHP "superglobal" arrays.

How We Get Info From Web Users

In this lecture (part A), we will only address $_GET and $_POST.

- URLs and links : $_GET
- Forms with user input : $_POST

How We Get Info From Web Users

Let's start with this one:

- URLs and links : $_GET

Using the $_GET Superglobal

- URLs and links : $_GET

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4  <meta http-equiv="Content-Type" content="text/html;
   charset=UTF-8">
5  <title>Page 1</title>
6  </head>
7
8  <body>
9
10 <a href="get_page2.php">Go to Page 2</a>
11
12 </body>
13 </html>
```

First, let's start by making new file called "get_page_1.php" and in it, create a simple link to another page.

Here we have a normal link that goes to "get_page2.php".

Using the $_GET Superglobal

- URLs and links : $_GET

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4  <meta http-equiv="Content-Type" content="text/html;
   charset=UTF-8">
5  <title>Page 1</title>
6  </head>
7
8  <body>
9
10 <a href="get_page2.php?name=test_value">Go to Page 2</a>
11
12 </body>
13 </html>
```

Now let's add a parameter to pass along with the page when the link is clicked.

The "?" after a link's filename is a query mark. Everything after the query is passed into the "$_GET" superglobal array to provide information to the target page.

Using the $_GET Superglobal

- URLs and links : $_GET

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4  <meta http-equiv="Content-Type" content="text/html;
   charset=UTF-8">
5  <title>Page 1</title>
6  </head>
7
8  <body>
9
10 <a href="get_page2.php?name=test_value">Go to Page 2</a>
11
12 </body>
13 </html>
```

Because $_GET behaves as an associative array, we pass both keys and values. In this example, “name” is the key, and “test_value” is its current value.

In the next slide, we’ll see how to access that info in “get_page2.php”.

Using the $_GET Superglobal

- URLs and links : $_GET

To access the parameter passed from page one, the second page must echo the value of the "name" key (see right image).

get_page1.php

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Page 1</title>
</head>

<body>

<a href="get_page2.php?name=test_value">Go to Page 2</a>

</body>
</html>
```

get_page2.php

```
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Page 2</title>
</head>

<body>

<?php
    echo $_GET['name'];
?>

</body>
</html>
```

Using the $_GET Superglobal

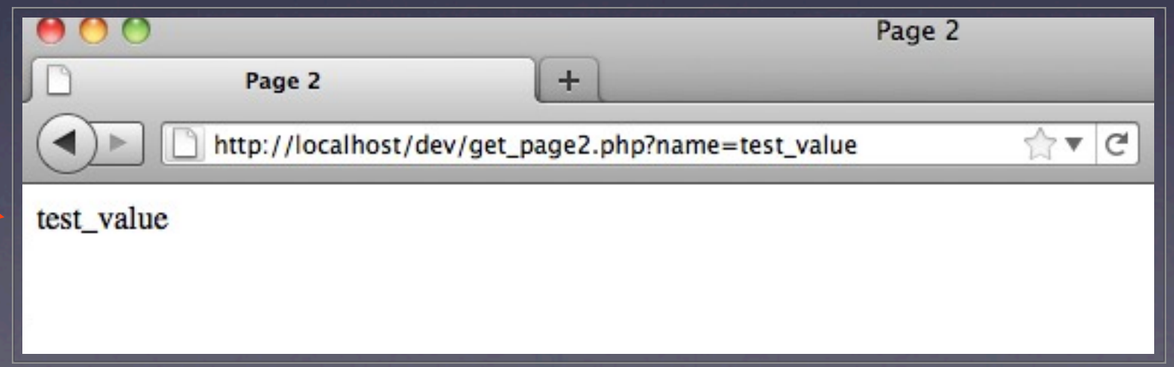
- URLs and links : $_GET

Here's the result when you click on the link in "get_page1.php". Look closely at the URL in the picture on the right to see how the parameters explicitly show up in the address bar. This is why the $_GET method is not great for information requiring security, like passwords and private information. It's too revealing without encryption.

source code from "get_page2.php"

```
10  <?php
11      echo $_GET['name'];
12  ?>
```

browser output when coming from "get_page1.php"



Using the $_GET Superglobal

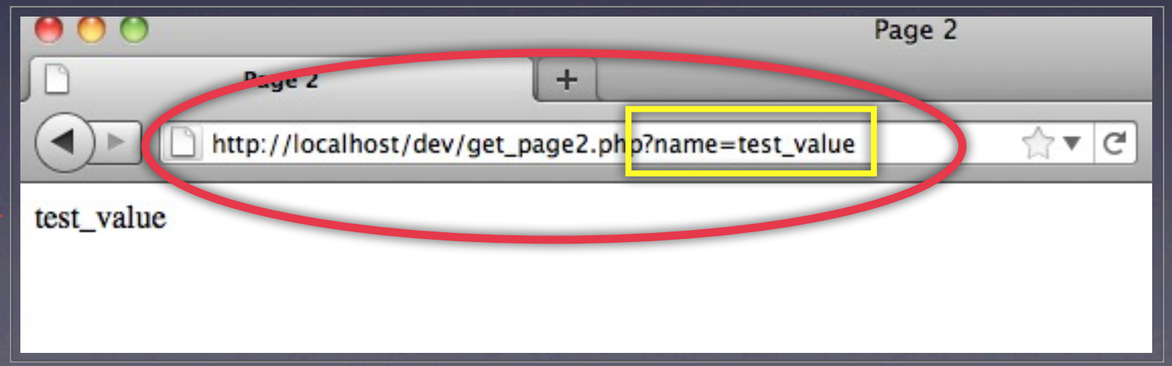
- URLs and links : $_GET

Here's the result when you click on the link in "get_page1.php". Look closely at the URL in the picture on the right to see how the parameters explicitly show up in the address bar. This is why the $_GET method is not great for information requiring security, like passwords and private information. It's too revealing without encryption.

source code from "get_page2.php"

```
10  <?php
11      echo $_GET['name'];
12  ?>
```

browser output when coming from "get_page1.php"



Using the $_GET Superglobal

- URLs and links : $_GET

Next, we need to look at what happens when we try to pass more than one parameter.

Using the $_GET Superglobal

URLs and links : $_GET

Next, we need to look at what happens when we try to pass more than one parameter.

Modified link on first page....Notice that we separate parameters with an ampersand "&".

```
9
10  <a href="get_page2.php?name=apple&type=fruit">Go to Page
     2</a>
11
```

Modified code on target page....We access the values by calling for the key names in the $_GET array.

```
9
10  <?php
11      echo $_GET['name']." is a type of ".$_GET['type'];
12  ?>
13
```

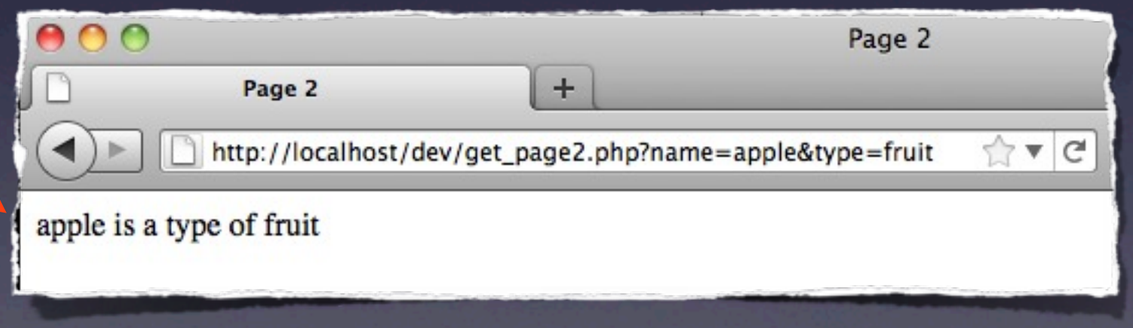
Using the $_GET Superglobal

- URLs and links : $_GET

Next, we need to look at what happens when we try to pass more than one parameter.

The code below on target page results in the output on the right.

```php
<?php
    echo $_GET['name']." is a type of ".$_GET['type'];
?>
```



Here's the resulting browser output (above).

Using the $_GET Superglobal

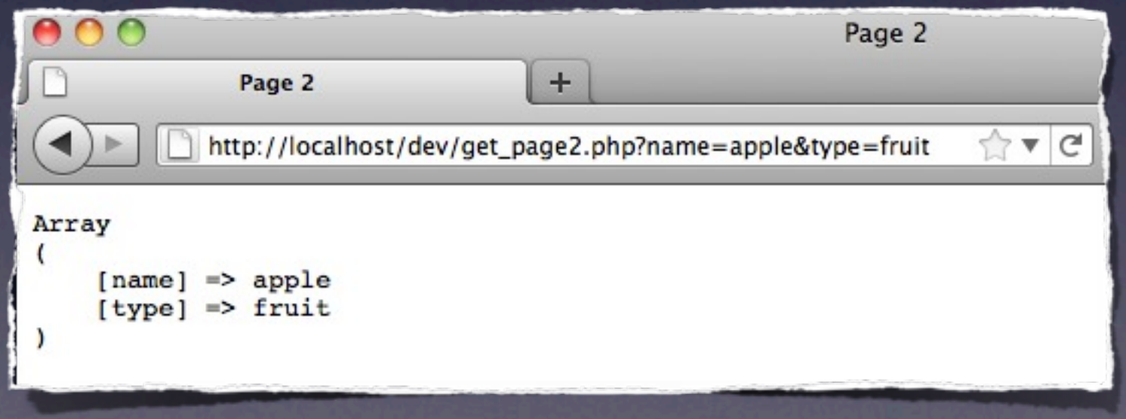
- URLs and links : $_GET

Next, we need to look at what happens when we try to pass more than one parameter.

We can also change the code on target page (below) so that we inspect the $_GET array.

```
<pre>
<?php
    print_r($_GET);
?>
</pre>
```

Below are the results of the browser output:



Using the $_GET Superglobal

- URLs and links : $_GET

So, you may have noticed that we are already using some special characters, or "metacharacters," as operational devices in the URL to pass parameters . . . like "?" for querying and "&" as a parameter separator.

What happens if we want to include special characters inside of the array keys or values? This is where we have to use special php functions for encoding.

Using the $_GET Superglobal

- URLs and links : $_GET

When creating a link, there are three types of encoding you want to specify for three separate parts of the link. See below:

```
<a href="page.php?id=something">CLICK HERE</a>
                  1                2
```

1 : urlencode()

2 : htmlspecialchars()

Using the $_GET Superglobal

- URLs and link encoding for $_GET

First, let's set up some variables inside a php block to build a link.

```
<?php
 $host = 'http://localhost';
 $path = '/dynamically/generated/page.php';
 $value1 = 'Garman & Sons';
 $value2 = 'Best deals!';
 $link = 'See Garman & Sons "Best Deals"';
 // Set up the url:
 $url = $host;
 $url .=$path;
 $url .= "?param1=" . urlencode($value1);
 $url .= "&param2=" . urlencode($value2);
?>
```

Here's how we would finish writing it in the HTML:

```
<a href="<?php echo $url; ?>">
  <?php echo htmlspecialchars($link); ?>
</a>
```

Using the $_GET Superglobal

- URLs and link encoding for $_GET

To be extra-safe, here's how we *should* ultimately write it in the HTML:

```
<a href="<?php echo htmlspecialchars($url); ?>">
   <?php echo htmlspecialchars($link); ?>
</a>
```

This protects how the link will render out if it contains special characters as well (a security measure).

How We Get Info From Web Users

Next, let's look at this method:

- URLs and links : $_GET
- Forms with user input : $_POST

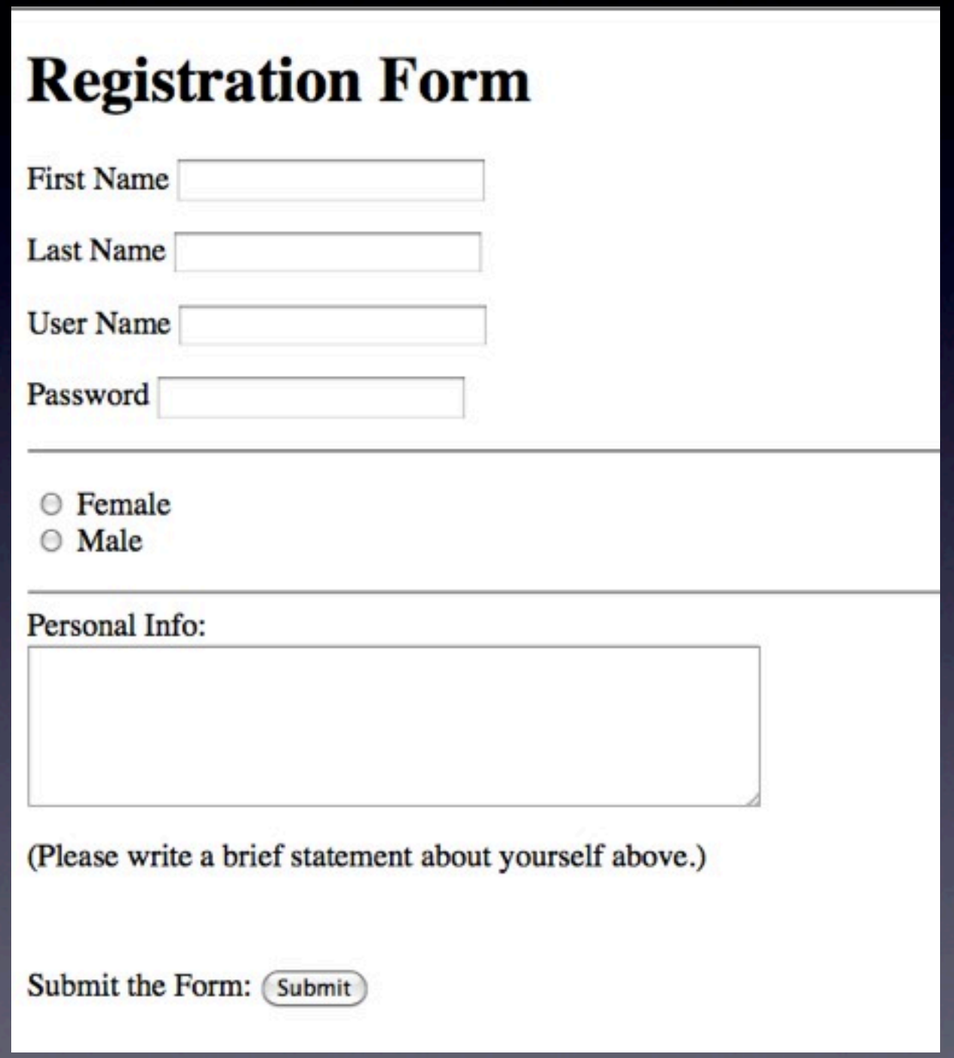
How We Get Info From Web Users

Next, let's look at this method:

- Forms with user input : $_POST

First, let's start by making a page with a form using a few basic field types. The field types we will employ are:

- Text Input
- Radio Button Input
- Textarea Input



How We Get Info From Web Users

- Forms with user input : $_POST

We must first create a new php file. Let's call it "form_register.php". Inside the body of this new file, create a basic, empty form with the following HTML attributes and values:

- action="form_register_do.php"
- method="post"

```
 8  <body>
 9  <h1>Registration Form</h1>
10  <form action="form_register_do.php" method="post" ></form>
11
12
```

How We Get Info From Web Users

- Forms with user input : $_POST

Here's what each of those attributes do for us:

- "action" will execute a php script page that will utilize the superglobal array's values
- "method" will tell PHP in which superglobal array type to store the data

```
 8  <body>
 9  <h1>Registration Form</h1>
10  <form action="form_register_do.php" method="post" ></form>
11
12
```

How We Get Info From Web Users

- Forms with user input : $_POST

  Next, inside of the <form></form> tags, create your first **input** field. The first field will be a **text** field type for the user's first name, which allows for the user to type information into the field on screen.

  Here's how it will look at first:

```
11    <p>
12      <input type="text" name="f_name" id="f_name" accesskey="1" tabindex="1">
13    </p>
```

How We Get Info From Web Users

- Forms with user input : $_POST

Here's the anatomy of this HTML <input> element:

input . . . is the HTML tag name
type . . . is the attribute determining what kind of input device to display
name . . . is the attribute defining this input's $_POST array key name
id . . . is for CSS styles as well as for the label tag we have not yet applied
accesskey . . . is the ADA-accessible hot-key access attribute for this field
tabindex . . . is the ADA and user-friendly tab key order within the form

```
11    <p>
12      <input type="text" name="f_name" id="f_name" accesskey="1" tabindex="1">
13    </p>
```

How We Get Info From Web Users

- Forms with user input : $_POST

  In order for ADA-accessible screen reader devices to clearly identify the purpose of each field, valid markup requires that we also use a label tag with a “for” attribute, clearly identifying which input tag ID the label is describing:

```
11    <p>
12      <label for="f_name">First Name</label>
13      <input type="text" name="f_name" id="f_name" accesskey="1" tabindex="1">
14    </p>
```

How We Get Info From Web Users

- Forms with user input : $_POST

  Something important to notice about this particular text input type is that you don't see a "value" attribute that defines the $_POST value for the "name" key. The "value" attribute is only used on input types that have predefined value choices (as with radio buttons). The text input type, however, is user-customizable, so the "value" that the $_POST array will grab for the name key below is what the user types into the field.

```
11    <p>
12      <label for="f_name">First Name</label>
13      <input type="text" name="f_name" id="f_name" accesskey="1" tabindex="1">
14    </p>
```

How We Get Info From Web Users

- Forms with user input : $_POST

  Alternately, though, you can use an empty "value" attribute in this type of input tag if it seems easier to understand.

```
12      <label for="f_name">First Name</label>
13      <input type="text" name="f_name" id="f_name" accesskey="1" tabindex="1" value="">
```

How We Get Info From Web Users

- Forms with user input : $_POST

Next, let's create three more text type input fields for Last Name, User Name, and Password. Use the following attributes for these fields:

Label text	type	name	id	accesskey	tabindex
Last Name	text	l_name	l_name	2	2
User Name	text	u_name	u_name	3	3
Password	password	password	password	4	4

NOTE: the "password" type masks a password on the user's screen and prevents the browser's attempt at storing the entry in the browser's auto-completion cache.

How We Get Info From Web Users

- Forms with user input : $_POST

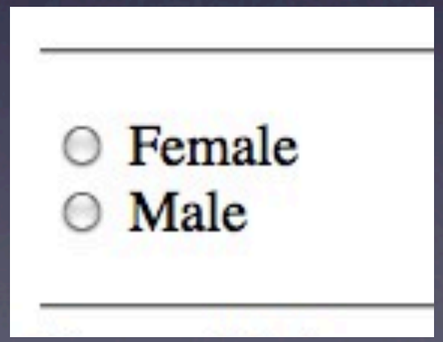
  The section inside the form should now look something like:

```
11    <p>
12      <label for="f_name">First Name</label>
13      <input type="text" name="f_name" id="f_name" accesskey="1" tabindex="1">
14    </p>
15    <p>
16      <label for="l_name">Last Name</label>
17      <input type="text" name="l_name" id="l_name" accesskey="2" tabindex="2">
18    </p>
19    <p>
20      <label for="u_name">User Name</label>
21      <input type="text" name="u_name" id="u_name" accesskey="3" tabindex="3">
22    </p>
23    <p>
24      <label for="password">Password</label>
25      <input type="password" name="password" id="password" accesskey="4" tabindex="4">
26    </p>
```

How We Get Info From Web Users

- Forms with user input : $_POST

  Another common input type is the **radio button**. You can have a single-choice radio item, or you can also have a "radio group." A radio group will give users a chance to select an item out of several pre-defined choices. In our example form, users are prompted to select their sex (Female or Male) using a radio button group:



How We Get Info From Web Users

- Forms with user input : $_POST

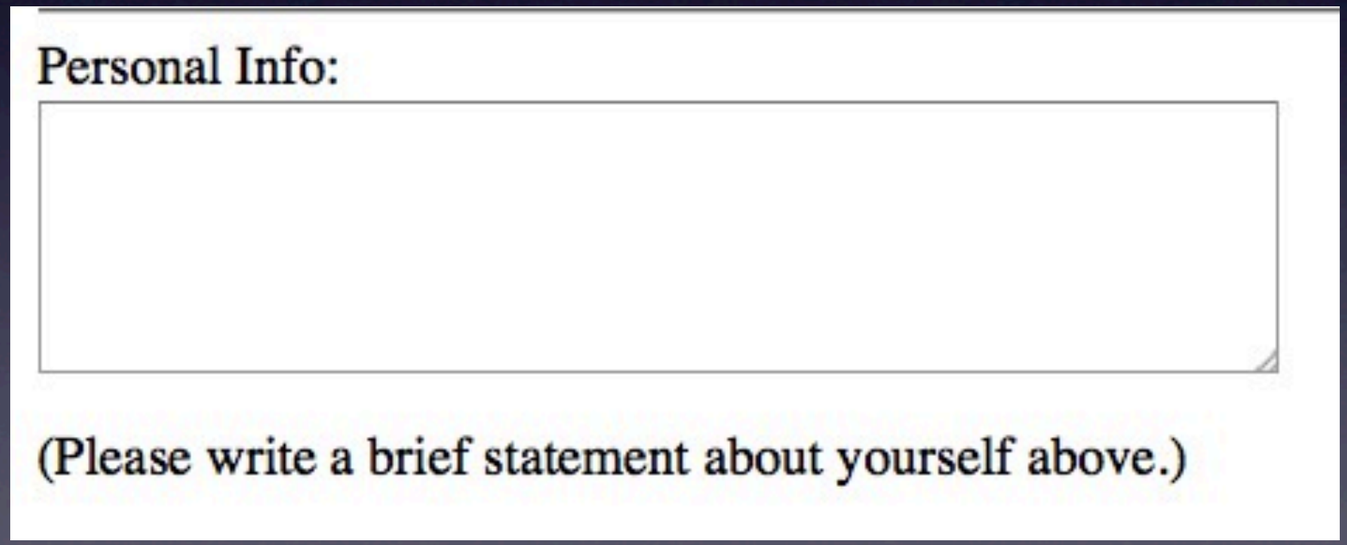
  In the code below we are defining the previous slide's Female/Male radio group choices. The way we determine that they are in the same group is by assigning multiple radio input types with the same "name". Below, both inputs have a name attribute of "sex". But you will also see that each has a separate and unique "value" attribute ('f' and 'm' respectively). Radio buttons will only allow **one** of a group to be selected, and that selection is what will go into the $_POST array.

```
29    <p>
30        <input type="radio" name="sex" value="f" id="sex_0" accesskey="5" tabindex="5">
31        <label for="sex_0">Female</label>
32      <br>
33      <input type="radio" name="sex" value="m" id="sex_1" accesskey="6" tabindex="6">
34        <label for="sex_1">Male</label>
35    </p>
```

How We Get Info From Web Users

- Forms with user input : $_POST

  Another kind of field is the <textarea> tag, which allows users to type much longer sections of text than a standard input field. In our example, we used it for a brief user bio statement:



See the next slide for the code details.

How We Get Info From Web Users

- Forms with user input : $_POST

  Notice that once again, because we are allowing users to type customized data as the value of this textarea field, it is not required that we declare a "value" attribute.

  Also notice that we have two new attributes, "cols" for columns, and "rows". These define the space that the text box is allowed to occupy on the web page. There are different ways to set up most of these field tags, but we won't go into all of the specifics here.

```
39  <label for="bio">Personal Info:</label><br />
40  <textarea name="bio" id="bio" cols="45" rows="5" accesskey="7" tabindex="7"></textarea>
41  <p>(Please write a brief statement about yourself above.)</p>
```

How We Get Info From Web Users

- **Forms with user input : $_POST**

  The last thing in the example we need to address is the Submit button. In order for the form to execute the "action" specified in the <form> tag, we must place a submit button **inside** of the form.


```
43  <p>
44    <label for="submit">Submit the Form: </label>
45    <input type="submit" name="submit" id="submit" value="Submit" accesskey="8" tabindex="8">
46  </p>
47  </form>
```

How We Get Info From Web Users

To the left is a snapshot of the entire form in HTML code view.

Okay, so now that we have our form, how does it work?

In order for the form to do anything for us, we need to create the script php page "form_register_do.php" that we referenced in the "action" attribute.

```
<form action="form_register_do.php" method="post" name="register">
  <p>
    <label for="f_name">First Name</label>
    <input type="text" name="f_name" id="f_name" accesskey="1" tabindex="1" value="">
  </p>
  <p>
    <label for="l_name">Last Name</label>
    <input type="text" name="l_name" id="l_name" accesskey="2" tabindex="2">
  </p>
  <p>
    <label for="u_name">User Name</label>
    <input type="text" name="u_name" id="u_name" accesskey="3" tabindex="3">
  </p>
  <p>
    <label for="password">Password</label>
    <input type="password" name="password" id="password" accesskey="4" tabindex="4">
  </p>
  <hr />

  <p>
      <input type="radio" name="sex" value="f" id="sex_0" accesskey="5" tabindex="5">
      <label for="sex_0">Female</label>
    <br>
    <input type="radio" name="sex" value="m" id="sex_1" accesskey="6" tabindex="6">
      <label for="sex_1">Male</label>
  </p>

<hr />

<label for="bio">Personal Info:</label><br />
<textarea name="bio" id="bio" cols="45" rows="5" accesskey="7" tabindex="7"></textarea>
<p>(Please write a brief statement about yourself above.)</p>

<p>
  <label for="submit">Submit the Form: </label>
  <input type="submit" name="submit" id="submit" value="Submit" accesskey="8" tabindex="8">
</p>
```

How We Get Info From Web Users

- **Forms with user input : $_POST**

  Create a new php file in the same directory as the "form_register.php" page. Name the new file "form_register_*do*.php".

  In order to simply see how the variables are passed from the first page with the HTML form to this new action page, let's simply write a block of php code in the <body> of the page that prints the *keys* => *values* for the $_POST array.

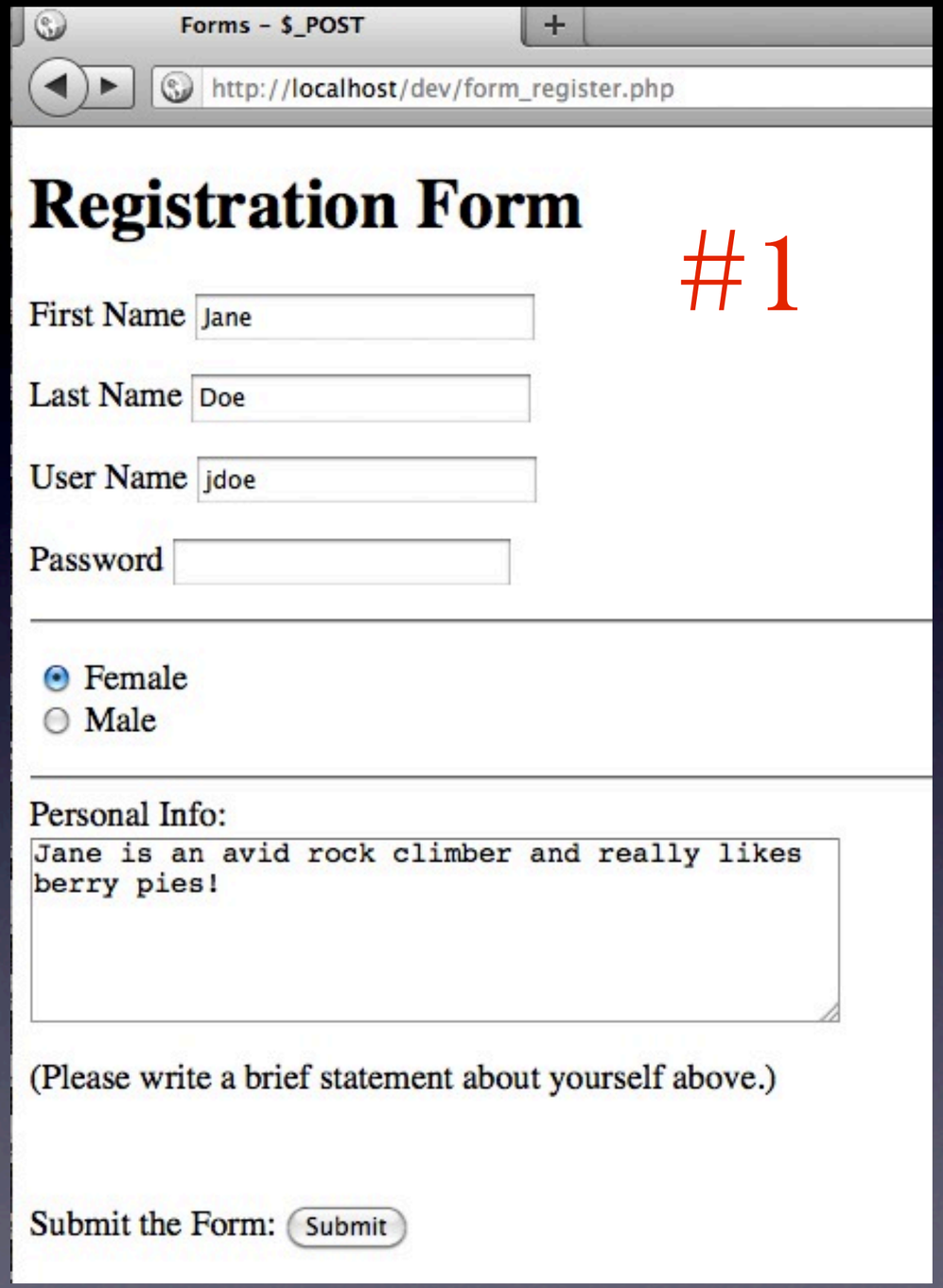
  ```
  <?php print_r($_POST); ?>
  ```

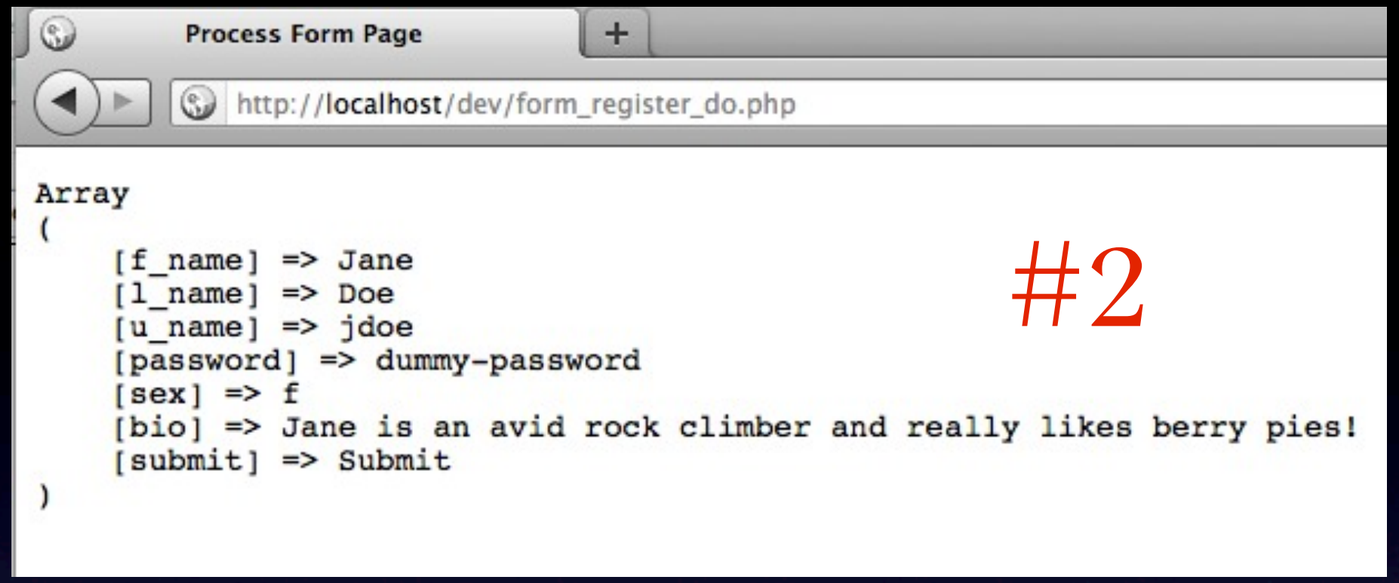
  Save this new page, and now open the original "form_register.php" page in a browser.

How We Get Info From Web Users

- Forms with user input : $_POST

  On the next slide, see how we entered info on the left, and how the processing page on the right displays the posted data using the “print_r” function.





Looking at the associative array keys and values passed to the processing page (above) from the original form page (left), you can see how the data could be manipulated and stored into variables using PHP to do a variety of different things.

Please note that this form did NOT use important form validation required to eliminate user errors.