# PHP-II

# PHP - Dealing with the Client

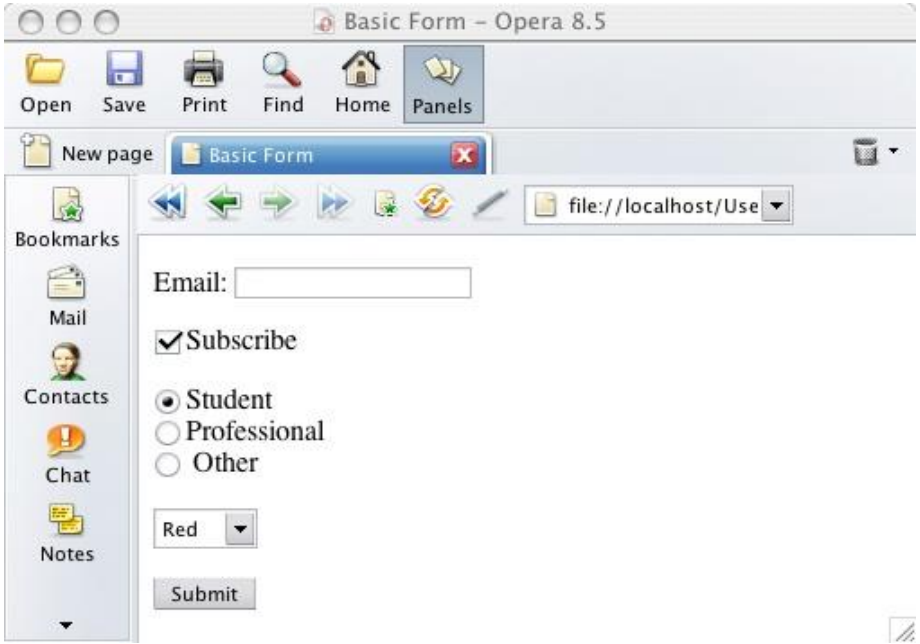
- All very nice but …
- … How is it useful in your web site?
- PHP allows you to use HTML forms
- Forms require technology at the server to process them
- PHP is a feasible and good choice for the processing of HTML forms

# PHP - Dealing with the client

- Quick re-cap on forms
- Implemented with a <form> element in HTML
- Contains other input, text area, list controls and options
- Has some method of submitting

# PHP - Dealing with the client

- Text fields
- Checkbox
- Radio button
- List boxes
- Hidden form fields
- Password box
- Submit and reset buttons



# PHP - Dealing with the client

- <form method="post" action="file.php" name="frmid" >
  - Method specifies how the data will be sent
  - Action specifies the file to go to. E.g. file.php
  - id gives the form a unique name
- Post method sends all contents of a form with basically hidden headers (not easily visible to users)
- Get method sends all form input in the URL requested using name=value pairs separated by ampersands (&)
  - E.g. process.php?name=trevor&number=345
  - Is visible in the URL shown in the browser

# PHP - Dealing with the client

- All form values are placed into an array
- Assume a form contains one textbox called “txtName” and the form is submitted using the post method, invoking process.php
- process.php could access the form data using:
  - $_POST[‘txtName’]
- If the form used the get method, the form data would be available as:
  - $_GET[‘txtName’]

# PHP - Dealing with the client

- For example, an HTML form:
  - <form id="showmsg" action="show.php" method="post">
    - <input type="text" id="txtMsg" value="Hello World" />
    - <input type="submit" id="submit" value="Submit">
  - </form>

# GET vs. POST

- Both GET and POST create an array (e.g. array( key => value, key2 => value2, key3 => value3, ...)).
- This array holds key/value pairs, where
  - **keys are the names of the form controls**
  - **values are the input data from the user.**
- Both GET and POST are treated as $_GET and $_POST.

# GET vs. POST

- $_GET is an array of variables passed to the current script via the URL parameters.
- $_POST is an array of variables passed to the current script via the HTTP POST method.
- **These are superglobals, which means**
  - They are always accessible, regardless of scope .
  - You can access them from any function, class or file without having to do anything special.

# When to use GET?

- Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL).
- GET also has limits on the amount of information to send.
- The limitation is about 2000 characters.

# When to use GET?

- However, because the variables are displayed in the URL, it is possible to bookmark the page.
- GET may be used for sending non-sensitive data.
- GET should NEVER be used for sending passwords or other sensitive information!

# When to use POST?

- Information sent from a form with the POST method is **invisible to others**
- all names/values are embedded within the body of the HTTP request and
- has **no limits** on the amount of information to send.

# When to use POST?

- Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.
- **Developers prefer POST for sending form data.**

# Using HTML Forms

```php
<?php
   if( $_POST["name"] || $_POST["age"] ) {
      if (preg_match("/[^A-Za-z'-]/",$_POST['name'] )) {
         die ("invalid name and name should be alpha");
      }

      echo "Welcome ". $_POST['name']. "<br />";
      echo "You are ". $_POST['age']. " years old.";

      exit();
   }
?>
<html>
   <body>

      <form action = "<?php $_PHP_SELF ?>" method = "POST">
         Name: <input type = "text" name = "name" />
         Age: <input type = "text" name = "age" />
         <input type = "submit" />
      </form>

   </body>
</html>
```

It will produce the following result –

# PHP Arrays

- > An array variable is a storage area holding a number or text. The problem is, a variable will hold only one value.
- > An array is a special variable, which can store multiple values in one single variable.

# PHP Arrays

- If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1="Saab";
$cars2="Volvo";
$cars3="BMW";
```

# PHP Arrays

- > However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
- > The best solution here is to use an array.
- > An array can hold all your variable values under a single name. And you can access the values by referring to the array name.
- > Each element in the array has its own index so that it can be easily accessed.

# PHP Arrays

**In PHP, there are three kind of arrays:**

- **> Numeric array** - An array with a numeric index
- **> Associative array** - An array where each ID key is associated with a value
- **> Multidimensional array** - An array containing one or more arrays

# PHP Numeric Arrays

- > A numeric array stores each array element with a numeric index.
- > There are two methods to create a numeric array.

# PHP Numeric Arrays

- In the following example the index is automatically assigned (the index starts at 0):

```
$cars=array("Saab","Volvo","BMW","Toyota");
```

- In the following example we assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

# PHP Numeric Arrays

- In the following example you access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

```
Saab and Volvo are Swedish cars.
```

- The code above will output:

# PHP Associative Arrays

- > With an associative array, each ID key is associated with a value.
- > When storing data about specific named values, a numerical array is not always the best way to do it.
- > With associative arrays we can use the values as keys and assign values to them.

# PHP Associative Arrays

- In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

- This example is the same as the one above, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

# PHP Associative Arrays

The ID keys can be used in a script:

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output:

```
Peter is 32 years old.
```

# PHP Multidimensional Arrays

- In a multidimensional array, each element in the main array can also be an array.
  - For a two-dimensional array you need two indices to select an element
  - For a three-dimensional array you need three indices to select an element
- And each element in the sub-array can be an array, and so on.

# PHP Multidimensional Arrays

```php
<?php
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );

echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
```

Result:

Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15.

# PHP Multidimensional Arrays

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2] .
" a part of the Griffin family?";
```

The code above will output:

```
Is Megan a part of the Griffin family?
```

# Concatenation

- Use a period to join strings into one.

```
<?php
$string1="Hello";
$string2="PHP";
$string3=$string1 . " " . $string2;
Print $string3;
?>
```

```
Hello PHP
```

# PHP include and require Statements

- The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.
- Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

# PHP include and require Statements

- **The include and require statements are identical, except upon failure:**
  - require will produce a fatal error (E_COMPILE_ERROR) and stop the script
  - include will only produce a warning (E_WARNING) and the script will continue
- Use **require** when the file is required by the application.
- Use **include** when the file is not required and application should continue when file is not found.

# PHP include and require Statements

Assume we have a standard footer file called "footer.php", that looks like this:

```
<?php
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";
?>
```

```
<!DOCTYPE html>
<html>
<body>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
<?php include 'footer.php';?>

</body>
</html>
```

Result:

**Welcome to my home page!**

Some text.

Some more text.

Copyright © 1999-2016 W3Schools.com

# What is a Cookie?

- A cookie is often used to identify a user.
- A cookie is a small file that the server embeds on the user's computer.
- Each time the same computer requests a page with a browser, it will send the cookie too.
- With PHP, you can both create and retrieve cookie values.

# Create Cookies With PHP

- **A cookie is created with the setcookie() function.**

Syntax

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

- Only the *name* parameter is required. All other parameters are optional.

# Create Cookies With PHP

- PHP can
  - **Create/Retrieve a Cookie**
  - **Modify a Cookie Value**
  - **Delete a Cookie**
  - **Check if Cookies are Enabled**

- The **setcookie()** function must appear BEFORE the <html> tag.

# What is a PHP Session?

- A session is a way to store information (in variables) to be used across multiple pages.
- Unlike a cookie, the information is not stored on the users computer.
- Session variables hold information about one single user, and are available to all pages in one application.

# What is a PHP Session?

- When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.
- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

# Start a PHP Session

- A session is started with the session_start() function.
- Session variables are set with the PHP global variable: $_SESSION.

```php
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

# How does it work?

- Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12.
- Then, when a session is opened on another page, it scans the computer for a user-key.
- If there is a match, it accesses that session, if not, it starts a new session.

# PHP - Error & Exception Handling

- Error handling is the process of catching errors raised by your program and then taking appropriate action.
- If you would handle errors properly then it may lead to many unforeseen consequences.
- Its very simple in PHP to handle an errors.

# PHP - Error & Exception Handling

- Lets explain there new keyword related to exceptions.
  - **Try** − A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown".
  - **Throw** − This is how you trigger an exception. Each "throw" must have at least one "catch".
  - **Catch** − A "catch" block retrieves an exception and creates an object containing the exception information.

# PHP - Error & Exception Handling

- There are following functions which can be used from **Exception** class.
  - **getMessage()** – message of exception
  - **getCode()** – code of exception
  - **getFile()** – source filename
  - **getLine()** – source line
  - **getTrace()** – n array of the backtrace()
  - **getTraceAsString()** – formated string of trace

# Exercise

| | |
|---|---|
| Salary of Mr. A is | 1000$ |
| Salary of Mr. B is | 1200$ |
| Salary of Mr. C is | 1400$ |

# Exercise

| | |
|---|---|
| Salary of Mr. A is | 1000$ |
| Salary of Mr. B is | 1200$ |
| Salary of Mr. C is | 1400$ |

view plain copy to clipboard print ?

```
01. <?php
02. $a=1000;
03. $b=1200;
04. $c=1400;
05. echo "<table border=1 cellspacing=0 cellpading=0>
06. <tr> <td><font color=blue>Salary of Mr. A is</td> <td>$a$</font></td></tr>
07. <tr> <td><font color=blue>Salary of Mr. B is</td> <td>$b$</font></td></tr>
08. <tr> <td><font color=blue>Salary of Mr. C is</td> <td>$c$</font></td></tr>
09. </table>";
10. ?>
```


Thank you!
Any Questions ?