Creating Dynamic Web Sites

Ed Crowley

Chapter 3

Dynamic Web Sites

- Can alter content in response to differing situations.
- Easier to maintain
- More responsive to users
- Emphasis here will be on modularization and form handling techniques.
- Absolute and relative paths are also covered.
- Functions are introduced.

Web Applications

- Three ways to create more sophisticated Web applications.

1. External files.
   - More complex sites often benefit from modularizatoin.
2. Sophisticated handling of HTML forms.
3. Define/use your custom functions.

- To this point, every script has consisted of a single file that contains all of the required HTML and PHP code.
- Often when developing more complex Web sites, this approach isn't practical.

Modularization

- Better way to create dynamic Web applications is to divide scripts and Web sites into modules.
  - Each module stored in its own file.
  - Extract HTML from PHP or separate out commonly used processes.

Four PHP functions for incorporating external files:

1. include( )
2. include_once( )
3. require( )
4. require_once( )

- For Example:

```
include_once('filename.php');
require('/path/to/filename.html');
```

Include & Require

- Have the effect of taking all the content of the included file and dropping it in the parent script (the one calling the function).
- Unless the file contains code within the PHP tags, PHP will treat the included code as HTML.
  - Send it directly to the browser.
- Included file's extension doesn't matter.
  - Symbolic name and extension can help convey file's purpose.
    - An included file of HTML might use .inc.html.
- Absolute or relative paths to the included file can be used.

Absolute vs. Relative Paths

- Any external file, be it an included file, or an image, can be referenced using either an absolute or a relative path.
- Absolute path references a file starting from the root directory of the computer:

```
include ('C:/php/includes/file.php');
include('/usr/xyz/includes/file.php');
```

- Assuming file.php exists in the named location, the inclusion will work, no matter the location of the referencing (parent) file (barring any permissions issues).
- Can also use FQDN…

Relative Path

- Relative path uses the referencing (parent) file as starting point.
- To move up one folder, use two periods together.
- To move into a folder, use its name followed by a slash.
- So assuming the current script is in the www/ex1 folder and you want to include something in www/ex2, the code would be:

```
include('../ex2/file.php');
```

- As long as the files maintain their current relationship to each other, a relative path will remain accurate, even if the site is moved to another server.

Differences between include and require

- When working properly, the include( ) and require( ) functions are exactly the same.
- But they behave differently when they fail.
- If an include( ) function doesn't work (it cannot include the file for some reason), a warning will be printed to the Web browser, but the script will continue to run.
- If require( ) fails, an error is printed and the script is halted.

Include Once & Require Once

- Both functions also have a *_once( ) version, which guarantees that the file in question is included only once regardless of how many times a script may (presumably inadvertently) attempt to include it.

```
require_once('filename.php');
include_once('filename.php');
```

- Because require_once( ) and include_ once( ) require extra work from the PHP module (i.e., PHP must first check that the file has not already been included).
- It's best not to use these two functions unless a redundant include is likely to occur (which can happen on complex sites).

Site Structure

- When using multiple files, site structure becomes more important.

Two primary site considerations:

1. Ease of maintenance
2. Security

- Using external files that hold standard procedures (i.e., PHP code), CSS, JavaScript, and HTML greatly improves ease of maintenance.

.inc and .html Extensions

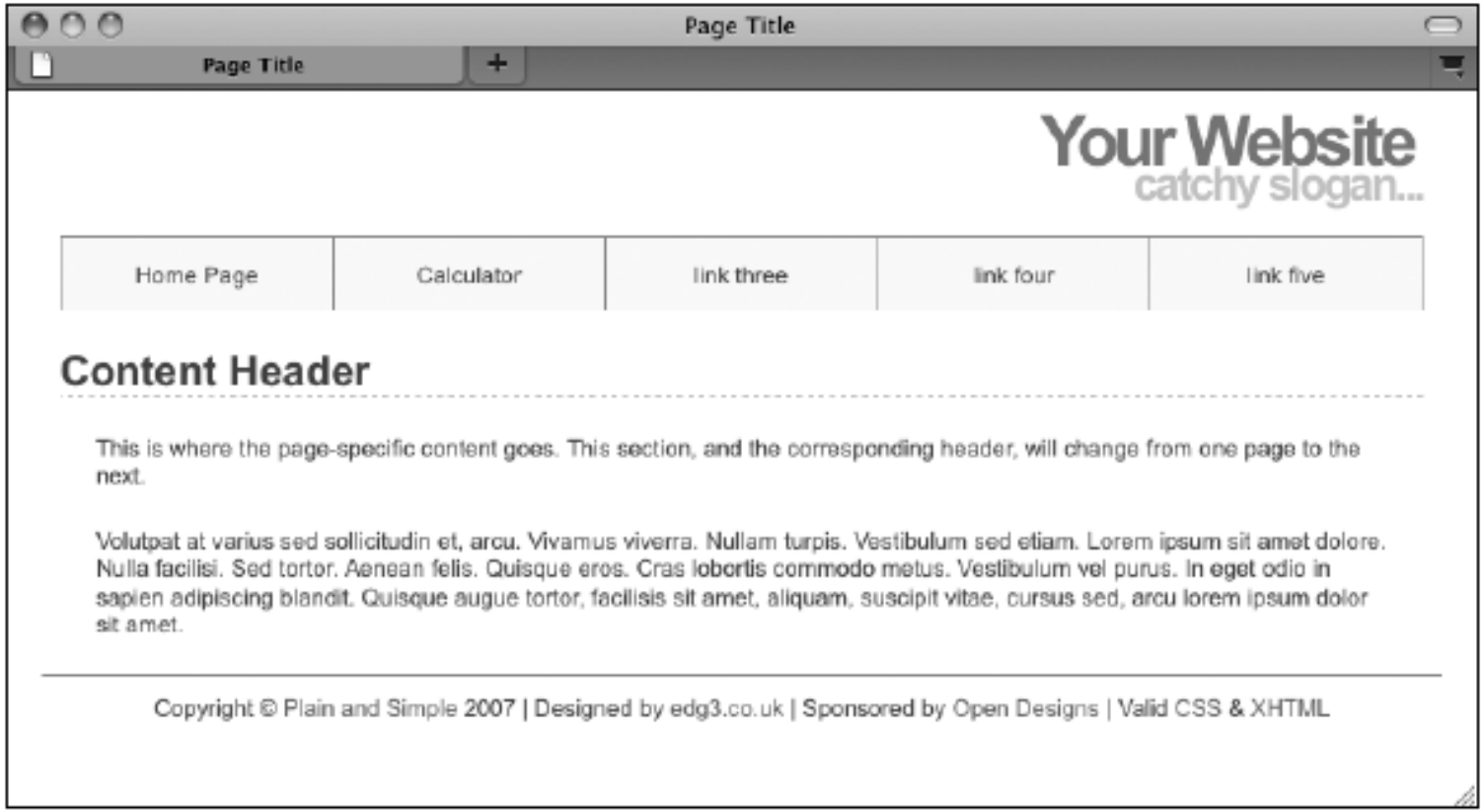
- For documents where security is not an issue, (such as HTML templates), use the .inc or .html file extension
- For files that contain more sensitive data (such as database access information), use .php.
- You can also use both .inc and .html or .php so that a file is clearly indicated as an include of a certain type: db.inc.php or header.inc.html

Modularization Example

- Included files separate primary HTML formatting from PHP code.
  - Then, remaining examples will be able to have the same appearance—as if they are all part of the same Web site—without needing to rewrite common HTML every time.
- This technique creates a template system.
  - Easy way to make large applications consistent and manageable.

Script 3.1 Index.html

/includes/style.css



Original design from:

https://www.edg3.uk/templates/

- Display of Script 3.1 and style.css

Compartmentalization

- Now, I'm going to make a new folder (ch3a) and place the following files in it…
- Script 3.2 /includes/header.html
- Script 3.3 /includes/footer.html
- Script 3.4 /script_03_04/index.php

```
1   <?php # Script 3.4 - index.php
2   $page_title = 'Welcome to this Site!';
3   include ('includes/header.html');
4   ?>
5
6   <h1>Content Header</h1>
7
8      <p>This is where the page-specific content goes. This section, and the
       corresponding header, will change from one page to the next.</p>
9
10     <p>Volutpat at varius sed sollicitudin et, arcu. Vivamus viverra. Nullam turpis.
       Vestibulum sed etiam. Lorem ipsum sit amet dolore. Nulla facilisi. Sed tortor.
       Aenean felis. Quisque eros. Cras lobortis commodo metus. Vestibulum vel purus.
       In eget odio in sapien adipiscing blandit. Quisque augue tortor, facilisis sit
       amet, aliquam, suscipit vitae, cursus sed, arcu lorem ipsum dolor sit amet.</p>
11
12  <?php
13  include ('includes/footer.html');
14  ?>
```

Index.php

- Internally, index.php is much different than Script 3.1 index.html.
- Nonetheless, when accessed through a web browser, it produces identical html.

Now:

- Display both files on your monitor.
- Right click and do a display source.
- Is the generated html identical to the original html?

Handling HTML Forms Revisited

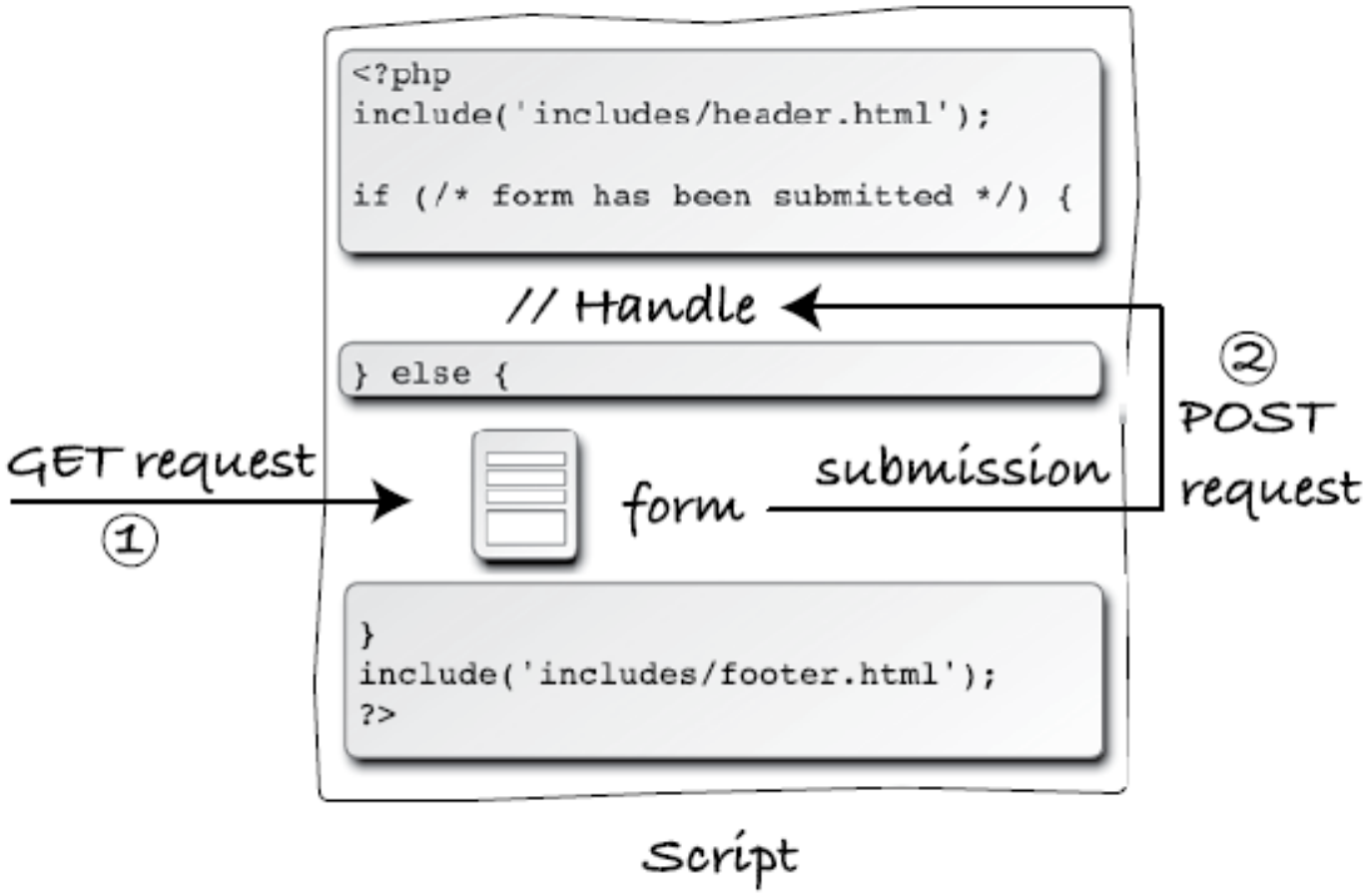
- Previous HTML Form examples used two separate files: one that displays the form and another that receives its submitted data.
- There are advantages to putting the entire process into one script.
- To have one page both display and handle a form, a conditional must check which action (display or handle) should be taken:

```
if (/* form has been submitted */) {
// Handle the form.
} else {
// Display the form.
}
```

Determining If Form Has Been Submitted

- When a form uses the POST method and gets submitted back to the same page, two different types of requests are made.
- First request, which loads the form, is a GET request.
  - Standard request made of most Web pages.
- A second request is made when a form is submitted, this time a POST request (as long as the form uses the POST method).

Two Requests



REQUEST_METHOD

- You can test for a form's submission by checking the requestmethod, found in the $_SERVER array:

```
if ($_SERVER['REQUEST_METHOD'] = =
  'POST') {
// Handle the form.
} else {
// Display the form.
}
```

Handle the Form

- If you want a page to handle a form and then display it again (e.g., to add a record to a database and then give an option to add another), drop the else clause:

```
if ($_SERVER['REQUEST_METHOD'] = =
  'POST') {
// Handle the form.
}
// Display the form.
```

- Using that code, a script will handle a form if it has been submitted and display the form every time the page is loaded

Calculator Script

- To demonstrate this (having the same page both display and handle a form), let's create a calculator that estimates the cost and time required to take a car trip, based upon user-entered values.

- Calculator Script Script 3.5

- Script has two logical parts.
  - An HTML part – the form
  - A PHP part – the calculator
- Lets look briefly at each part…

Calculator HTML

```
<form action="calculator.php" method="post">
<p>Distance (in miles): <input type="text" name="distance" /></p>
<p>Ave. Price Per Gallon: <span class="input">
<input type="radio" name="gallon_price" value="3.00" /> 3.00
<input type="radio" name="gallon_price" value="3.50" /> 3.50
<input type="radio" name="gallon_price" value="4.00" /> 4.00
</span></p>
<p>Fuel Efficiency: <select name="efficiency">
<option value="10">Terrible</option>
<option value="20">Decent</option>
<option value="30">Very Good</option>
<option value="50">Outstanding</option>
</select></p>
<p><input type="submit" name="submit" value="Calculate!" /></p>
</form>
```

Note that file specified in form action is itself…

Calculator PHP

```
// Check for form submission:
if ($_SERVER['REQUEST_METHOD'] = = 'POST') {
// Minimal form validation:
if (isset($_POST['distance'], $_POST['gallon_price'], $_POST['efficiency']) &&
is_numeric($_POST['distance']) && is_numeric($_POST['gallon_price'])
&& is_numeric($_POST['efficiency']) ) {
// Calculate the results:
$gallons = $_POST['distance'] / $_POST['efficiency'];
$dollars = $gallons * $_POST['gallon_price'];
$hours = $_POST['distance']/65;
// Print the results:
echo '<h1>Total Estimated Cost</h1>
<p>The total cost of driving ' . $_POST['distance'] . ' miles, averaging ' . $_POST
['efficiency'] . ' miles per gallon, and paying an average of $' . $_POST['gallon_price'] . '
per gallon, is $' . number_format ($dollars, 2) . '. If you drive at an average of 65 miles
per hour, the trip will take approximately ' . number_format($hours, 2) . ' hours.</p>';
} else { // Invalid submitted values.
echo '<h1>Error!</h1>
<p class="error">Please enter a valid distance, price per gallon, and fuel efficiency.</p>';
}
} // End of main submission IF.
```

Sticky Forms

- A standard HTML form that remembers how it has been filled out.
- To preset what's entered in a text input, use its value attribute:

```
<input type="text" name="city" value="Innsbruck"
  />
```

- To have PHP preset that value, print the appropriate variable (this assumes that the referenced variable exists):

```
<input type="text" name="city" value="<?php echo
  $city; ?>" />
```

- To preset the status of radio buttons or check boxes (i.e., to pre-check them), add the code checked="checked" to their input tags. Using PHP, you might write:

```
<input type="radio" name="gender" value="F"
<?php if ($gender == 'F') {
echo 'checked="checked"';
}
?>
/>
```

Setting Value Textarea

- To preset the value of a textarea, print the value between the textarea tags:

```
<textarea name="comments" rows="10"
  cols="50"><?php echo $comments;
?>
</textarea>
```

- Note that the textarea tag does not have a value attribute like the standard text input.

Select

- To preselect a pull-down menu, add selected="selected" to the appropriate option.
- This is really easy if you also use PHP to generate the menu:

```
echo '<select name="year">';
for ($y = 2011; $y <= 2021; $y+ +) {
echo "<option value=\"$y\"";
if ($year = = $y) {
echo ' selected= "selected"';
}
echo ">$y</option>\n";
}
echo '</select>';
```

Rewriting Calculator.php

- Let's rewrite calculator.php so that it's sticky.
- Existing values will be present in $_POST variables.
- Also, since it's best not to refer to variables unless they exist, conditionals will check that a variable is set before printing its value.

- Script 3.6 Calculator.php

Changes from Script 3.5

- Notice change in distance…

<input type="text" name="distance" value="<?php if (isset($_POST ['distance'])) echo $_POST ['distance']; ?>" />

- For each of the three radio buttons, the following code must be added within the input tag:

```
<?php if (isset($_POST['gallon_price']) &&
  ($_POST['gallon_price'] = = 'XXX'))
  echo'checked="checked" '; ?>
```

- For each option, within the opening option tag, the following code is added:

```
<?php if (isset($_POST['efficiency']) &&
  ($_POST['efficiency'] = = 'XX')) echo
  'selected="selected"'; ?>
```

Creating Functions

- PHP has the capability for you to define and use your own functions for whatever purpose.
- Syntax for making a custom function is:

```
function function_name ( ) {
// Function code.
}
```

- Name of your function can be any combination of letters, numbers, and the underscore, but it must begin with either a letter or the underscore.
- You also cannot use an existing function name for your function (print, echo, isset, and so on).

Functions

- One valid function definition is:

```
function do_nothing( ) {
// Do nothing.
}
```

- In PHP, function names are case-insensitive unlike variable names), so you could call that function using do_Nothing( ) or DO_NOTHING( ) or Do_Nothing( ), etc., but not donothing( ) or DoNothing( ).
- The code within the function can do nearly anything, from generating HTML to performing calculations to calling other functions.

Most Common Reasons To Create Your Own Functions

- Associate repeated code with one function call.
- Separate sensitive or complicated processes from other code.
- Make common code bits easier to reuse.

Function Examples

- For this example, a function will be defined to output HTML code for generating theoretical ads.
- Function then be called twice on home page.

- Script 3.7 (index.php) and 3.8 (calculator.php)

- Function from Script 3.7

```
function create_ad( ) {
echo '<p class="ad">This is an annoying ad!
  This is an annoying ad! This is an
annoying ad! This is an annoying ad!</p>';
} // End of the function definition.
```

Functions that take Arguments 3.8

```
function create_gallon_radio($value) {
// Start the element:
echo '<input type="radio"
  name="gallon_price" value="' . $value .
  '"';
// Check for stickiness:
if (isset($_POST['gallon_price']) &&
  ($_POST['gallon_price'] = = $value)) {
}
```

Arguments

- Another variant on defining your own functions is to preset an argument's value.
- To do so, assign the argument a value in the function's definition:

```
function greet ($name, $msg = 'Hello')
  {
echo "$msg, $name!";
}
```

Default Argument Value

- The end result of setting a default argument value is that that particular argument becomes optional when calling the function.
  - If a value is passed to it, the passed value is used; otherwise, the default value is used.
- You can set default values for as many of the arguments as you want, as long as those arguments come last in the function definition.
- In other words, the required arguments must always be listed first.
- With the example function just defined, any of these will work:

```
greet ($surname, $message);
greet ('Zoe');
greet ('Sam' 'Good evening');
```

- However, just greet( ) will not work.

Default Argument Values

- Also, there's no way to pass $msg a value without passing one to $name as well (argument values must be passed in order, and you can't skip a required argument).
- To take advantage of default argument values, let's make a better version of the create_gallon_radio( ) function.
- As originally written, the function only creates radio buttons with a name of gallon_price.
- It'd be better if the function could be used multiple times in a form, for multiple radio button groupings (although the function won't be used like that in this script).

- Script 3.9

Changes in Script 3.9

- First, the name of the function is changed to be reflective of its more generic nature.
- Second, the function now takes a second argument, $name, although that argument has a default value, which makes that argument optional when the function is called.
- Third, change the function definition so that it uses the $name argument in lieu of gallon_price:

```
echo '<input type="radio" name="'. $name .'" value="' .
  $value .'"';
if (isset($_POST[$name]) && ($_POST[$name] = = $value)) {
echo ' checked="checked"';
}
```

- Three changes are necessary.
- First, $name is used for the name attribute of the element.
- Second, the conditional that checks for "stickiness" now uses $_POST[$name] twice instead of $_POST['gallon_price'].
- The function calls must be changed to use the new function name. But because the second argument has a default value, it can be omitted in these calls.
- The end result is the same as executing this call—

```
create_radio('4.00', 'gallon_price');
```

Returning Values

- Some, but not all, functions do this.
- For example, print will return either a 1 or a 0 indicating its success, whereas echo will not.
- Another example, the number_format( ) function returns a string, which is the formatted version of a number
- To have a function return a value, use the return statement.

Return

- This function might return the astrological sign for a given birth month and day:

```
function find_sign ($month, $day) {
// Function code.
return $sign;
}
```

- A function can return a literal value (say a string or a number) or the value of a variable that has been determined within the function.
- When calling a function that returns a value, you can assign the function result to a variable:

```
$my_sign = find_sign ('October' 23);
```

- or use it as an argument when calling another function:

```
echo find_sign ('October' 23);
```

- Let's update the calculator.php script so that it uses a function to determine the cost of the trip.
- Script 3.10

Variable Scope

- Every variable in PHP has a scope, which is to say a realm in which the variable (and therefore its value) can be accessed.
- If you define $var, the rest of the page can access $var, but other pages generally cannot (unless you use special variables).
- Since included files act as if they were part of the original (including) script, variables defined before an include( ) line are available to the included file (as you've already seen with $page_title and header.html).
- Further, variables defined within the included file are available to the parent (including) script after the include( ) line.
- User-defined functions have their own scope:
  - Variables defined within a function are not available outside of it.
  - Variables defined outside of a function are not available within it.
- For this reason, a variable inside of a function can have the same name as one outside of it but still be an entirely different variable with a different value.
- This concept may seem confusing at first.

Scope

- To alter the variable scope within a function, you can use the global statement.

```
function function_name( ) {
global $var;
}
$var = 20;
function_name( ); // Function call.
```

- In this example, $var inside of the function is now the same as $var outside of it.
- This means that the function $var already has a value of 20, and if that value changes inside of the function, the external $var's value will also change.

Super Globals

- Another option for circumventing variable scope is to make use of the superglobals: $_GET, $_POST, $_REQUEST, etc.
- These variables are automatically accessible within your functions (hence, they are superglobal).
- You can also add elements to the $GLOBALS array to make them available within a function.
- All of that being said, it's almost always best not to use global variables within a function.
- Functions should be designed so that they receive every value they need as arguments and return whatever value (or values) need to be returned.
- Relying upon global variables within a function makes them more context-dependent, and consequently less useful.

Questions???