CSY2028
Web Programming
Topic 13

Tom Butler
thomas.butler@northampton.ac.uk

Topic 13

- Reusable HTML code
- Building a template function
- Template systems
  - Pros and cons

Last Week

- Last week covered reusable database functions
- You could use functions like:
  - save()
  - find()
  - delete()
- To very quickly and easily interact with the database

Reusable HTML

- Today we're going to do the same thing with HTML
- There are many cases where you want to display the same information in different ways
- Alternatively, you might want to use the same HTML for different data sets

Reusable HTML

- By separating out the HTML code from the code that interacts from the database there are several advantages with reusability

Separating HTML

- This code will display a list of books by a specific author:

```php
<?php

$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('authorId', 123);


echo '<ul>';
foreach ($books as $book) {
        echo '<li>' . $book['title'] . '</li>';
}
echo '</ul>';
```

Separating HTML

- This displays all the books by a publisher, however it's likely the same HTML would be used when something was searched for e.g.

```php
<?php

$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('publisher', 'Penguin Books');


echo '<ul>';
foreach ($books as $book) {
        echo '<li>' . $book['title'] . '</li>';
}

echo '</ul>';
```

Separating HTML

- Or find all the books based on a search term from a GET form:

```php
<?php

$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('title', $_GET['keyword']);


echo '<ul>';
foreach ($books as $book) {
        echo '<li>' . $book['title'] . '</li>';
}

echo '</ul>';
```

Separating HTML

- In all three cases, the type of information being displayed is the same: Information about any books that matched the search criteria
- The only thing that changes each time is the logic that queries the database
- The HTML and the code that creates it can be moved to its own file and included where needed:

```php
<?php
$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('title', $_GET['keyword']);

require 'book-list-template.php';
```

```php
<?php
$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('publisher', 'Penguin Books');

require 'book-list-template.php';
```

```php
<?php
$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('authorId', 123);

require 'book-list-template.php';
```

book-list-template.php

```php
<?php
echo '<ul>';
foreach ($books as $book) {
        echo '<li>' . $book['title'] . '</li>';
}

echo '</ul>';
```

Separating HTML

- This makes the HTML code reusable. Once you have the HTML code for displaying a list of books, you can re-use it as long as you create a variable called $books

```
<?php
echo '<ul>';
foreach ($books as $book) {
        echo '<li>' . $book['title'] . '</li>';
}

echo '</ul>';
```

Separating HTML

- You can also create *interchangeable* templates
- These will use the same variables but display the data in a different way e.g. displaying all the books in a table:

book-table-template.php

```
echo '<table>';
echo '<thead>';
echo '<tr>';
echo '<th>ISBN</th>';
echo '<th>Title</th>';
echo '<th>Published Date</th>';
echo '</tr>';
echo '</thead>';

foreach ($books as $book) {
        echo '<tr>';
        echo '<td>' . $book['ISBN'] . '</td>';
        echo '<td>' . $book['title'] . '</td>';
        echo '<td>' . $book['publishDate'] . '</td>';
        echo '</tr>';
}

echo '</table>';
```

Separating HTML

- Once the two templates exist, it's possible to quickly display all the books in a list:

```php
<?php
$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('title', $_GET['keyword']);

require 'book-list-template.php';
```

- Or a table by changing the require statement

```php
<?php
$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('title', $_GET['keyword']);

require 'book-table-template.php';
```

Separating HTML

- It's possible to write any kind of template that loops through an array of books and displays them in some way
- This will allow you to re-use the logic that selects the data with any template.
- This can be done by moving the data selection logic to its own file:

- books-by-keyword.php

```php
<?php
$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('title', $_GET['keyword']);
```

books-by-publisher.php

```php
<?php
$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('publisher', 'Penguin Books');
```

```php
<?php
require 'books-by-publisher.php';
require 'book-table-template.php';
```

```php
<?php
$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('publisher', 'Penguin Books');
```

```php
echo '<table>';
echo '<thead>';
echo '<tr>';
echo '<th>ISBN</th>';
echo '<th>Title</th>';
echo '<th>Published Date</th>';
echo '</tr>';
echo '</thead>';

foreach ($books as $book) {
        echo '<tr>';
        echo '<td>' . $book['ISBN'] . '</td>';
        echo '<td>' . $book['title'] . '</td>';
        echo '<td>' . $book['publishDate'] . '</td>';
        echo '</tr>';
}

echo '</table>';
```

```php
<?php
require 'books-by-publisher.php';
require 'book-list-template.php';
```

```php
<?php
$booksTable = new DatabaseTable($pdo, 'book');

$books = $booksTable->find('publisher', 'Penguin Books');
```

```php
<?php
echo '<ul>';
foreach ($books as $book) {
        echo '<li>' . $book['title'] . '</li>';
}

echo '</ul>';
```

- And then mix-and-match the data set and the template:
- Find books by publisher and show them in a list:

```php
<?php
require 'books-by-publisher.php';
require 'book-table-template.php';
```

- Find books by publisher and show them in a table:

```php
<?php
require 'books-by-publisher.php';
require 'book-list-template.php';
```

- Find books by keyword and show them in a table

```php
<?php
require 'books-by-keyword.php';
require 'book-list-template.php';
```

- This adds a very high level of flexibility
- Because the HTML generation code is completely separated from the logic which fetches the data, it's possible to substitute either the data or the HTML output

Exercise 1

- 1) Building on Topic 11's exercise list.php, move the HTML generation code for listing messages into its own file `user-list-template.php` and require it
- 2) Create a new template `user-table-template.php` that lists all the users in a table with columns for each field in the database
- 3) Move the database selection code to its own file `data-users.php` and create `user-table.php` and `user-list.php`. These files should only include 2 lines: Require statements

Separating HTML

- By separating out the HTML from the rest of the code it's possible to make the HTML reusable
- Once you have written the code to generate the HTML you can reuse it anywhere
- Managing this can take some extra thought:
  - You need to consider how you are breaking up the code
  - Usually this will be done in small sections that do a single specific job, e.g. looping through a list of records

Separating HTML

- It's a good idea to put common header/footer HTML in its own files
- This can be done using different files, e.g. head.php and foot.php

- head.php

```
<!DOCTYPE html>
<html>
        <head>
                <title>My Website</title>
                <link rel="stylesheet" href="styles.css" />
        </head>
        <body>
                <header>
                        <h1>My Website</h1>
                </header>
                <nav>
                        <ul>
                                <li><a href="index.php">Home</a>
                                <li><a href="about.php">About</a>
                                <li><a href="contact.php">Contact</a>
                        </ul>
                </nav>
                <main>
```

- foot.php

```
        </main>

                <footer>
                        &copy; 2016
                </footer>
        </body>

</html>
```

head.php and foot.php

- This can then be used to build an entire page:

```php
<?php
require 'books-by-publisher.php';
require 'head.php';
require 'book-list-template.php';
require 'foot.php';
```

Create the $books variable congaing all the books by a publisher

Load the top of the HTML page, the <head> tag, navigation, etc

Display the list of books

Load the footer, copyright notice, etc

head.php and foot.php

- This approach works but can be difficult to manage
- If you have start tags and end tags in different files it's difficult to see whether a tag has been closed correctly or in the correct order
- Instead, it's better to put it all in one file with variables in place of the content

```
<!DOCTYPE html>
<html>
        <head>
                <title><?php echo $title; ?></title>
                <link rel="stylesheet" h      "styles.css" />
        </head>

        <body>

                <header>
                        <h1>My Website</h1>
                </header>

                <nav>
                        <ul>
                                <li><a href="index.php">Home</a>
                                <li><a href="about.php">About</
                                <li><a href="contact.php">Co   act</a>
                        </ul>
                </nav>

                <main>
                        <?php echo $content; ?>
                </main>

                <footer>
                        &copy; 2016
                </footer>
        </body>

</html>
```

Variables in place of content that will be replaced

Layout.php

- If this was placed in layout.php it could be used like this:

```php
<?php
$title = 'My Website - Home';

$content = '<p>Welcome to my website</p>';

require 'layout.php';
```

```php
<?php
$title = 'My Website - Contact';

$content = '<p>You can contact me by telephone and by email:...</p>';

require 'layout.php';
```

Layout.php

- However, this causes a problem with the templates from earlier:

```
<?php
require 'books-by-publisher.php';
require 'layout.php';
require 'book-list-template.php';
```

```
        &copy; 2016
                </footer>
        </body>

</html>
<ul>
        <li>Moby Dick</li>
        <li>Treasure Island</li>
</ul>
```

Layout.php

- Because the template contains the echo command and prints to the screen, the content of layout.php is included
- Then the contents of book-list-template.php is
- What's needed is to load the contents of the file book-list-template.php into a variable e.g.

```php
<?php
require 'books-by-publisher.php';
$content = require 'book-list-template.php';
$title = 'Books by Penguin Books';
require 'layout.php';
```

Layout.php

```php
<?php
require 'books-by-publisher.php';
$content = require 'book-list-template.php';
$title = 'Books by Penguin Books';
require 'layout.php';
```

- Unfortunately the require function in PHP doesn't quite work this way
- However, it is possible to do this, it just needs a little more work.

Output buffer

- When an echo statement is called, the content is normally sent to the browser
- However, it is possible to redirect that output into a “buffer”
- Instead of being printed to the page, any output sent via echo commands is stored in a “buffer” that can be accessed later

Output buffering

- The output buffer can be started with the code:

```
ob_start();
```

- Anything that would be printed to the screen after this has been called will be captured and written to the buffer instead

```
<?php
ob_start();
echo '<p>Welcome to my website</p>';
```

Output buffering

- You can then read from the buffer using

```php
$contents = ob_get_clean();
```

```php
<?php
ob_start();
echo '<p>Welcome to my website</p>';
$contents = ob_get_clean();
```

The **$contents** variable will now store the string
Welcome to my website
And it won't have been printed to the screen

Output buffering

- This also works with require statements:

```php
<?php
ob_start();
require 'page.php';
$contents = ob_get_clean();
```

```php
<?php
require 'books-by-publisher.php';

ob_start();
require 'book-list-template.php';
$content = ob_get_clean();
```

The **$content** variable will now store the HTML generated in the book-list-template.php file
And it won't have been printed to the screen

Output buffering

- By combining output buffering and layout.php from earlier it's possible to set the content of the layout for the dynamic pages:

```php
<?php
require 'books-by-publisher.php';

ob_start();
require 'book-list-template.php';
$content = ob_get_clean();


$title = 'Books published by Penguin Books';
require 'layout.php';
```

Output buffering

- Because this is something that you'll probably want to do a lot, it's better to move it to a function:

```
function loadTemplate($fileName) {
        ob_start();
        require $fileName;
        $contents = ob_get_clean();
        return $contents;
}
```

- And use it like this:

```
<?php
require 'books-by-publisher.php';

$content = loadTemplate('book-list-template.php');


$title = 'Books published by Penguin Books';
require 'layout.php';
```

Template function

- This will load the template, but there's a problem:
  - The $books variable does not exist inside the function loadTemplate

```php
function loadTemplate($fileName) {
        ob_start();
        require $fileName;
        $contents = ob_get_clean();
        return $contents;
}
```

```php
<?php
//From  books-by-publisher.php
$books = $booksTable->find('publisher', 'Penguin Books');

$content = loadTemplate('book-list-template.php');
```

Template function

- One solution to this is passing the $books variable as an argument:

```
function loadTemplate($fileName, $books) {
        ob_start();
        require $fileName;
        $contents = ob_get_clean();
        return $contents;
}
```

- However, this limits the usefulness of the loadTemplate function
- It can only be used with templates that need a single variable named $books

```
<?php

$books = $booksTable->find('publisher', 'Penguin Books');

$content = loadTemplate('book-list-template.php',$books);
```

Template function

- Instead, it's possible to pass an array of variables to the function:
-

```php
$books = $booksTable->find('publisher', 'Penguin Books');

$templateVars = ['books' => $books];
$content = loadTemplate('book-list-template.php',$templateVars);
```

```php
function loadTemplate($fileName, $templateVars) {
        ob_start();
        require $fileName;
        $contents = ob_get_clean();
        return $contents;
}
```

- And then reference that in the template:

```php
<?php
echo '<ul>';
foreach ($templateVars['books'] as $book) {
        echo '<li>' . $book['title'] . '</li>';
}

echo '</ul>';
```

Template function

- This now allows the loadTemplate function to load any template file and set any variables:

```php
function loadTemplate($fileName, $templateVars) {
        ob_start();
        require $fileName;
        $contents = ob_get_clean();
        return $contents;
}
```

```php
$templateVars = ['title' => 'My Website',
'content' =>'<p>Welcome to my website</p>'
];
$content = loadTemplate('layout.php',$templateVars);
```

```php
<!DOCTYPE html>
<html>
        <head>
                <title><?php echo $templateVars['title']; ?></title>
                <link rel="stylesheet" href="styles.css" />
        </head>
        <body>
…..
```

Template function

- This works, however it does require using an array for all the variables in the template

```
<!DOCTYPE html>
<html>
        <head>
                <title><?php echo $templateVars['title']; ?></title>
                <link rel="stylesheet" href="styles.css" />
        </head>

        <body>
…..
```

- For simplicity it would be better to use just the variable name:

```
<!DOCTYPE html>
<html>
        <head>
                <title><?php echo $title; ?></title>
                <link rel="stylesheet" href="styles.css" />
        </head>

        <body>
…..
```

Template function

- PHP contains a function called *extract()*
- This function takes an array
  - For each element in the array, it creates a variable with the name of the key

```php
$myArray = ['title' => 'My page title'];

extract($myArray);

echo $title;
```

```
Output:
My page title
```

Template function

- This can be used in the template function to create the variables before including the file:

```php
function loadTemplate($fileName, $templateVars) {
        extract($templateVars);
        ob_start();
        require $fileName;
        $contents = ob_get_clean();
        return $contents;
}
```

```php
$templateVars = ['title' => 'My Website',
'content' =>'<p>Welcome to my website</p>'
];
$content = loadTemplate('layout.php',$templateVars);
```

```php
<!DOCTYPE html>
<html>
        <head>
                <title><?php echo $title; ?></title>
                <link rel="stylesheet" href="styles.css" />
        </head>
        <body>
…..
```

Template function

- This function can then be used to load any template or load one template into another:

```php
$books = $booksTable->find('publisher', 'Penguin Books');

$templateVars = ['books' => $books];

$content = loadTemplate('book-list-template.php', $templateVars);




$templateVars = ['title' => 'My Website',
'content' => $content
];
$content = loadTemplate('layout.php', $templateVars);

echo $content;
```

Load the list template and store it
In $content
the $books variable is set
in $templateVars

Load the layout and set the content variable to the loaded template

```php
$penguinBooks = $booksTable->find('publisher', 'Penguin Books');

$templateVars = ['books' => $penguinBooks];

$content = loadTemplate('book-list-template.php', $templateVars);
```

```php
<?php
echo '<ul>';
foreach ($books as $book) {
        echo '<li>' . $book['title'] . '</li>';
}

echo '</ul>';
```

```
The variable $content will now store:
<ul>
     <li>Moby Dick</li>
     <li>Treasure Island</li>
</ul>
```

```php
$templateVars = ['title' => 'My Website',
'content' => $content
];

$layoutHTML = loadTemplate('layout.php', $templateVars);

echo $layoutHTML;
```

```
Contents of $content variable (from last slide)
<ul>
      <li>Moby Dick</li>
      <li>Treasure Island</li>
</ul>
```

```php
<!DOCTYPE html>
<html>
    <head>
         <title><?php echo $title; ?></title>
        <link rel="stylesheet" href="styles.css" />
    </head>

    <body>
       <header>
              <h1>My Website</h1>
       </header>
       <nav>
          <ul>
            <li><a href="index.php">Home</a>
            <li><a href="about.php">About</a>
            <li><a href="contact.php">Contact</a>
          </ul>
       </nav>

       <main>
         <?php echo $content; ?>
       </main>
       <footer>&copy; 2016</footer>
   </body>
</html>
```

```
<!DOCTYPE html>
<html>
    <head>
         <title>My title</title>
        <link rel="stylesheet" href="styles.css" />
    </head>

    <body>
       <header>
              <h1>My Website</h1>
       </header>
       <nav>
          <ul>
            <li><a href="index.php">Home</a>
            <li><a href="about.php">About</a>
            <li><a href="contact.php">Contact</a>
          </ul>
       </nav>

       <main>
         <ul>
            <li>Moby Dick</li>
            <li>Treasure Island</li>
          </ul>
       </main>
       <footer>&copy; 2016</footer>
   </body>
</html>
```

Layouts

- By having a single layout.php template file, it’s easier to manage. All the code that’s repeated on each page (the header/footer/navigation, etc) is in one place
- Making a change to layout.php will affect every page on the website

Templates

- Storting HTML in its own file has advantages:
  - It's easier to manage, the HTML is all in one place and on its own
  - It's easy to re-use the HTML for different pages
  - You can easily re-use the same logic with different HTML

Template Systems

- There are several popular libraries you can download to do this
- These are called "Template systems"
- Some popular implementations are:
  - Smarty http://www.smarty.net/
  - Twig http://twig.sensiolabs.org/
  - Blade https://laravel.com/docs/5.1/blade
  - Transphporm (Shameless plug) https://github.com/Level-2/Transphporm

Template Systems

- These all work in a very similar way and templates look like something like this using a custom syntax (Twig example):

```
{% for user in users %}
    * {{ user.name }}
{% else %}
    No users have been found.
{% endfor %}
```

Note: this is not PHP code

Template Systems

- The usefulness of these systems is debated.
- There are several reasons for using them, one of them is often cited as:
  - It prevents logic from markup
    - (Or PHP code from HTML code)
- However, this is irrelevant, does the code here contain less logic than the PHP equivalent?

```
<ul>
{% for user in users %}
    <li>{{ user.name }}</li>
{% endfor %}
</ul>
```

```
<ul>
<?php foreach($users as $user) { ?>
    <li><?= $user['name']; ?></li>
<?php } ?>
</ul>
```

Template systems

- Pros:
  - (arguably) easier to understand code
  - Security: Only certain functions e.g. basic functionality are available in the template. This allows you to give non-developers access to write some basic code without giving them access to everything that PHP can do
  - Some common tasks e.g. formatting dates/times can be slightly simpler

Template systems

- Cons:
  - An extra syntax to learn, increasing the learning curve
  - Extra code: The template system needs a lot of code (The Smarty library is several thousand lines of code)
  - Performance, template systems are slower, PHP needs to convert them to PHP code before running the templates
  - More difficult to debug: If there is an error, PHP won't be able to display the line number that contains the error
  - Each template system uses a different syntax, meaning if you encounter projects using different systems you have extra syntax to learn

Reusability

- In general, the more you break a problem up, the more code can be reused
  - If you have a long 500 line file, it's difficult to reuse because it will be doing a very specific job
  - By separating out the code into different files (or functions) it's possible to reuse chunks of the code
- This avoids copying/pasting code saving time and making changes simpler to do

Other advantages

- When code is separated, it's easy to test!
- One of the book templates from earlier:

```php
<?php
echo '<ul>';
foreach ($books as $book) {
        echo '<li>' . $book['title'] . '</li>';
}

echo '</ul>';
```

It's possible to use this code without needing a database set up

```php
$books = [
  ['title' => 'Moby Dick'],
  ['title' => 'Treasure Island'],
  ['title' => 'Wuthering Heights'],
  ['title' => 'A Tale of Two Cities']
];

echo loadTemplate('book-list-template.php', ['books' => $books]);
```

Testing

- This allows you to test how something will look without needing all the logic
- This is very useful when you're testing something that is only displayed after a form has been submitted.
- You can load some test data into the template and check it looks ok without having to fill out all the previous forms

```
$books = [
  ['title' => 'Moby Dick'],
  ['title' => 'Treasure Island'],
  ['title' => 'Wuthering Heights'],
  ['title' => 'A Tale of Two Cities']
];

echo loadTemplate('book-list-template.php', ['books' => $books]);
```

Testing

- This also allows you to test extremes that might break the layout. E.g. very long text and check that it doesn't break the way the page is displayed
- Along with empty values

Exercise 2

- 1) Create a `layout.php` by taking the HTML code from Topic 1's 3 column layout
- 2) Add placeholders to the new layout.php for $title and $content
- 3) Separate the logic from the HTML code in each of the PHP files from the last exercise. Create a template for each HTML form and list
  - Hint: listmessages.php might be hard, do that last!
  - Hint: You can have a template for "Record added" and reuse it as the success page for all form submissions.
- 4) **All** pages should appear within the common layout
- Hint: The completed list.php and relevant files are available on NILE