Week 6
Forms and Templates

CIS 086 • PHP and MySQL • Mission College

Tonight's Topics

- Review
- Midterm exam in week 8
- Form validation
- Two-part vs. All-in-one forms
- Advanced escaping to HTML
- Templates

What you should know so far

- Variables
- Data types
- Arrays
- Operators
- Conditions: If, switch
- Loops: For, while, do…while
- Functions and parameters
- Include files
- Strings, escape sequences, regular expressions

You should be learning ...

- More advanced HTML
  - Form tags: form, input, radio, checkbox, submit
  - Form attributes: action, method

- PHP
  - Regular expressions
    - I don't expect anyone to be/become an expert at RE in this class. I'm not sure if you'll need it for the midterm or final exam programming project.
  - Web page templates

You should be learning …

- PHP form handling
  - Autoglobals: $_GET, $_POST, etc.
  - URL tokens (name/value pairs)
  - Form handler
  - Validating user data
  - Handling multiple errors
  - Re-displaying a web form
  - Two-part form vs. All-in-one form

Midterm exam

- Two parts:
  - Multiple choice part
  - Coding part

- Multiple choice part
  - On Canvas, scored automatically

- Coding part
  1. Write the code to process a form
  2. Post the form on the Mission College PHP server
  3. Link the form to your home page

HTML Forms

- Chapter 11 in 4th edition book
- Chapter 12 in 3rd edition book

- Required elements:
  - Opening and closing <form> tags
  - Method
    - GET
    - POST
  - Action: the URL of the form handler
  - Input fields

Forms

- Forms **must** include:
  - ACTION attribute – this tells which PHP file will process the form input.
  - METHOD attribute – this can be POST or GET.
- Form **may** include:
  - ID – which may be useful if you have 2 forms on one page.
  - CLASS – which can be used with CSS to style the form.

Form fields

Input field types

- Text boxes
- Checkboxes
- Radio buttons
- Hidden fields
- Submit button

Other field types:

- Text areas
- Select menu

New features in HTML5:

- placeholder
- color picker
- number
- range
- date and time pickers

Form Fields

- All form fields **must** have:
  - **NAME** – tells the form processor which variable goes with which value.
- Some form fields must have:
  - **VALUE** – tells the form processor what is the value of a checkbox, radio button, menu option, etc.

Autoglobals

- $_GET
  - Use $_GET when you want to GET information from the server: such as a specific web page, a specific inventory item, or a specific category
- $_POST
  - Use $_POST when you want to POST information to the server: such as a change to the database, or an uploaded file.
- It takes a long time to get a feel for the difference in usage. I try to give you tips.

GET vs. POST

- GET puts your form data in the URL string.
- **If** you don't want the form data to be **visible** in the URL string, **or** if there is **a lot** of form data, use POST instead of GET.
- If the form is going to **query** information from a database, GET is usually OK.
- But if the form is going to **change** information in a database, you always want to use POST instead.
- Think of the difference between **looking up** a book on Amazon and **ordering** the book on Amaaon.

GET vs. POST

- $_GET
    - Puts the parameters in the URL (**visible** to the user).
    - Lets you **bookmark** your URL.
    - You want users to be able to **bookmark** their favorite products or categories in your inventory.
    - www.myfavoritebookstore.com/?isbn=1234567890&category=programming
    - The user can add more parameters to the URL manually, which is a **security** flaw.

GET vs. POST

- $_POST
  - Passes the parameters in a separate data structure that is **not visible** to the browser or the URL.
  - You **cannot bookmark** your URL.
  - Won't let users bookmark URLs that cause change to the database or that upload files.
  - The user cannot add parameters because they can come only from the form, which partially closes the security problem.
  - However, a sophisticated user can create their own form that adds parameters. This is hard to automate, though.

Form Validation

- Check whether all **required** fields are non-blank.
- A field might have only **spaces** entered, so you should trim before checking, or your regular expression should disallow strings that consist entirely of white space.
- Check whether **numeric** fields actually have numbers, and those numbers fall within a required range.
- Check whether fields have a sufficient number of characters. (Phone number, credit card number, SSN.)
- **Email** is particularly hard to check.
- Check for **multiple errors** and let the user know about all of them at one time.

Sanitizing input

- Some users may try to trick your page into doing bad things.

- These are actually string functions:
  - stripslashes ()
  - addslashes ()
  - htmlentities ()
  - strip_tags ()

Security

- A user might try to gain access to information by putting a manufactured name-value pair in the URL.
- Always check the values in $_GET against a list of expected responses.
    - This is less of a problem since **`register_globals`** was finally removed in PHP 5.4.
    - http://php.net/manual/en/ini.core.php#ini.register-globals
- If there is an unexpected $_GET name or value, ignore it or perform a default action.

Types of Forms

- **All-in-one**: a PHP file that has both the user interface and the form processing in one file.
    - Disadvantages: user interface and back-end together in one file. Some consider this bad practice. Seldom used in business web sites.

- **Two-part**: an HTML file that has the user interface, and a separate PHP file that has the form processing.
    - Disadvantage: if the user makes a mistake, you need to show them their errors somehow, and HTML does not make this easy.
    - You may have to include the user interface part twice, once in the HTML file, and again in the PHP file.

Other considerations

- If the user makes a mistake, you can show them the error using PHP. Redraw the form and highlight the error.

Advanced Escaping

- Escaping from HTML to PHP
- Escaping from PHP to HTML

More Autoglobals

- We will use these in future chapters.
- $_FILES
  - For uploading files to the server.
  - Chapter 5
- $_COOKIE and $_SESSION
  - For managing state information.
  - Chapter 9
- $_SERVER
  - For getting information such as the current URL and host name.
  - You can use this to distinguish between localhost (running at home) and php.missioncollege.edu.

3 Types of Templates

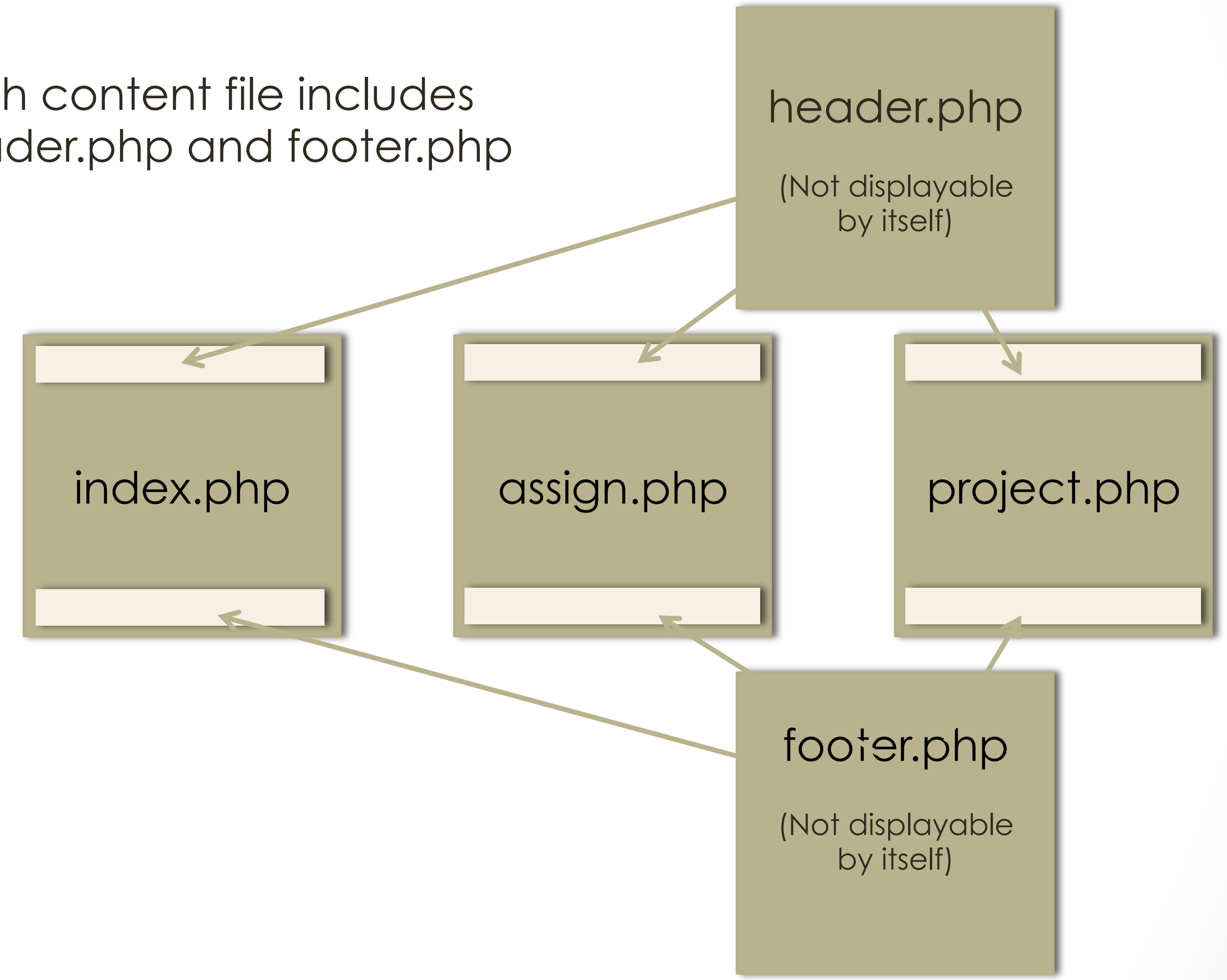
1. Each file holds its own content, and the boilerplate information (header, footer, navigation) is loaded from separate files using *include*.

2. A master file holds the boilerplate information (header, footer, navigation), and the content is loaded from separate files using *include*. The content file is specified by parameters passed in the URL.

3. A master file holds only the structure, and both the boilerplate information and the content are loaded from separate files using *include*.

Template #1

- Each content file includes header.php and footer.php

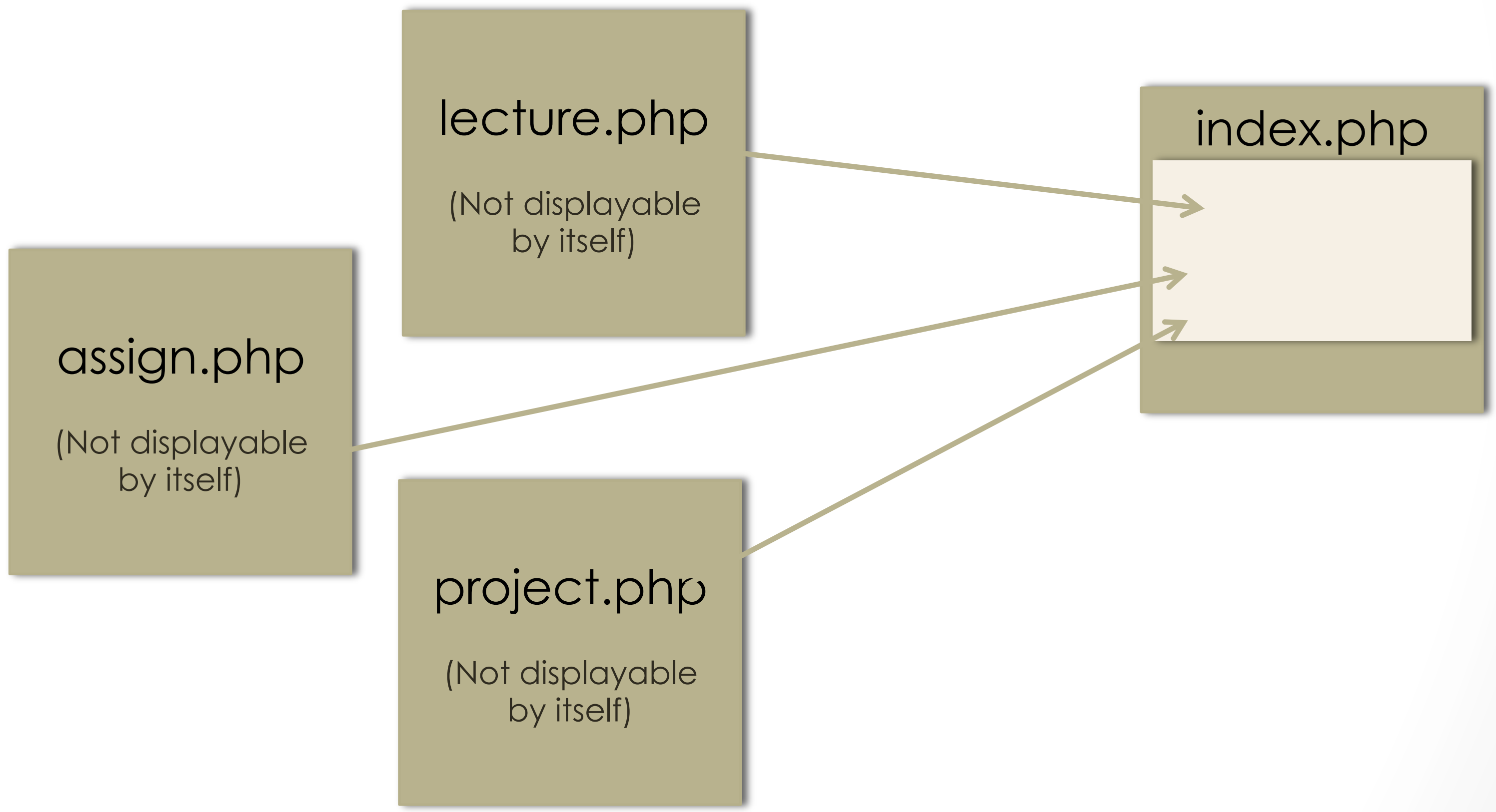


Template #1 code

```
include (“headers.php”);
include (“navigation.php”);
// insert your custom page code here
include (“footers.php”);
```

Template #2

- index.php chooses one content file to include


lecture.php
(Not displayable by itself)
assign.php
(Not displayable by itself)
project.php
(Not displayable by itself)
index.php

Template #2 code

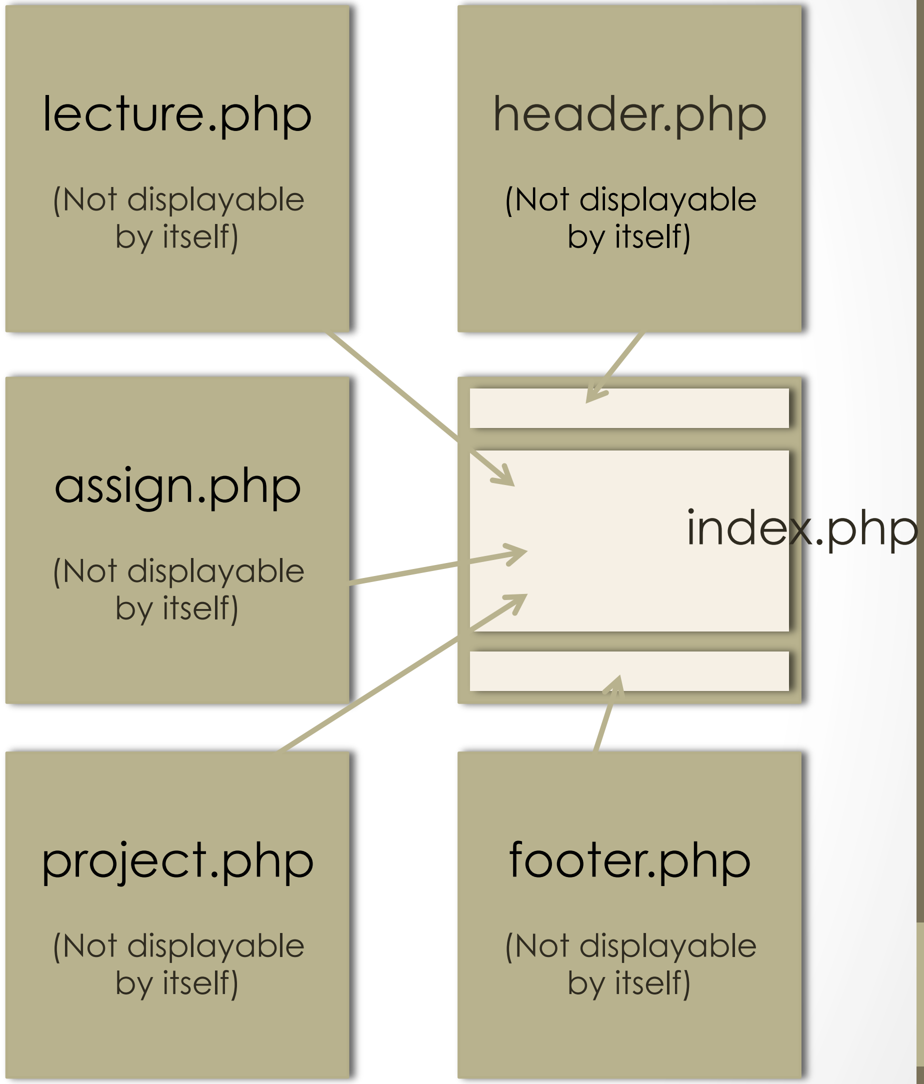
```
<!-- header HTML code here -->
<!-- navigation HTML code here -->

if ($_GET[‘page’] == ‘lecture’)
    include ‘lecture.php’;
else if ($_GET[‘page’] == ‘assign’)
    include ‘assign.php’;
else if ($_GET[‘page’] == ‘project’)
    include ‘project.php’;

<!-- footer HTML code here -->
```

Template #3

- index.php chooses one content file to include
- Index.php include both header.php and footer.php
- There is very little actual content in index.php


lecture.php
(Not displayable by itself)
header.php
(Not displayable by itself)
assign.php
(Not displayable by itself)
index.php
project.php
(Not displayable by itself)
footer.php
(Not displayable by itself)

Template #3 code

```
include (“headers.php”);
include (“navigation.php”);

if ($_GET[‘page’] == ‘lecture’)
    include ‘lecture.php’;
else if ($_GET[‘page’] == ‘assign’)
    include ‘assign.php’;
else if ($_GET[‘page’] == ‘project’)
    include ‘project.php’;

include (“footers.php”);
```