CS637 Midterm Review
Coverage: Duckett

- Chapter 1-2: Basics: Can skip pp. 53-56
- Chapter 3: Lists: all important
- Chapter 4: Links: all important
- Chapter 5: Images: can skip “old code”
- Chapter 6: Tables: all important
- Chapter 7: Forms: can skip fieldset, legend
- Chapter 8: Extra Markup: can skip iframe, meta info
- Chapter 9: Video and Audio: skip Flash (pp.202-212)
- Chapter 10: Intro CSS: all important
Coverage: Duckett

- Chapter 11: Color: Can skip pp. 255-256
- Chapter 12: Text: can skip @font-face, pp. 282-284, 288-292
- Chapter 13: Boxes: can skip pp. 318-322
- Chapter 14: Lists, tables, and forms: just understand the aligning form controls problem—see slides.
- Chapter 15: Layout: understand float
- Chapter 17: HTML5 Layout: all important
- Last few pages: useful indexes
Coverage: M&H

• Chapter 1: Intro
  – Can skip pp. 10-13, 22-23, 34-on (no exam questions on Netbeans, Notepad, phpAdmin)

• Chapter 2: PHP app basics: all important
  – Pg. 65: note that “isset($var)” requires $var “is set” (i.e. exists) and is not NULL
  – Pg. 73: skip NOT, AND, OR: use !, &&, || as in Java.

• Chapter 3: Intro MySQL
  – Note that we are assuming database identifiers are caseless, as in the SQL standard.
  – No exam questions on how things work in phpMyAdmin, just command line use.
• Chapter 4: PHP + MySQL
  – Note that we want to enable exceptions and put a try-catch around each database access.
  – In this chapter, the examples have PHP files all in the same directory, making includes easy.
  – However, the examples do not follow MVC rules exactly: the index.php on pg. 149 has controller code and then view code all in the same file.
  – It would be easy to fix this up to be MVC.

• Chapter 5: MVC
  – Pg. 161 important diagram
  – Pg. 164: redirection: send special response with new URL for browser to use immediately. Use here: to get the controller to restart its processing.
  – Make sure you understand everything about the two apps in this website, and how the code is subdivided into model, view, and controller code.
• Chapter 6 Testing
  – Types of errors: syntax, runtime, logic
  – PHP has non-fatal runtime errors, unlike Java

• Chapter 7 Forms
  – Also, HTML character entities like &lt;
  – Use of htmlspecialchars() to change < to &lt;, etc., “sanitize” strings

• Chapter 8 Control Statements
  – Control statements themselves are just like Java
  – Type coercion AKA type juggling: can lead to bugs
  – Use of identity operators === and !== to avoid type coercion
  – Dangerous to mix strings and numbers when they can get compared, as in searching and sorting

• Chapter 9: Strings and Numbers
  – Can skip htmlentities() function, sprintf, can always lookup details on a function
• Chapter 10 Dates—skip
• Chapter 11 Arrays
  – Arrays can have gaps, count($a) doesn’t count gaps, just non-null elements
  – Skip end($a) and key($a): use foreach to process an array with gaps
  – Arrays of arrays: important way to hold a collection of items, each with attributes, for example.
  – Array assignment copies array contents, and this happens when an array is used as a function argument (without &).
• Chapter 17 Creating DBs
  – Pg. 569: script my_guitar_shop2.sql
• Chapter 18 SQL: treated as review
• Chapter 19 PDOs
  – Pg. 625: how to turn on exceptions
Homework

• HW1
  – Basic HTML, URLs, SQL
  – Loading mysql db using command line
  – Writing and running command-line PHP

• HW2
  – HTML forms
  – Using XAMPP, book_apps
  – CSS
  – HTML with <aside>, CSS with float
Homework

• HW3
  – Page flow of pizza project
  – Foreign keys
  – More SQL
  – Start on Pizza1 project
HTTP request-response cycle

• A HTTP request-response cycle uses one TCP connection in two-way interchange from client to server and back--
  ➢ Client sends GET or POST request over TCP connection
  ➢ Server receives request, interprets URL, parameters, sends back response (HTML, image file (etc.), CSS file, or redirect)
  ➢ Client receives response, browser renders it (or immediately issues another GET if the response was a redirect)
• Server is “stateless”: treats each request as a completely new thing, spec’d by URL, parameters
• So it’s our job as programmers to pass info from one request (server execution of PHP) to the next in our app if needed, or use a database for long-term storage.
A simple HTTP request
GET / HTTP/1.1
Host: www.example.com

A simple HTTP response
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 136
Server: Apache/2.2.3

<html>
<head>
    <title>Example Web Page</title>
</head>
<body>
    <p>This is a sample web page</p>
</body>
</html>
Example of a POST Request

POST /path/script.php HTTP/1.1
Host: jmarshall.com
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

home=Coston&favorite_flavor=flies

• We never see this text in practice, only the resulting name-value information it carries.
• We can see the “form data” home=Coston, favorite_flavor=flies using Chrome’s Inspect>Network facility
HTTP parameters

- **GET case:** the parameters are part of the URL
  
  Example: GET request with parameters
  
  ```
  GET /myapp/index.php?action=add&name=foo HTTP/1.1
  (headers, then blank line)
  ```

- **POST case:** the parameters are in the POST body
  
  Example: POST request with parameters
  
  ```
  POST /myapp/index.php HTTP/1.1
  (request headers, then blank line)
  (POST body with encoded parameters)
  ```
HTTP request in server: to PHP or not

• HTTP request-response cycle:
  ➢ Client sends GET or POST command over TCP connection
  ➢ Server receives command, interprets URL, parameters, sends back response (HTML, image file (etc.), CSS file, or redirect)
  ➢ Client receives response, browser renders it

• Server interprets URL:
  ✓ If the URL ends in .html, .css, .jpg, etc., the server just reads the corresponding file and sends it back (static web server behavior)
  ✓ If the URL ends in .php, the server hands its processing over the PHP system:
    ❑ The PHP system puts the request parameters in global arrays _GET[] or _POST[], runs the PHP program
    ❑ The request parameters contain the user input from forms and links
    ❑ The PHP program generates the response, usually HTML or redirect
HTTP and the webapp: The MVC pattern

- The request comes into the controller, index.php
- The controller code interprets user input from \_GET[] or \_POST[]
- The controller code calls model functions to get needed data
- When the controller has all the data needed for the response, it forwards the request to the view file for composing the needed HTML.
  ✓ In some cases, the controller redirects back to itself or another controller
Project 1: a website like Ch05’s

- Note solution will be available for study, use for exam, on Saturday.
- MVC: all HTML-generating PHP (the view code) will be in separate source files from controller code, itself in “index.php” files.
- All database access will be handled in the model code, in the model directory.
- All shared (between users) variable data is held in the database.
- View code will be stored in the same directory as the index.php that controls it, unless it is view code that is included from code in more than one directory, like header.php and footer.php, in which case it is stored in the view directory.
- There is one top-level index.php that has links to the five major web apps of this website: for student operations and the various admin operations.
  - This index.php is not a “controller” because it does not use PHP code to interpret user input.
  - So it’s OK that it has HTML in it.
Project 1 web apps

• Like the web app “product_catalog” in the website “ch05_guitar_shop”, there is an “pizza” directory, containing an index.php to control the actions for students. Views involved in this activity are in other files in this directory.

• Like product_manager in ch05_guitar_shop, there is a topping directory, containing an index.php to control the topping actions for admins. Views involved in this activity are in other files in this directory.

• Similarly, also like product_manager in ch05_guitar_shop, there is an size directory, containing an index.php to control the pizza size actions for admins. Views involved in this activity are in other files in this directory.

• Similarly for order and day, more admin actions, with index.php controllers.
Pizza1 Database Data

• Each student order causes inserts in the database, easily retrievable in future requests.
• Student order status can be read from the DB given the room number.
• Each admin action changes or reads database data.
• The retrieved database data is held in variables only during a single request.
• Each request starts fresh getting needed DB data.
• This is important to make sure DB data isn’t “stale”.
• It also means the server only needs to allocate memory for the duration of the request (maybe 50 ms).
Pizza1 User-private data

• The only user id in pizza1 is the room number.
• The room number is like the current category in ch05_guitar_shop: it is specified by the user and not stored in the database.
• In the posted solution, the room number follows the user from request to request via a request parameter, like the current category in ch05_guitar_shop.
• It’s OK if your solution requires the user to specify the room number each time, since we only recently covered this point properly.
  – But be sure you understand the room number handling of the solution, for the midterm exam.
Pizza1 Controllers

- The controllers are coded in the index.php files of each webapp directory: pizza, topping, size, order, day
- The controllers are the most central code, defining what the app does.
- The controller interprets user input and specifically takes the user “command” from the “action” parameter. This parameter could be named otherwise of course, but M&H sticks with “action” consistently through the book.
- All calls to the model functions are from the controllers, not the view code (MVC rule).
- The controller has code specific for an action value, and then either forwards to a view file (include ...) or redirects back to itself, with needed info for that execution in parameters.
- Variable values persist through a PHP forward (include ...), but not a redirect, since that entails another browser GET request.
Pizza1 View Files

• View code will be stored in the same directory as the index.php that controls it, unless it is view code that is included from code in more than one directory, like header.php and footer.php, in which case it is stored in the view directory.

• View code in the app directories contains forms and links to capture user commands for that app.
  – A form has an action=URL, a link has href=URL
  – These URLs all point back to the controller: “index.php” or “.” or “”, by the MVC rule that each incoming request is received by a controller.
  – The controller will execute in a new request cycle, and the related user input is in request parameters.

• Thus the user stays with an app until they opt out by following a link provided by header.php, which is included in each app view file.
Pizza1 model code

- Each needed database action has a function in a file in the model directory.
- The function headers make up the database API, and their code is separate from the controller as expected in MVC.
- Although database actions can cause exceptions, no try {} catch {} is coded in model files.
- Instead, the calling code in the controllers has try {} around the model calls and a catch {} that forwards to database_error.php:
  ```php
  include ('../errors/database_error.php');
  ```
- The idea is that the controller better knows what to do when things go wrong than the model code itself.