CS637 Final 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: Murach & Harris

• 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 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: 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()

• Chapter 8 Control Statements
  – Control statements themselves are just like Java
  – Type coercion AKA type juggling: can disguise bugs
  – Use of identity operators 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.

Chapter 12 Cookies and Sessions: slide to come

Chapter 13 More on Functions: slide to come

Chapter 14 Objects: slide to come

Chapter 15 Regex, Validation: skip

Chapter 16 DB Design: treated as review

Chapter 17 Creating DBs: treated as review
- Pg. 569: script my_guitar_shop2.sql

Chapter 18 SQL: treated as review

Chapter 19 PDOs
- Pg. 625: how to turn on exceptions

Chapters 20-24: slides to come
Chapter 12 Cookies and Sessions

- We didn’t use cookies directly in our work, so can skip pp. 350-355
- We used cookies indirectly for session tracking
- Pg. 357 is important: how session tracking works
- Pg. 359: only session-start case studied: `session_start();
  session_set_cookie_params(60*60*24*365, '/');` second line overrides restrictive default values
- Pg. 360-361 session variables: important
- Can skip pp. 362-365
- Setting up and using a session variable: examples of cart array, (Cart app pp. 366-377), string name in numberguess example.
- Knowing when to use a session variable: normally build data structures for each request, from constants, DB data
Chapter 13: More on Functions

• Pp. 382-389 Args passed by reference, global, default parameters
• Pp. 392-393: simple library
• Pg. 395: include path basics (also understand project2’s main.php)
• Skip rest of chapter (namespaces, closures)
Chapter 14: PHP Objects

• We covered everything to pg 443
  – Properties, constructors, methods, static properties and methods
  – Copying object variables: doesn’t copy object itself, unlike PHP arrays (there’s an intermediate “object handle” that gets copied)
  – Need to use $this-> in method code to access properties (or self:: for static properties)
  – Even static properties are user-private, unlike Java. PHP keeps all memory objects purely user-private and whisks everything out of memory at the end of a request-response cycle.

• HW5 (solution) worked on a Cart object for ch14_guitar_shop, the OO guitar_shop, including unit tests and using a mock object.
• We used PDO objects for DB access
• We used Guzzle objects for sending GETs and POSTs for web services
Chapter 15: Validation

• Validation is checking user input to see if it makes sense.

• We skipped the validation coverage in Chapter 15, since it depends heavily on regex (regular expressions), also skipped.

• We had our own simpler validation example: numberguess. See Chapter12 slides, on why to redirect to the success page, and how to preserve good user input. Also in hw5.
Chapter 20: larger website

• Pp 658-659: Utility file main.php. Argued for using __DIR__ to set include_path
• Understand main.php of project2
• Sets the include_path to the project root directory so that #include(‘model/database.php’); works in any controller that has included main.php
• Sets $app_path to the local part of the url that selects the base directory.
• Example: pizza1 app: $app_path is underlined http://topcat.cs.umb.edu/cs637/eoneil37/pizza1
• $app_path is used for specifying URLs in shared view files: $app_path = ‘/book_apps/ch20_guitar_shop/’ for book’s site, for ex., in sidebar.php:
  
  <a href="<?php echo $app_path . 'admin'; ?>">Admin</a>

  becomes, in the HTML sent to the browser:
  
  <a href="/book_apps/ch20_guitar_shop/admin">Admin</a>

  a Document-root-relative URL
Chapter 20 #2

• We studied this guitar shop version, see slides.
• No session variable in use here.
• Note its adherence to MVC principles:
  – Index.php’s: controller code, needed calls to DB, no HTML
    • Interprets incoming request query parameters (user input)
    • Forwards to view file for HTML generation
    • Or redirects back to itself
  – One index.php has HTML: /admin/index.php has just links
  – View file: no calls to DB, finishes the request-response cycle by generating HTML for the response
Chapter 21: Secure HTTP

- Use of http: vs https: in URLs
- Server certificate: Authenticates the server that sends it. Browsers have a collection to check against.
- Both XAMPP and topcat.cs.umb.edu have Apache web servers that operate with “self-signed” certificates, not authoritative, and not in any browser’s collection. We see warnings in browser display.
- How to check if using incoming request is using https:

```php
$https = filter_input(INPUT_SERVER, 'HTTPS');
```
Chap 21: Form-based authentication

• Is for user authentication (user has username, password)
• Server is already authenticated at this point, should be doing https:
• We website programmers get to compose the login page to fit our needs
• How to encrypt passwords for saving them in DB:
  $password = sha1($email . $password);
• Once the user has successfully provided a password, the user is “logged in” to the website.
• The website remembers this fact by setting up a session variable named, say, ‘user’ or ‘admin’ or, in Chap. 21 case, ‘is_admin_user’.
Chapter 22

- How to use libcurl, though Chap. 22 example is obsolete.
- Also, it’s hard to use libcurl for advanced needs such as access to a response Location header.
- So we used the Guzzle PHP component, which itself uses libcurl but provides an easier OO programming interface.
- Chap 22 show how we can easily process JSON coming back by using `json_decode($json_data, true)` to create a PHP associative array of all the data.
- From Google YouTube web service, get a massive array back: all the info on many videos. More efficient than getting only top-level data back and having to query again.
Chapter 23: Files, Uploads

• We skipped this chapter, but it should be a resource to you. Of course PHP can work with files, like any decent programming language.
• Doing uploads is more challenging but is covered here.

Quick file example:

```php
$path = getcwd();    // or __DIR__
$items = scandir($path);
$file = fopen('listing.txt', 'wb');
foreach ($items as $item) {
    $item_path = $path . DIRECTORY_SEPARATOR . $item;
    if (is_dir($item_path)) {
        fwrite($file, $item . "\n");
    }
} 
fclose($file);
```
Chapter 24: an eCommerce site

• What this website has beyond chapter 20’s:
  – User login, admin login using SSL to protect passwords on the wire, and corresponding session variables to remember logged-in status
  – Working cart, held in a session variable
  – Secure checkout for user, creating an order (a wizard, i.e., a sequence of pages with only one way for the user to progress at each step)
  – https: for admin actions and user checkout, but not for user browsing.

• But it also depends on Chapter 20 features:
  – Has util/main.php for common code used in all controllers, and session_start() is now in main.php.
  – Sets up an include path based on the project root directory.
  – Uses document-root-relative URLs (with $app_path) for website navigation links.
HTTP Slides

• PHP webapps use:
  – GET, POST verbs
  – MIME types: text/html for pages
  – Response Status codes: 200, 301, 404, 500
  – Headers: Location for redirection (along with response code 301)
  – Query parameters (not in HTTP slides)

• PHP web services use, additionally:
  – PUT, DELETE verbs
  – MIME types: text/xml, text/json, others
  – Headers: Accept:, Content-type: if doing content negotiation
PHP Web Services beyond M&H

- In “PHP Web Services” slides, handout:
  - REST Web services: use HTTP directly, everything is a resource, stateless
  - Simple order service
  - Link to Dr Dobbs tutorial
  - Multilevel array examples
  - Project 2 orders
  - JSON (itself not on final)
  - json_encode/decode
  - Project 2 web services, resources, outline of code
  - Using the Guzzle PHP component for generating HTTP GETs, POSTs (i.e. client-side web services)
  - Testing the server side using command-line curl
  - Note: *shell scripts* not covered on final, just command-line curl
PHP Web Services

• In “PHP Web Services” slides
  – Naming resources
  – Query parameters in URIs
  – Links between resources
  – Content Negotiation
  – Idea of hypermedia (not on final)
Homework

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

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

• HW3
  – Page flow of pizza project
  – Foreign keys, and lack of FK constraints in book’s my_guitar_shop2.sql
  – More SQL
  – Start on Pizza1 project
Homework, continued

• HW4
  – Multi-level arrays drawn like pg. 337, used in code
  – File searches on the pizza1 solution to verify MVC principles
  – Managing state using hidden parameters vs. session variables, case of room number in pizza project
  – Passing arrays into functions: they are call-by-value, so arrays are logically copied in unless use &

• HW5
  – PHP object class Cart, with unit tests and a mock object
  – Using a session variable for numberguess
  – HTTP headers and status codes: observing requests and responses, complete with headers, with Chrome
Pizza1/Pizza2/ch20_guitar_shop: websites of similar setup

- MVC: all HTML-generating PHP (the view code) is in separate source files from controller code, itself in “index.php” files.
- All database access is handled in functions in the model code, in the model directory.
- All shared (between users) variable data is held in the database.
- View code is 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.
Pizza1/Pizza2/ch20_guitar_shop: websites of similar setup, cont.

• Not all “index.php” files are controllers. Some contain just links in HTML (top level index.php in pizza1/pizza2, admin/index.php in pizza2 and ch20_guitar_shop).
• No logins are required for users, and in fact, no session variables are in use.
• Pizza1/pizza2 does track the room number of the user, a kind of user id, and ch20_guitar_shop, like ch05_guitar_shop, tracts the current category of interest to the user. This tracking is done by parameters passed with requests.
Pizza1/Pizza2/ch20... Database Data

- Each student order in pizza1/pizza2 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).
User-private data

• The only user id in pizza1/pizza2 is the room number.
• Ch20_guitar_shop has no user id, only a notion of the current category of interest to the user, which is also in ch05_guitar_shop
• The pizza1/pizza2 room number is like the current category in ch05_guitar_shop: it is specified by the user and not stored in the database.
• The room number follows the user from request to request via a request parameter, like the current category in ch05_guitar_shop/ch20_guitar_shop.
• HW4 explored how we could use a session variable for the room number instead.
• Ch24_guitar_shop has user and admin logins, and creates user and admin session variables to remember the logged-in state
From Intro: PHP is made for web apps

It maintains global arrays for easy access to HTTP request parameters, HTTP headers, session variables, and cookies.

Don’t worry if this doesn’t make sense yet!
Added: Hope it does now!

PHP deallocates or saves away memory data after each request cycle is done, to minimize memory footprint. In other words, it assumes it is sharing the system with many other requestors.

Added: Even session variables are saved away to files at the end of each request: “serialized” and later “deserialized” back to memory in a new request.