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 <
  - 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
- Chapter 21 Securing the website: SSL, HTTPS, user authentication

Chapter 12 Cookies and Sessions

- We didn't use cookies directly in our work, so can skip pp. 350-355 (either edition, for all pages on this slide)
- 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 how to use session variables: important
- Can skip rest of chapter
- Knowing when to use a session variable: normally build data structures for each request, from constants, DB data.
- Used for remembering that a user is "logged in" (Chap. 21)

Chapter 13: More on Functions

- Pp. 382-389 (either ed.) Args passed by reference, global, default parameters
- Pp. 392-393 (394-395 for 3ed): simple library
- Skip rest of chapter (include path, namespaces, closures)

Chapter 14: PHP Objects

- We covered everything to the top of pg 443 (445) inclusive
  - 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.
- HW4 has a program with Product, Category, *DB objects
- We used PDO objects for DB access
- We used Guzzle objects for sending GETs and POSTs for web services in pizza2_phpclient
- We used the Slim App object for receiving GETs, POSTs, etc. in pizza2_server. Also Slim's Request and Response objects (short name set up by "use \Psr\Http\Message\ServerRequestInterface as Request")

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), not covered until the last week, and optional then.
- We had our own simpler validation example: PHP numberguess. See Chapter12 slides, on why to redirect to the success page to avoid double submission, and how to preserve good user input.
- Note that with JS, submitting a form does not itself involve a network request. The click event handler may do a POST to the server for the indicated action, and this can take time. But unlike the PHP case, the UI isn't frozen meanwhile, and can take away or gray out the submit button, for example, to avoid double submission.

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.
- XAMPP has an Apache web server that operate with a "self-signed" certificate, 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: `$https = filter_input(INPUT_SERVER, 'HTTPS');`
- The slides also covered the Javascript case, FYI.

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'.

PHP support for HTTP requests

- PHP usually comes with libcurl, a fairly primitive library supporting HTTP requests from PHP. See Nov. 6 slides for intro.
  - Second ed of Murach Chap. 22 only: How to use libcurl, though the 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.
  - See pizza2_phpclient for code using Guzzle.
- Data usually moves across the network in JSON format. We can easily process JSON coming in by using PHP's json_decode($json_data, true) to create a PHP associative array of all the data. Similarly json_encode(…).
- From Google's YouTube web service, we 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:

```
$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);
```

HTTP Basics

- PHP webapps (examples: pizza1, pizza2_phpclient) use:
  - GET, POST verbs
  - MIME types: text/html for HTML pages
  - Response Status codes: 200, 301, 400, 404, 500
  - Headers: Content-type:, Location for redirection (along with response code 301),
  - Query parameters Ex: …?action=list&user_id=3
- PHP web services use:
  - GET, POST, PUT, DELETE verbs
  - MIME types: application/json, others
  - Response Status codes: 200, 400, 404, 500
  - Headers: Content-type:, Location for reporting new URI assigned by server, (Accept:, if doing content negotiation, but this will not be on the final exam)
  - Query parameters: Can be used, but we didn't in pizza2_server

PHP Web Services

- 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
  - Provided ch05_gs_client/server projects
  - JSON (itself not on final exam)
  - 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

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 (counts as another hw)

Homework, continued

- HW4
  - Multi-level arrays drawn like pg. 337, used in code
  - 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 &
  - Using PHP objects, specifically Product and Category objects, in a command-line program to list all products
  - Intro to PHP Web Services: installing and using the provided ch05_gs_server
- Project 2 first delivery (counts as a hw)
- So 6 hw's, drop lowest score, provides 50 points.

Pizza1/Pizza2_phpclient/ch05_gs_client 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 (either direct or via web services) is handled in functions in the model code, in the model directory.
  - The controller calls the model functions, sets up variables with *all* the needed data of the view files, forwards to a view file, or redirects back to itself.
- 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_phpclient/ch05_gs_client websites of similar setup, cont.

- Not all "index.php" files are controllers. Some contain just links in HTML (top level index.php in each of these).
- No logins are required for users, and in fact, no session variables are in use.
- Pizza1/pizza2_phpclient does track the user_id of the current user, and ch05_gs_client tracks the current category of interest to the user. This tracking is done by parameters passed with requests.

User-private data

- The only user-private data in pizza1/pizza2 is the user id (user_id/selectedUser)
- Ch05_gs_client has no user id, only a notion of the current category of interest to the user
- In the pizza2 PHP client, the user id follows the user from request to request via a request parameter
  - HW4 explored how we could use a session variable for the user_id instead.
- In the JS client the selectedUser variable is simply held in the browser, never communicated to a server at all.

Database Data

- Each student order in pizza1/pizza2 causes inserts in the database, easily retrievable in future requests.
- Of course, in pizza2 the data goes over the network for the student actions, but the result is the same.
- Student order status can be read from the DB given the user_id.
- Each admin action (pizza1) changes or reads database data.
- The retrieved database data is held in variables only during a single request, in the PHP client.
  - In the JS client, some DB data is held longer, in the browser
- Each request starts fresh getting needed DB data (PHP case).
- This is important to make sure DB data isn't "stale".
  - The JS client should request changeable data (orders) frequently
  - Should refresh toppings too, though not expected in project solution
- It also means the webserver running the PHP client only needs to allocate memory for the duration of the request (maybe 50 ms).
  - The JS client is running in the user's browser, no server memory in use!

From Course 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.

24

Javascript Execution Steps

1. User browses to something.html
2. Server sends HTML to the client, as we have just seen
   - In that HTML: <script> with URL for Javascript code
   - Also URL for library code, if being used
3. Browser finds URL, does second HTTP request for Javascript (JS) code, again to a server.
   - JS code is loaded into browser memory
4. The JS code can detect load-complete event, start executing to set things up for itself.
   - Can build more HTML elements, for example.
   - Can attach event listeners to HTML elements
5. Later, when user clicks on something, it can be handled by JS immediately
   - No server involvement unless persistent changes are needed.

© 2014, Mike Murach & Associates, Inc.

What is a web application?

App vs. web app:

- An app (desktop or mobile) is a program that runs directly on the OS of the device.
  - Thus has sub-species: Android app, ioS app, Linux app, Windows app, MacOS app, etc., and each works only on its own OS.
  - An organization that wants all its users to use "its app" needs to implement all these versions.
- A web app runs inside a web browser. The web browser itself is an app on the desktop or mobile device.
  - Thus we would expect sub-species Chrome app, Safari app, Firefox app
  - But browsers are much more standardized than OSs, so we just say "web app" and expect it to run on any of these modern browsers.
  - An organization that wants all its users to use "its webapp" should only need to implement one version (possibly with some conditional code if using non-standardized features).
  - "Progressive web app" (PWA): web app that looks and feels like an app
    - See https://www.makeuseof.com/tag/progressive-web-apps/

A web application has a website

- A web app pulls its pages, etc., from a website on a server connected to the Internet (or possibly a smaller network).
- A web app can accept user input and send it back to its server for processing (the PHP way), or process it right in the browser (the Javascript way).
- A web app depends on its server for saving data persistently.
- A web app UI is much like an app UI: buttons, forms, text input, etc., but may be slowed down by the network (PHP case)