**DIGM-243: Web Authoring II**

**Week 05**

**Midterm Critique**

**GOLDEN ORDER of Wordpress site modifications:**

1. **Dashboard (easy, point and click)**
2. **CSS (medium, style sheet edit)**
3. **PHP (changelling, php code modifications)**

**Coding A WordPress Theme**

***PHP/MySQL, HTML5/CSS3/JavaScript, & WordPress:***

***Putting it all together***

As I mentioned before there are three roads that you can take.

1. You can design your own custom template from scratch.

2. You can use a framework or a pre-built theme as a Parent, and modify it with a Child Theme

3. You can find an existing theme, re-skin it, and tweak it to fit your needs

For the purpose of this class, I highly recommend you go with either option 3, since option 1 requires you really know WordPress, and item 2 has some inherent issues with .

**What Do You Need To Build A Theme?**

Themes consist of various files inside a self-contained folder inside of your WordPress install.

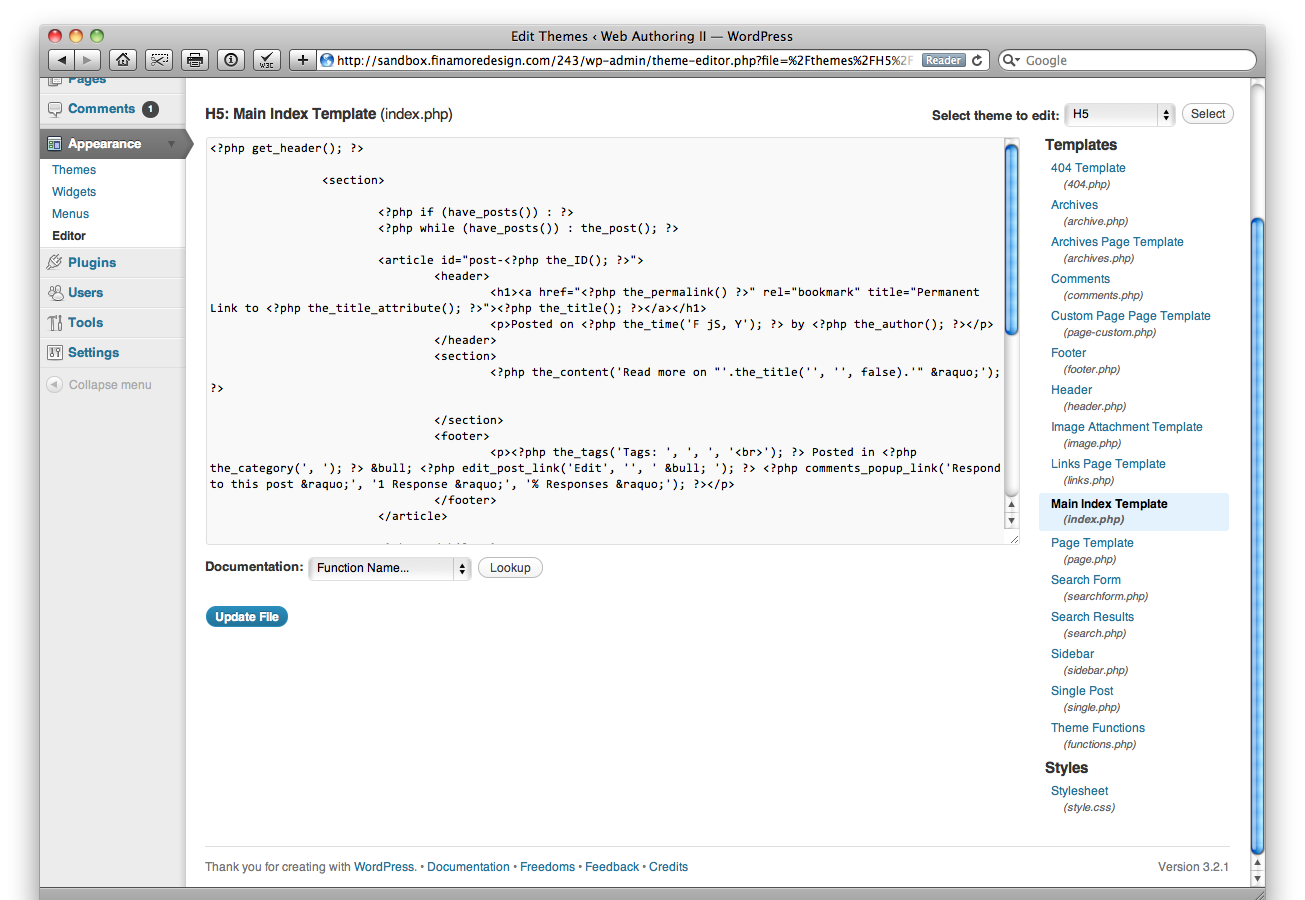
To have a theme work properly, the minimum number of files that you will need are 2: style.css and index.php.

style.css is not only where you write all the CSS you need to get the site looking the way you want it, it also holds a special header wrapped in a CSS comment at the top of the file. Without this, WordPress will not recognize your folder as a theme and will not let you apply it to your site in the Admin Panel.

Index.php serves as the general layout of your site. This is the ultimate fallback file. It is used when no other template is found in your theme that is a better match to the page you want to display.

While it is possible to build a theme in WordPress using only these two files, you would be severely limiting your site by doing so. WordPress allows you to create a full range of custom layout templates that not only give your site a more interesting effect, but also allows you to simplify your work.

Remember back in week two when we discussed visual design for a theme and we broke the page apart into wire frames with sections for header, footer, sidebar, nav, etc… When WordPress builds a single HTML page, it creates each of these parts via individual template files, then assembles them into a single page. You can see files like index.php, header.php, page.php, post.php, archive.php, sidepar.php, footer.php, in most themes you download.



In addition to template files for layouts of pages, WordPress also has a PHP file called functions.php. This file is where you can write your own custom functions and then have them accessible to you in any template file. This allows you to add features much like plugins to expand the functionality of your WordPress site. We will be using this file to add in our own custom taxonomy.

**What Other Templates Can Be Used?**

WordPress is great because you can customize as much or as little as you want. For just about any type of page it creates, WordPress allows you create a new template that dictates the information displayed and how that information is styled. Using functions provided by WordPress you can limit things like the number and/or category of posts it displays and where it display this information.

http://codex.wordpress.org/Template\_Hierarchy

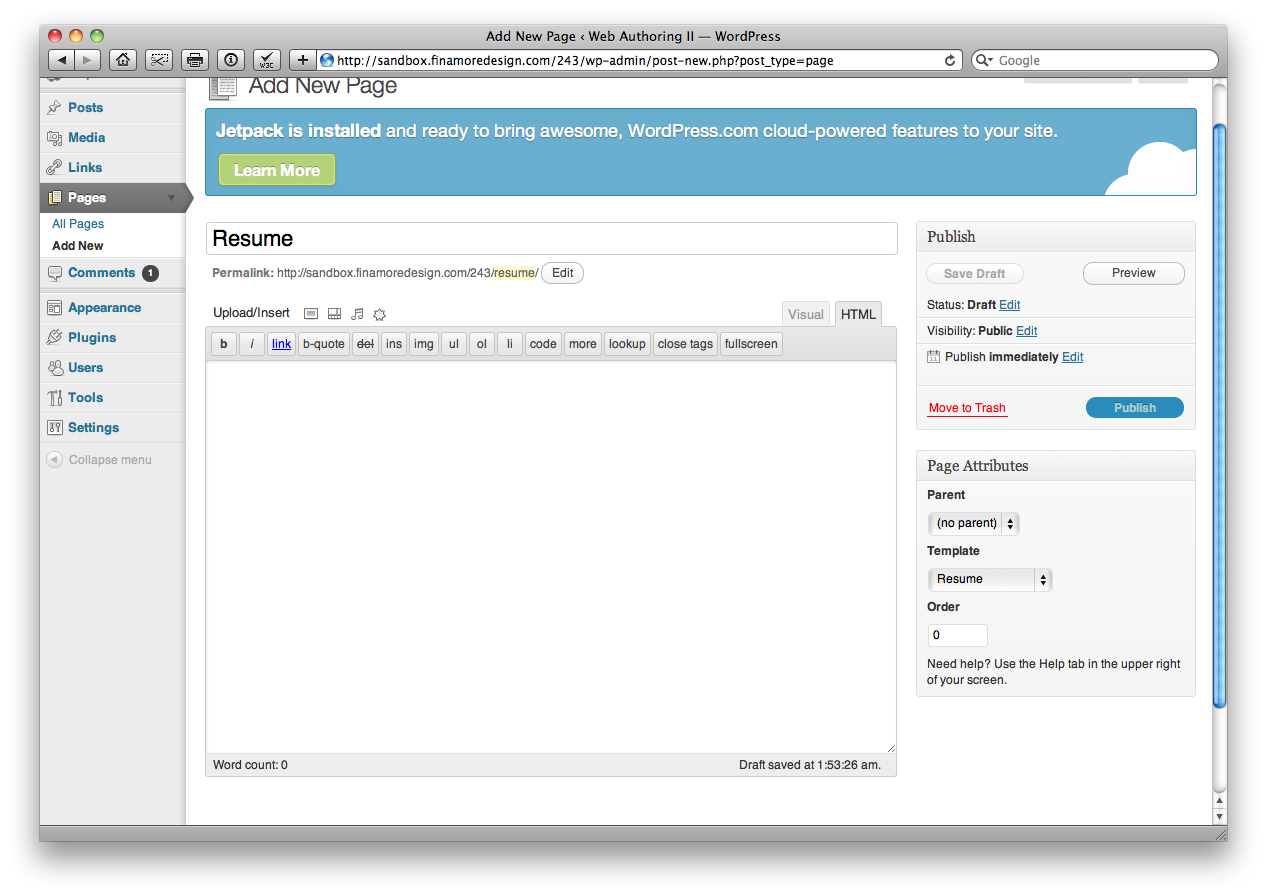
One that you may want to include is the home.php template. When user requests your home page and a static page has not been set in the Reading preferences, WordPress then looks for home.php. If home.php cannot be found it then defaults to index.php. This is very nice because index.php is typically used for things like listing search results or an archive of posts (which can, if you wish, have there own template files, search.php and archive.php respectively) and also acts as a fallback when no other template exists that fits the information WordPress is being asked to display. This is good when you don't want your homepage to be the standard listing of posts, but rather something compelling to draw users deeper into your site.

Most Likely you will have multiple Pages on your site for things like a contact page and your resume page. Obviously, one template would not do the trick of displaying information for both. In this case, you could create a contact.php and resume.php page and then style them accordingly. This also shows the flexibility of pages. Many times you'll find that you want a page on your site that has static information, information that won't change that often or at all (like an about page). Since you can create a template for each specific page, you can create a template that doesn't require calls to WordPress' database and instead hard code this information in the template while taking advantage of features like including the footer, header and a sidebar.

To create a new page template you can start by duplicating page.php, rename it (in this case resume.php) and adding the following php code to the top of the file...

<?php  
/\*  
Template Name: Resume  
\*/  
?>

Now when you create a new page you can choose the Template Name (Resume) from the drop menu in the sidebar.



You can also have a template for dictating how a page that displays a single blog post or portfolio piece will look and what information it will display. For this there is the single.php page. This template will be used by WordPress whenever a user clicks to see a single post/work that you have added to the site. In addition, you can customize how post listings will display pages based on the category or tag of the posts the user wishes to see. For this you have multiple options for creating templates. You can either create a category.php and tag.php which will be used when displaying any category or tag respectively. Another option is to create templates specific to a category or tag with category-X.php and tag-X.php where the X is either the ID assigned to the category/tag or the slug (the simplified code friendly version of the category/tag name, ie a category "Site Updates" would typically have a slug "site-updates" making the specific template would require a file called category-site-updates.php).

There is also the ability to have a custom template for how you want comments to be displayed. comments.php allows you to pull out the code required to include comments on a page and have a separate template file to handle this code. This is useful for the same reason separate header footer and sidebar templates are useful, because you can often have comments in a couple different template files. With this functionality separate, you can change this file and update how they are displayed across the entire site. Also, with this, it allows you to customized where comments are present, if there is a category or tag which you don't want posts to have comments, you can simply leave out the template for those posts with simple conditional statements.

There are a number of additional templates you can use but these are the most commonly used. While all are useful, you should only use those that you feel are really necessary.

**How WordPress Themes Really Work**

Themes work within WordPress using two central concepts, "The Loop", and template tags.

The loop is basically a function that sifts through the database and pulls out the content that is needed to display on the requested page. Learning how to control The Loop is one of the most important aspects of becoming a WordPress theme designer.

Template tags are WordPress specific functions that help you put specific info on your page. Some can be placed outside the loop, but some *have* to be placed inside the loop.

**The Loop**

Understanding and controlling The Loop means having complete control over what content is selected and the order in which it is rendered for display.

The Loop by default is used in your WordPress theme template files. Custom Loops can be created anywhere in your theme template files. Multiple Loops can be used throughout your theme template files. Custom Loops can be created in your header, sidebars, footer, and main content areas of your web site. There is no limit to the number of Loops that can be displayed on your web site.

The Loop uses standard programming conditional statements to determine what and how to display. The first statement in the Loop is an 'if' statement, checking whether any posts exist, because you might not have any posts with the specified category or tag. If content exists, the while statement is used to initiate the Loop and cycle through all posts or pages that need to be displayed. Finally, the\_post() function is called to build the post data, while making it accessible to other WordPress functions. Once the post data has been built, Loop content can be displayed in whatever format you like.

Following is a minimal Loop example. This example features the only required elements for the Loop to function properly:

<?php

if (have\_ posts()) :

while (have\_ posts()) :

the\_ post();

//loop content (template tags, html, etc)

endwhile;

endif;

?>

Remember that this is PHP code, so it needs to be surrounded in <?php and ?> tags.

Let's break down this example to look at the different parts of the Loop:

if (have\_posts()) :

This line checks if any posts or pages are going to be displayed on the current page you are viewing. If posts or pages exist the next line will execute:

while (have\_posts()) :

The preceding while statement starts the Loop, essentially looping through all posts and pages to be displayed on the page until there are no more. The Loop will continue while content exists to be displayed. Once all content has been displayed the while loop will end.

The have\_posts() function simply checks to see if the list of posts being processed is exhausted, or had no entries to begin with.

the\_post();

Next, the the\_post() function is called to load all of the post data. This function **must** be called inside your loop for the post data to be set correctly. This data is assigned to a global variable each time through the Loop iteration.

//loop content

This is where all Loop template tags are placed and any additional code you want displayed inside the Loop. We cover this in more detail later on.

endwhile;

endif;

The endwhile and endif calls end the Loop. Any code placed after these two lines will show at the bottom of your page, after all posts have been displayed. You could also place an else clause to display a message if there is no content to display in the Loop.

**Working Sample of The Loop**

The Loop is usually surrounded by HTML tags in your theme template files. <http://codex.wordpress.org/The_Loop>

The following code shows how the Loop is structured in the H5 Theme as developed by Jeff Starr (http://digwp.com/2009/07/free-html-5-WordPress-theme/):

**<?php if (have\_posts()) : while (have\_posts()) : the\_post(); ?>**

<section>

<article id="post-[<?php the\_ID(); ?>](http://codex.wordpress.org/Function_Reference/the_ID)">

<header>

<h1><a href="[<?php the\_permalink() ?>](http://codex.wordpress.org/Function_Reference/the_permalink)" rel="bookmark" title="Permanent Link to [<?php the\_title\_attribute(); ?>](http://codex.wordpress.org/Function_Reference/the_title_attribute)"><[?php the\_title(); ?>](http://codex.wordpress.org/Function_Reference/the_title)</a></h1>

<p>Posted on [<?php the\_time('F jS, Y'); ?>](http://codex.wordpress.org/Function_Reference/the_time) by [<?php the\_author(); ?>](http://codex.wordpress.org/Function_Reference/the_author)</p>

</header>

<section>

[<?php the\_content('Read more on "'.the\_title('', '', false).'" &raquo;'); ?>](http://codex.wordpress.org/Function_Reference/the_content)

</section>

<footer>

[<?php wp\_link\_pages(array('before' => '<p><strong>Pages:</strong> ', 'after' => '</p>', 'next\_or\_number' => 'number')); ?>](http://codex.wordpress.org/Function_Reference/wp_link_pages)

</footer>

</article>

</section>

**<?php endwhile; else: ?>**

<section>

<article>

<p>Sorry, no posts matched your criteria.</p>

</article>

</section>

**<?php endif; ?>**

An interesting thing about this theme is that it uses as few tags and attributes as possible, and Contains no <div>s, <span>s, classes, or ids. As compared to the Twenty Ten theme loop.php.

Notice how the minimal Loop elements exist, but are surrounded by HTML tags. This is how a normal theme template file will be structured to utilize the Loop. The HTML elements can certainly change, but the Loop elements stay the same. Customizing the style in which content is displayed and choosing post meta data to include in the page composition is done through template tags.

Sometimes you may want to organize your content in a specific way. Say you wanted to have a horizontal row showcasing your three greatest work samples, then three columns where each listed only Animations, Games, and Websites respectively. Using WP\_Query(), rewind\_posts() and query\_posts() you can run through the loop multiple times to pull out specific information.

here is a sample of how this would be done:

<section>

**<?php $featured\_query = new WP\_Query('category\_name=featured&showposts=3');**

**while ($featured\_query->have\_posts()) : $featured\_query->the\_post();**

**$do\_not\_duplicate[] = $post->ID ?>**

<article>

<header>

<h1><a href="<?php the\_permalink(); ?>" title="<?php the\_title(); ?>"><?php the\_title(); ?></a></h1>

</header>

<section>

<?php the\_excerpt(); ?>

</section>

</article>

**<?php endwhile; ?>**

</section>

<section>

<header>

<h1>Animation</h1>

</header>

<nav>

<ul>

**<?php query\_posts('category\_name=animation&showposts=5'); ?>**

**<?php while (have\_posts()) : the\_post();**

**if (in\_array($post->ID, $do\_not\_duplicate)) continue; update\_post\_caches($posts); ?>**

<li>

<h2><a href="<?php the\_permalink(); ?>" title="<?php the\_title(); ?>"><?php the\_title(); ?></a></h2>

<?php the\_excerpt(); ?>

</li>

**<?php endwhile; ?>**

</ul>

</nav>

</section>

<section>

<header>

<h1>Gaming</h1>

<header>

<nav>

<ul>

**<?php rewind\_posts(); ?>**

**<?php query\_posts('category\_name=gaming&showposts=5'); ?>**

**<?php while (have\_posts()) : the\_post();**

**if (in\_array($post->ID, $do\_not\_duplicate)) continue; update\_post\_caches($posts); ?>**

<li>

<h2><a href="<?php the\_permalink(); ?>" title="<?php the\_title(); ?>"><?php the\_title(); ?></a></h2>

<?php the\_excerpt(); ?>

</li>

**<?php endwhile; ?>**

</ul>

</section>

<section>

<header>

<h1>Web Development</h1>

<header>

<nav>

<ul>

**<?php rewind\_posts(); ?>**

**<?php query\_posts('category\_name=webdev&showposts=5'); ?>**

**<?php while (have\_posts()) : the\_post();**

**if (in\_array($post->ID, $do\_not\_duplicate)) continue; update\_post\_caches($posts); ?>**

<li>

<h2><a href="<?php the\_permalink(); ?>" title="<?php the\_title(); ?>"><?php the\_title(); ?></a></h2>

<?php the\_excerpt(); ?>

</li>

**<?php endwhile; ?>**

</ul>

</section>

What is happening here is that you are setting up a custom query to pull out only the first three posts that have been listed with a "Featured" category and we put these into an array named $do\_not\_duplicate.

The loop is like an old school cassette tape. It plays through the database pulling out what it needs then it stops at the end. So once we've got our featured posts, we rewind the loop and query the database again for our animation posts, but the trick here is we check to make sure what we pull is not in our $do\_not\_duplicate array before adding it to the page so that we do not end up with redundant entries on the same page.

Wash. Rinse. Repeat.

**Template Tags**

PHP functions used in your WordPress theme templates to display Loop content are called *template tags.* These tags are used to display specific pieces of data about your web site and content. This allows you to customize how and where content is displayed on your web site.

For example, the the\_title() template tag displays the title of your post or page inside the Loop. The major benefit of using template tags is that you don't need to know PHP code to use them.

Many different template tags are available in WordPress. Some template tags must be inside the Loop, whereas other tags can be used anywhere in your theme template files. Note that in this context, template tags refer to the WordPress functions used to extract post data for display; template files are the theme elements that control how content for a particular content type is displayed. Put another way, template files contain Loops comprising template tags. For an updated list of template tags available in WordPress visit

[http://codex.WordPress.org/Template\_Tags](http://codex.wordpress.org/Template_Tags).

**Commonly Used Template Tags**

There is no shortage of template tags, but typically you will use only a handful of tags in your Loops. Following are the most commonly used template tags available in the Loop. These template tags will return and display the post data listed.

* the\_permalink(): Returns the URL of your post.
* the\_title(): Returns the title of the post.
* the\_ID(): Returns the unique ID of your post.
* the\_content(): Returns the full content of your post.
* the\_excerpt(): Returns just an excerpt of your post. If the Excerpt field is filled out on the Post edit screen, that will be used. If not WordPress will auto-generate a short excerpt from your post content.
* the\_time(): Returns the date/time your post was published.
* the\_author(): Returns the author of the post.
* the\_tags(): Returns the tags attached to the post.
* the\_category(): Returns the categories assigned to the post.
* edit\_post\_link(): Displays an "edit" link that is shown only if you are logged in and allowed to edit the post.

To learn how template tags work, just place any template tag inside the Loop and view the results. The following example views the values of a couple different template tags:

**Using Template Tags**

Template tags are php code and need to be wrapped with <?php ?>.

The most common place you will see template tags being used is the include tags that pull the parts of the page together from the separate template files.

<?php get\_header() ?>

<?php get\_sidebar() ?>

<?php get\_footer() ?>

Most template tags have parameters that can be added to refine the returned value. If you look in the code above you will see that <?php the\_permalink() ?> has no parameters. This simply displays the numeric ID of the current post ( http://codex.WordPress.org/Function\_Reference/the\_ID ). But a little farther down we will see that he uses the\_time tag which has multiple parameters ( http://codex.WordPress.org/Function\_Reference/the\_time )

<?php the\_time('F jS, Y'); ?>

There are a lot of options for how you can display the time on your post ( http://codex.WordPress.org/Formatting\_Date\_and\_Time ).

**Working Outside the Loop**

There are times when you'll want to access generic post information, or manipulate some information about the currently displayed post outside of the Loop. WordPress provides some functions to operate on sets of posts for even finer-grain control over post display.

Along with access to global variables, there are a set of Wordpress functions to return generic information that's not specific to a single post, or the post currently displayed. Following is a list of frequently used outside the Loop functions:

* wp\_list\_pages(): Displays a list of pages as links
* wp\_list\_categories(): Displays a list of categories as links
* wp\_list\_bookmarks(): Displays links saved in the Links SubPanel
* wp\_tag\_cloud(): Displays a tag cloud from all tags
* get\_permalink(): Returns the permalink of a post
* next\_posts\_link(): Link to display previous posts
* previous\_posts\_link(): Link to display next posts

You already saw how you could create navigational links using next\_posts\_link() and previous\_posts\_link() in the custom Loop example. Let's explore some of these functions in action to get a real feel for how they work.

Most menus in a WordPress theme used to be generated using the wp\_list\_pages() function ( http://codex.WordPress.org/Function\_Reference/wp\_list\_pages ). This function will return your pages in a list format, so it's important to wrap the function call with <ul> tags as shown here:

<ul>

<?php wp\_list\_pages('title\_li='); ?>

</ul>

The preceding code would generate a list of pages from WordPress with links. Notice you set the parameter title\_li to nothing, which eliminates the default title displayed for your pages. The function would generate your menu list like so:

<ul>

<li class="page\_item page-item-1">

<a href="http://example.com/about/" title="About">About</a>

</li>

<li class="page\_item page-item-2">

<a href="http://example.com/order/" title="Order">Order</a>

</li>

<li class="page\_item page-item-3">

<a href="http://example.com/contact/" title="Contact">Contact</a>

</li>

</ul>

If you look in the codex you will see that this is a very powerful function. You can show only specific pages, or exclude specific pages. You can sort alphabetically or chronologically, or only show posts by a specific author. There are a lot of other options too.

You could also use the newer wp\_page\_menu() function to generate a page menu ( http://codex.WordPress.org/Function\_Reference/wp\_page\_menu ). There are several advantages to this newer menu function. The first is a new show\_home parameter allowing a Home link to automatically be added to the list of pages. You also don't have to remove the title using title\_li as we did in the preceding code. This function also wraps a custom <div> around your menu, the class of which you can set. An example of this function follows:

<?php wp\_page\_menu('show\_home=1&menu\_class=my-menu&sort\_column=menu\_order'); ?>

**If you are Hook... Then who am I?**

Plugins add functionality via 'Action Hooks.'

<?php wp\_head() ?>

A few Action Hooks do need to be present in your Theme in order for Plug-ins to display information directly in your header, footer, sidebar, or in the page body. l Action Hook Template Tags include:

wp\_head()

Goes in the <head> element of a theme, in header.php.

Example plugin use: add JavaScript code.

wp\_footer()

Goes in footer.php, just before the closing </body> tag.

Example plugin use: insert PHP code that needs to run after everything else, at the bottom of the footer. Very commonly used to insert web statistics code, such as Google Analytics.

wp\_meta()

Typically goes in the <li>Meta</li> section of a Theme's menu or sidebar; sidebar.php template.

Example plugin use: include a rotating advertisement or a tag cloud.

comment\_form()

Goes in comments.php directly before the comment form's closing tag (<tt</form></tt>).

Example plugin use: display a comment preview. As of WordPress 3.0, you should use the default comment form instead, see comment\_form().

dynamic\_sidebar()

This function goes in the sidebar and calls each of the active widget callbacks in order, which prints the markup for the sidebar.

**Theme Frameworks**

Back in week2 I gave you a list of common frameworks. Using a framework is not cheating. It is what the pros do. Every developer worth their salt know that starting from scratch every time is a tremendous waste of time. Frameworks were developed to get you us an running as quickly as possible. They provide the foundational markup, functional elements, and often some CSS.

**Child Themes**

Once you have chosen your framework and have uploaded it to your wp-content/themes/ folder you will create another folder for your child theme. At minimum all this needs is a style.css with a special header that defines the Parent Theme.

Here is an example information header of a child theme’s style.css:

/\*

Theme Name: Whiteboard Child

Theme URI: http://sandbox.westphal.drexel.edu/~abc123/

Description: Child theme for the Whiteboard theme

Author: Abra Cadabra

Author URI: http://sandbox.westphal.drexel.edu/~abc123/about/

Template: whiteboard

Version: 0.1.0

\*/

A quick explanation of each line:

• Theme Name. (required) Child theme name.

• Theme URI. (optional) Child theme webpage.

• Description. (optional) What this theme is. E.g.: My first child theme. Hurrah!

• Author URI. (optional) Author webpage.

• Author. (optional) Author name.

• Template. (required) directory name of parent theme, case-sensitive.

NOTE. You have to switch to a different theme and back to the child theme when you modify this line.

• Version. (optional) Child theme version. E.g.: 0.1, 1.0, etc.

The part after the closing \*/ of the header works as a regular stylesheet file. It is where you put the styling rules you want WordPress to apply.

This will apply all the functionality of the Parent theme to your site, but none of the CSS. If you want to include the CSS from the parent you can use the @import rule:

@import url("../whiteboard/style.css");

Then anything that you write in your Child's style.css can overwrite styles reflected in the Parent.

Unlike style.css, the functions.php of a child theme does not override its counterpart from the parent. Instead, it is loaded in addition to the parent’s functions.php. (Specifically, it is loaded right before the parent’s file.)

Conversely, any template file that you add to your Child theme that shares the same name as a template file in the Parent Theme will override the parent theme

Once you have your Parent and Child Theme installed it is a good idea to navigate a few different types of pages (Home, Page, Post, Archive) in your browser and view the source code so you can get an idea of the structure and the id and class names that are used so that you can start developing your Child Theme.

---

**Homework:**

**Build:**

Update your WordPress content:

* Update your WP website with at least three meaningful post.

Based midterm critiques and on what you have learned thus far.

* Find, download, and install a WP Theme that closely matches your design concept.
* Research what changes you will need to make to said theme to get it to exactly match your final design concept.
  + List what changes you believe you will be able to make in the Dashboard.
  + List what changes you believe you will be able to make in CSS.
  + List what changes you believe you will have to make with PHP.

**Submit:**

URL to new WordPress installation that you will be using for this class.

*http://www.domain.com/wordpressfoldername/*

URL to the source WP theme you will be using

Itemized description of:

* List what changes you believe you will be able to make in the Dashboard.
* List what changes you believe you will be able to make in CSS.
* List what changes you believe you will have to make with PHP.