MMP350 Class Notes Week 6

This Week's goals

- Review the WordPress Dashboard
- PHP overview
- Review of WordPress Architecture
    - File system
    - Template hierarchy
- Your first WordPress theme
- Review of Fifteen Common WordPress Mistakes, if there is time.

Homework

Read the following article on the WordPress template hierarchy and summarize it in your own words.

http://codex.wordpress.org/Template_Hierarchy

Note that there has been a slight change, relative to the syllabus. The assignment to create your own customized template will be *next week's homework*, not this week's homework.

What is WordPress?

WordPress was created as a blogging platform and has become a powerful backend for many sites that are content based. It is a:

Content Management System – WordPress has an online interface for users with little or not knowledge of HTML or code to post content.

WordPress is open source, it can be downloaded, installed and used for free.

Dashboard – This is the client side of WordPress where content is created and uploaded to a server.

Themes – Visual design of the site is determined by themes, which are applied to content from blog posts, pages and widgets.

WordPress pages are populated by taking code from many sources, which allows them to be dynamic and host many different kinds of content.

WordPress.com – Free, hosted on WordPress server, limited customization. We'll start with a WordPress.com blog today.

WordPress.org – Free, hosted on your own server, completely customizable.

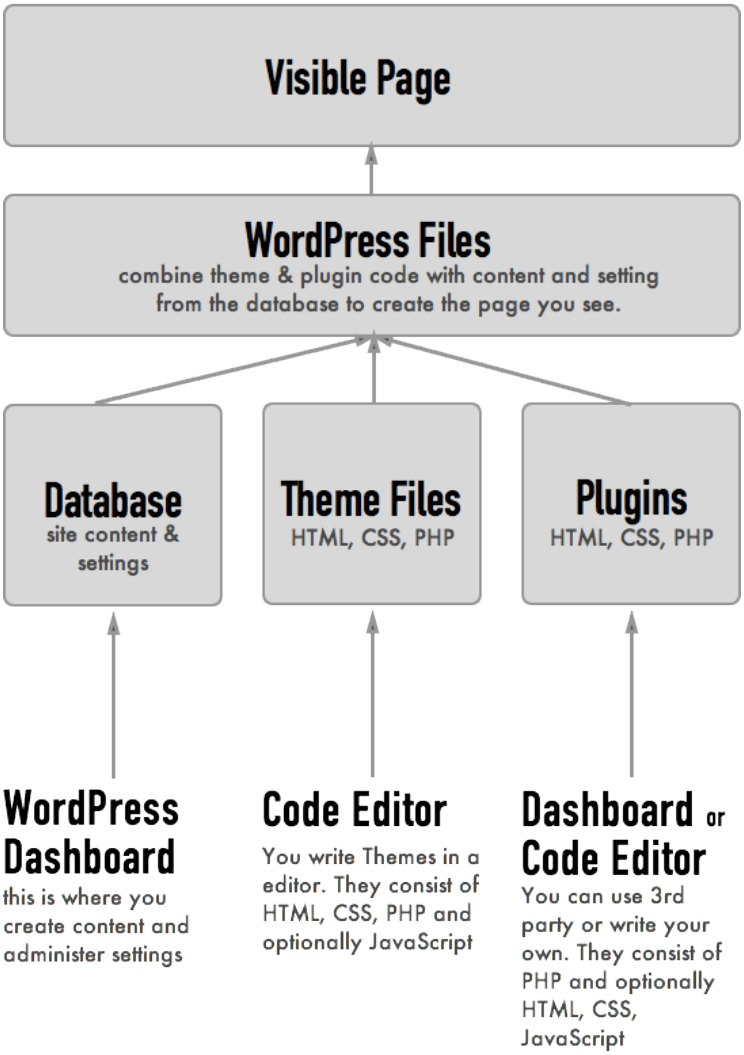


Figure: Wordpress components

The WordPress Dashboard

Now we're going to review some basics of the WordPress Dashboard.

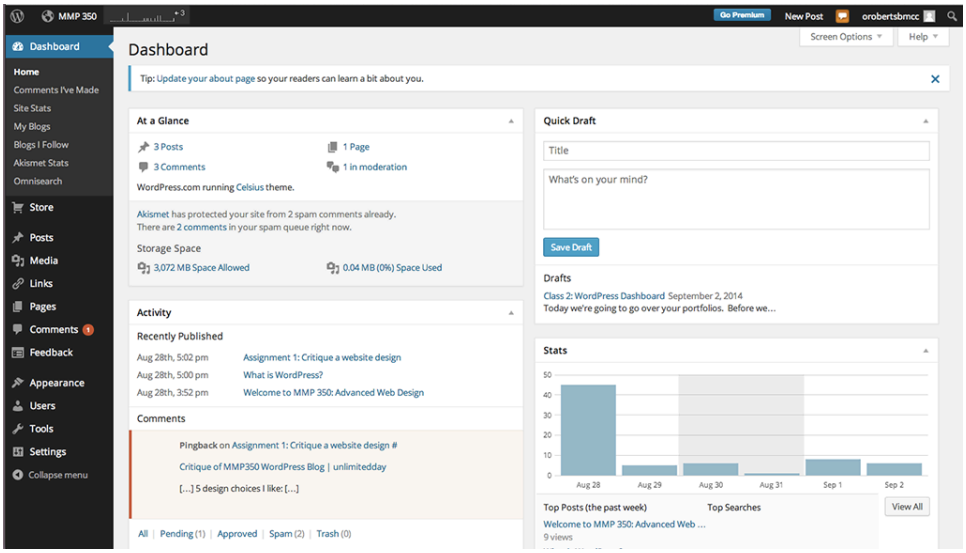


Figure: The WordPress Dashboard

On the WordPress Dashboard you will find everything you need to post new pages, blog posts and edit the look and structure of your site.

Your portfolio site will include pages for navigation, basic information and organization as well as posts for individual projects.

Pages

Pages are static individual pages that usually contain navigation to other groups of pages. Let's quickly create an About page, and edit the home page using the Page editor. Note that the Page editor is very similar to the Post editor (which we would look at later), but Pages will not be included in the blog feed.

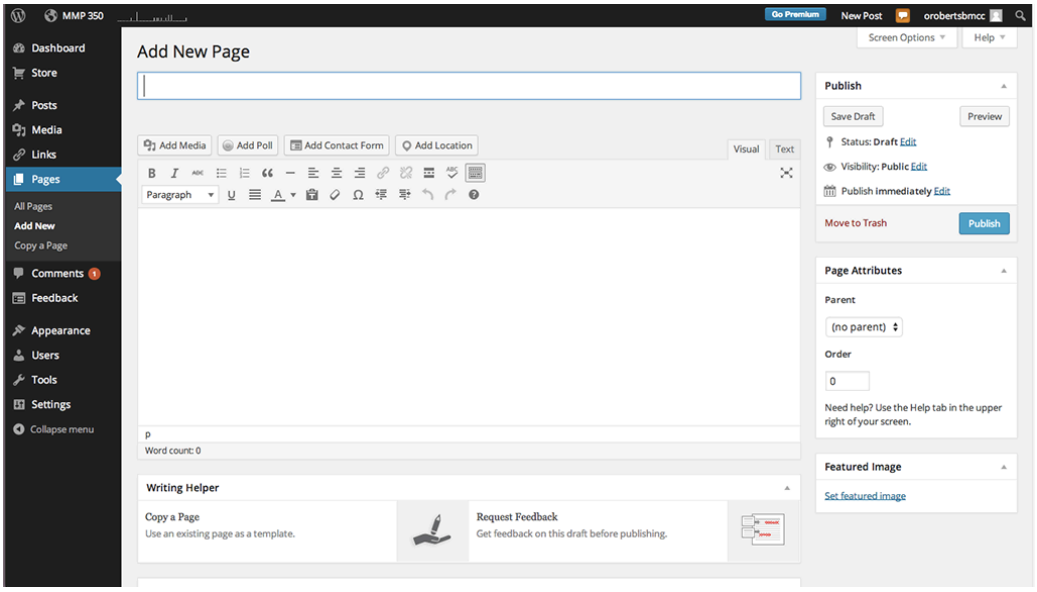


Figure: Add New Page

There are two ways to create a new Page from the Dashboard:

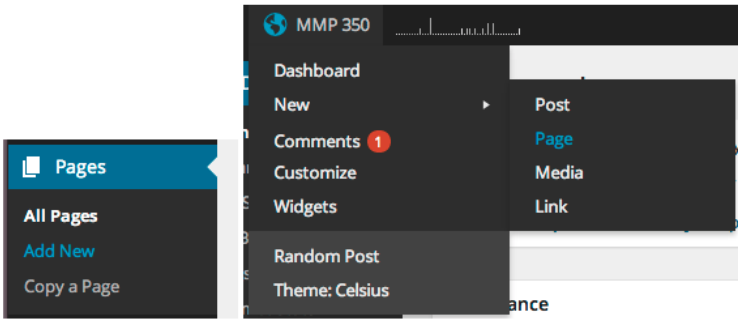


Figure: Two ways to add a page to WordPress, from the Pages menu item or by pressing the + New option and selecting Page.

Your site can have a number of pages including Home, About, Blog, and links to specific posts.

Posts and Pages

Posts and pages are created in similar ways but serve different functions. *Pages organize content, while posts contain the actual content*. You can add projects you've worked on to posts and organize them with categories and tags. Those posts will then appear in your feed. The interface for creating and editing posts is the same as that for Pages, with a couple of important extra functions.

Further exploration: The following article has an excellent example that illustrates the relationship between posts and pages:

https://www.webmechanix.com/how-to-add-posts-to-pages-in-wordpress-tutorial/#guide

Categories

Categories are a useful way to organize content. You might have categories like: Web Design, Interactive, Fine Art, Game Design, or whatever other kinds of work you want to show on your portfolio.

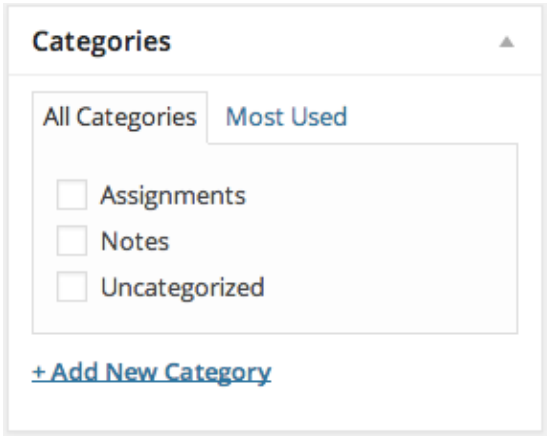


Figure: Categories

Tags

Tags are another way to organize content. You might add tags like HTML, CSS, JavaScript, PHP, Design, Graphics, Motion, BMCC, or other tags to describe the content in your posts. This will make it easy for readers of your portfolio to see what kind of work you have done.

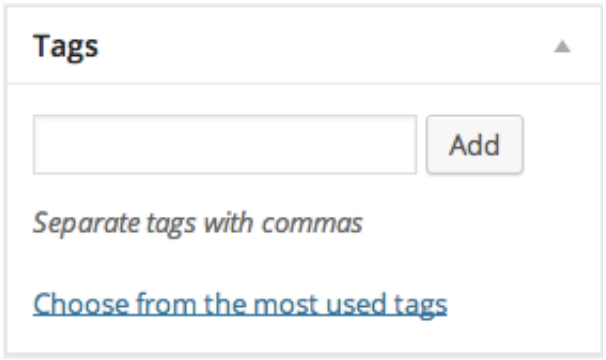


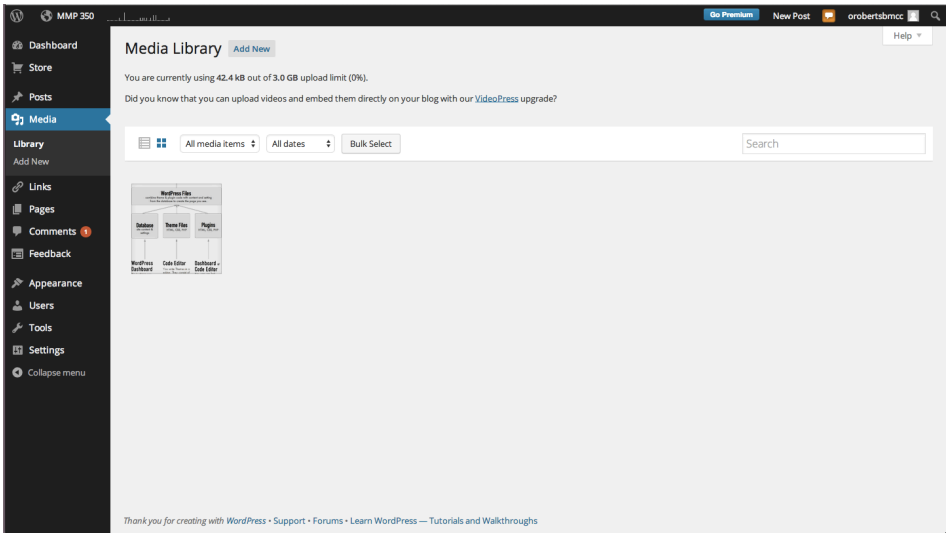
Figure: Tags

Discussion

What is the difference between categories and tags?

Media Library

The WordPress media library will store all of the images, videos and sound you upload to your blog. You can upload new media either by using the link to the Media Library in the Dashboard, or by simply dragging and dropping images and other media directly onto your blog post.



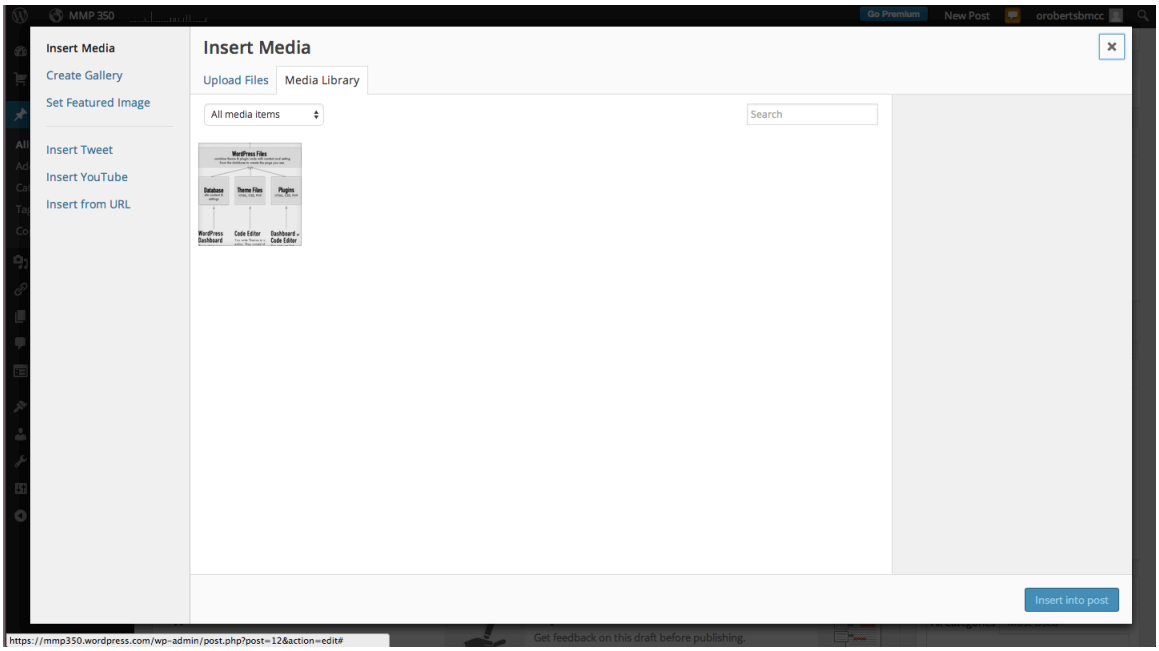


Figure: Media

General Settings

Take a moment to look at the general settings for you blog. You can change the site title and the formatting for displaying information like the date and time of posts.

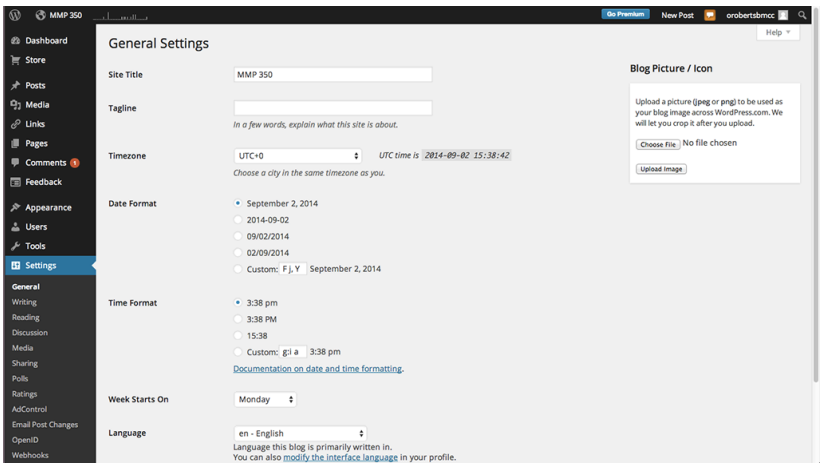


Figure: General Settings

Reading Settings

The reading settings contain organization that the blog reader or user will see, like the number of posts.

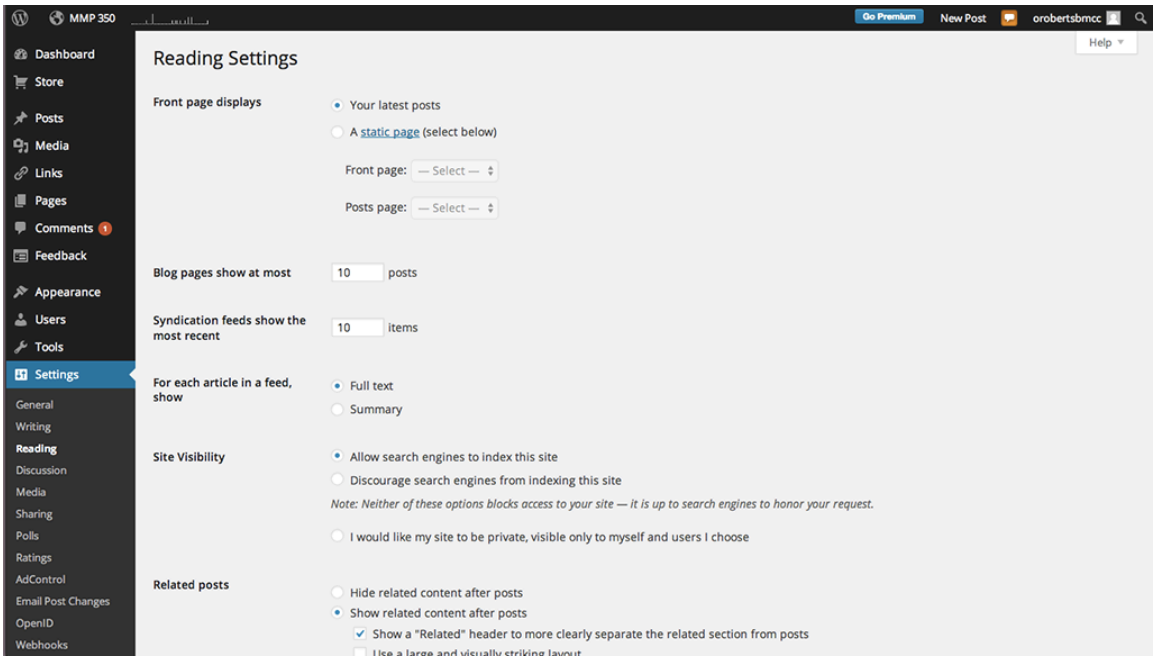
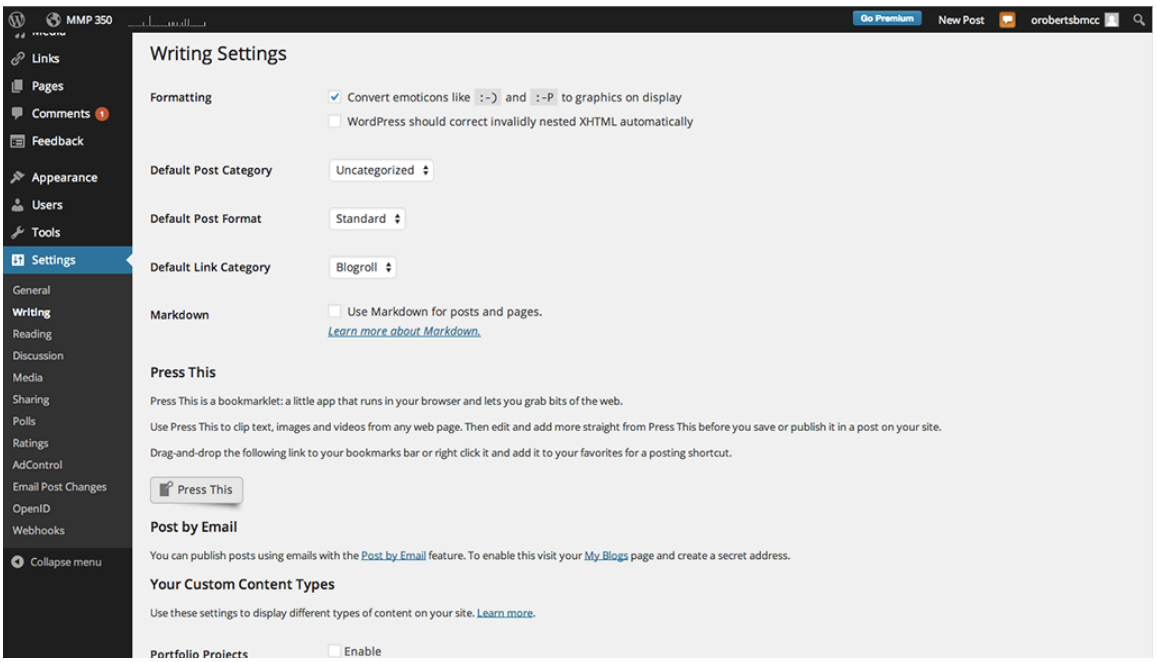


Figure: Reading Settings

Writing Settings

Writing settings will affect the CMS and has some short cuts you can use for your own blog posts.



Php Basics

Before we create our first theme, let us review some php basics.

What is PHP?

PHP is a hypertext preprocessor. In practical terms, that means it is a server-side scripting language that is used to process http requests (typically to a fileserver or database server) and create html code.

Helloworld

Let's examine a very simple php file. To start, go to:

http://php.net/manual/en/tutorial.firstpage.php

… and create the file helloworld.php

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo '<p>Hello World</p>'; ?>
 </body>
</html>
```

Try to view it in a browser. It doesn't work.

Now click on this link to my webserver:

http://45.55.178.198/BMCC/hello.php

PHP needs to be served by a webserver like apache. It will not automatically display in a browser.

Let's look at the syntax of a second php file:

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
    <?php phpinfo(); ?>
 </body>
</html>
```

What do you notice about this file?

Now let's look at a more complicated php file, getDemographicsJSON.php, which was mailed to you.

```
<?php
    if(session_id() == '') {session_start();}

    $SCRIPT_NAME="getDemographicsJSON.php";
    #override until path handling finalized
    $root = "";
    try
    {
        include_once($root."config/config_db.php");  // login information - connectivity
        and message constants.
        include_once($root."clsDatabase.php");  // wrapper for database connectivity

        $dbConnect = new DatabaseAPI();

        $_SESSION['host']=$HOST;
        $_SESSION['username']=$USERNAME;
        $_SESSION['password']=$PASSWORD;
        $_SESSION['dbName']=$DBNAME;

        $SUCCESS_CODE=0;
        $SUCCESS_MESSAGE="Success";

        $NO_DATA_CODE=2;
        $NO_DATA_MESSAGE="No demographic data";

        $ERROR_CODE=3;
        $ERROR_MESSAGE="Error in ".$SCRIPT_NAME;

        $sResultsArr="";
```

Figure: A code fragment from getDemographicsJSON.php

Summary

1. PHP requires a server like Apache in order to be executed.
2. PHP can be embedded in html. This embedding is called a server-side-include, because it executes code on the server.
3. Variables begin with $
4. All lines end with a semi-colon
5. PHP files / statements begin with **<?php** and end with **?>**
6. **=** is the assignment operator
7. **==** is the string comparison operator
8. **.** (period) is the string concatenation operator
9. php variables can be concatenated within a strong, for example
   sFullname = "$fname . $lname"

Building a WordPress Theme: index.php

We're going to start building out the basics of our WordPress theme starting with a template with two files: **index.php** and **style.css**. This is the minimum requirement for a WordPress theme to be installed.

To get started unzip **myfirsttheme.zip**.

http://wpmmp.bmcc.cuny.edu/~bmacmill/classes/themestarter.zip

Note that index.php is the base file for the folder, as opposed to index.html. We'll add a few things to **style.css** and then test it on our WP installations.

Let's start with the `<head>` section of index.php; it is at the beginning of the document.

index.html

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6      <title>Theme Starter Files</title>
7      <link rel="stylesheet" href="style.css">
8  </head>
```

Figure: the head section

Most of this will stay the same, but we need to change two lines.

First is the `<title>`. WordPress themes should be customizable, so we will begin replacing default or hard coded content with php variables that are provided by WordPress.

```
6     <title><?php bloginfo('name');?> | <?php bloginfo('description')?></title>
```

Figure: Inserting WordPress functions into the <title> element.

This is the code you will be adding:

```
<?php bloginfo('name'); ?> | <?php bloginfo('description')?>
```

The first php tag uses the bloginfo() function to retrieve the blog name. The second line uses the same function to retrieve the blog description, which is a reference to the tagline from the Dashboard settings

By using php variables, we can get those pieces of information, filled in by the user or by us, at any part of our website.

Next change the CSS link tag:

```
<link rel="stylesheet" type="text/css" href="<?php bloginfo('stylesheet_url'); ?>">
```

Figure: Adding a php function to a <link> element

This is the code you will be adding:

```
<link rel="stylesheet" href="<?php bloginfo('stylesheet_url'); ?>">
```

Again we're using the `bloginfo()` function, this time to get the URL location of the stylesheet. Because WP uses a more complex file structure than a static website, it is a good idea to use this references instead of a direct URL to the style sheet.

Note that php can be used inside of the quotes of html attributes, like href.

Finally, add this tag inside the head section:

```
<?php wp_head(); ?>
```

Figure: Adding a function in the head section.

This is the code you will be adding:

```
<?php wp_head(); ?>
```

This is going to load some WordPress functionality into the head of the website. We're going to skip the `<header>` and `<nav>` for now, we'll cover those in depth in upcoming lessons.

Skip ahead to this section:

```
25          <div class="posts">
26              <article class="post">
27                  <header>
28                      <h2 class="entry-title"><a href="permalink_to_title">The Title of
                        the Post</a></h2>
29
30
31                  </header>
32
33                  <div class="content">
34                      <p>
35                          the_content() of the post goes here
36                      </p>
```

We're going to add a few lines and make some changes here.

First add this line, called the WP Loop. We'll cover this in depth soon. The important thing to know now is that WP Loop does most of the work of going through all of your posts or pages and grabbing the content from the database.

```
25          <div class="posts">
26
27              <?php if(have_posts()) : while(have_posts()) : the_post(); ?>
28
29              <article class="post">
```

This is the code you will be adding:

```
<?php if(have_posts()) : while(have_posts()) : the_post(); ?>
```

Next, replace all of `<a href="permalink_to_title">` The Title of the Post `</a>` with the following:

```
<h2 class="entry-title">
    <a href="<?php the_permalink(); ?>">
        <?php the_title(); ?>
    </a>
</h2>
```

This is the code you will be adding:

```
<h2 class="entry-title">
      <a href="<?php the_permalink();?> ">
            <? the_title(); ?>
      </a>
</h2>
```

These lines are using functions to get the link to each post and the title of the post. You should start to see a pattern emerging.

Next, replace the entire section `<div id="content"> ... </div>` with this line:

```
<?php the_content(); ?>
```

Next replace the line `<p class="entry-meta">Posted on February 21, 2005 by <span class="author">author</span></p>` with this:

```
<p class="entry-meta">Posted on <?php the_date(); ?> by
<span class="author"><?php the_author(); ?></span>
</p>
```

Figure: adding the author of the website and date of each post.

This is the code you will be adding:

```
<p class="entry-meta">Posted on <?php the_date(); ?> by

      <span class="author"><?php the_author(); ?></span>

</p>
```

Finally, delete the entire second `<article>`. With WordPress working, we only have to make a single template for each post, which will be populated with new content.

Then, after the `<article>` section, before the end of the `<div id="content">` section, add the following:

```
<?php endwhile; else: ?>
    <p><?php _e('Sorry, no posts matched your criteria.'); ?></p>
<?php endif; ?> <!--End of posts -->
```

Figure: Exception handling while loading blog posts.

This tells WordPress what to do once it's loaded all of the blog posts or there are no blog posts to load.

Let's change a couple of things in the `<footer>`:

```
<footer>
    <p class="copyright">&copy; 2014</p>
    <p class="webdesigner">website by Your Name</p>
</footer>
```

Add the current the_date() and the_author () functions to the footer:

```
<footer>
    <p class="copyright">&copy; <?php the_date(Y);?></p>
    <p class="webdesigner">website by <?php the_author(); ?></p>
</footer>
```

Figure: the_date() and the_author () functions

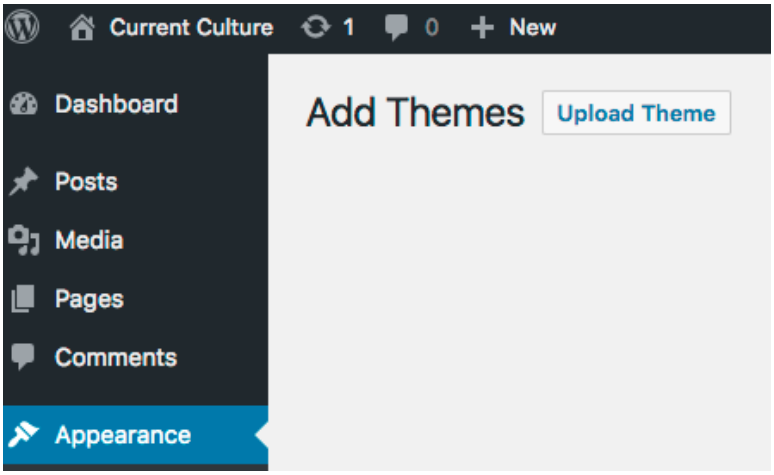
That's it for **index.php** for now.

Let's edit **style.css** and then test our theme.

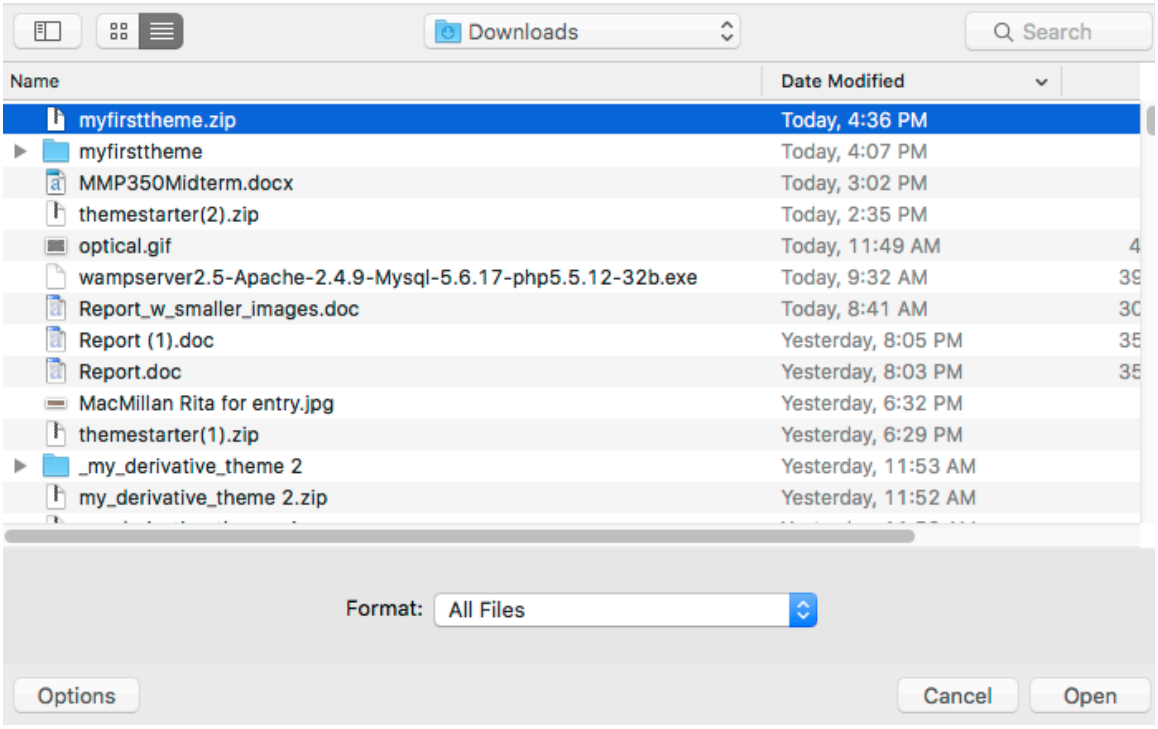
```
/*
Theme Name: My First Theme
Author: Brian MacMillan
Description: A basic starter theme for MMP350 portfolio project.
*/
```

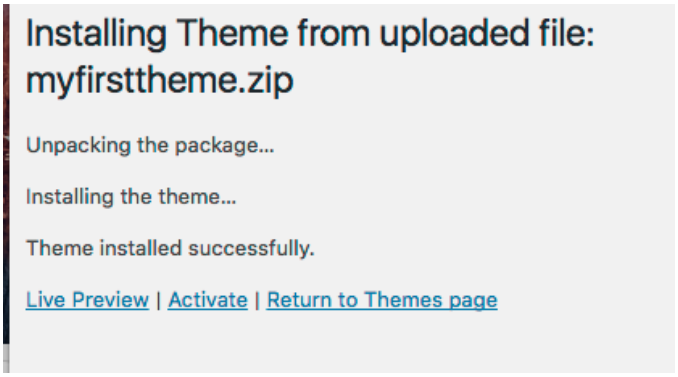
Right now this is all you will see in **style.css**. In order for WordPress to recognize the theme, it needs to have this comment with this basic information. Go ahead and replace the info for **Theme Name**, **Author** and **Description** with your own information.

Now let's upload these themes and test them out. If this is the first time uploading the theme, then turn its folder structure into a zip file and upload it. Note that in this example I have created a zip file called myfirsttheme.zip by highlighting its parent folder and compressing it.

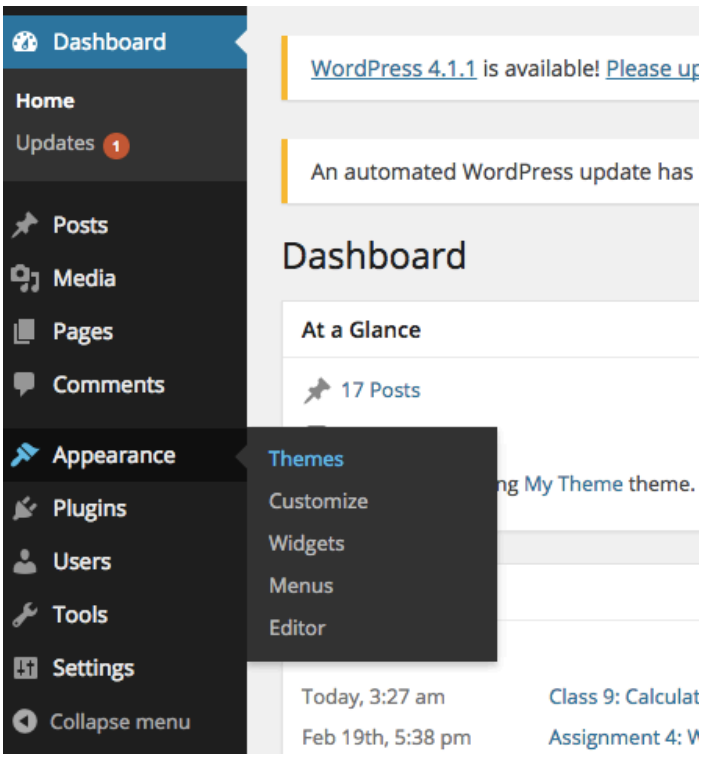


Select Upload Theme

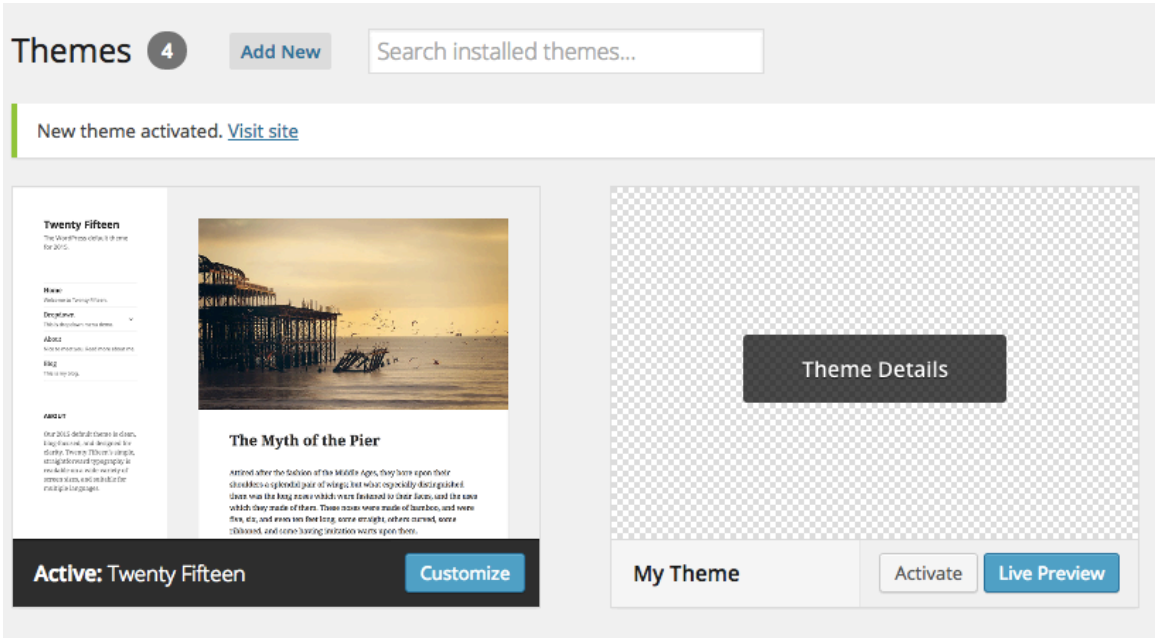




Your theme should just consist of **index.php** and **style.css**. Once it's uploaded, you should the theme in the WordPress Dashboard in **Appearance > Themes**.



From here, click **Activate** to see your theme in action:



The WordPress Loop and front-page.php

Now we're going to go over the WordPress Loop, which is used to populate most of the content on our pages and posts, and creating a front page for the portfolio sites.

Exercise file: wp_loop.zip

The Loop

The WordPress loop is the process by which WordPress compiles pages from its component parts.

Here's the WordPress explanation of the Wordpress Loop.

The WordPress loop iterates through available content and makes it available to the template code. This makes it easy to customize the content we create in the WordPress CMS to fit our theme and design.

Loop template tags

There are many template tags that can be used only within the WordPress loop. A few tags that may be useful on our blogs:

```
<?php the_excerpt(); ?>

<?php the_content(); ?>
```

The excerpt is a short line or description of the post. If there is no excerpt, WP fills in some text from the beginning of your post.

the_content() is the content of the post/page, so any text, images or other media that are part of the body of an entry.

```
<div class="entry-content">
  <?php the_content(); ?>
</div>
```

The content will typically go into a div or article.

```
<?php the_ID(); ?>
<?php post_class(); ?>

<article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
```

IDs and classes can be set for the post, which can be used for CSS styling or JavaScript functionality.

```
<?php the_date(); ?>
<?php the_date('m-d-Y'); ?>
```

The date the post was written is added with **the_date()**. An argument can be added for format.

```
<?php the_author_posts_link(); ?>

<p class="entry-meta">Posted on Posted on <?php the_date(); ?>
  by <span class="author"><?php the_author_link(); ?></span>
</p>
```

the_author_posts_link() adds a link to all posts by the author. Your site will most likely have one author, but this will be useful for future projects or as an option on your theme.

```php
<?php the_tags(); ?>

<?php the_tags( $before, $sep, $after ); ?>

<?php the_tags('Tags: ', ', ', '<br />'); ?>

<?php the_tags('Tagged with: ', ' • ','<br />'); ?>
```

Tags are a great way to organize content. WP helps style the tag with arguments to determine how they are separated with HTML and text.

```php
<?php the_category(); ?>
<p>Categories: <?php the_category(', '); ?></p>
<p>Categories: <?php the_category(' &bull; '); ?></p>
```

Categories can also take an argument for how they are separated.

```php
<?php the_post_thumbnail(); ?>
```

the_thumbnail() will show the featured image tagged in the post/page.

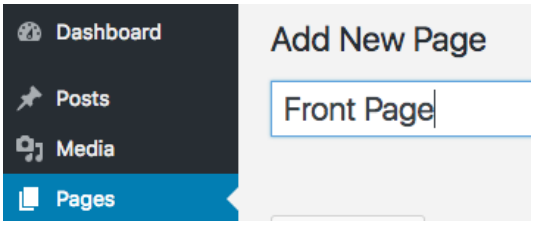
```php
<?php previous_post_link('%link', '%title', TRUE ); ?>

<?php next_post_link('%link', '%title', TRUE ); ?>
```

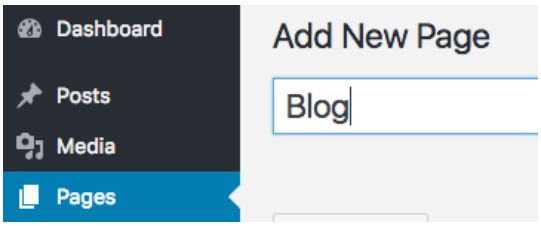
Links to next/last post. Third argument is set TRUE to only let posts with same category, FALSE for another post/page to come up.

Okay, now that we have some new WP toys to play with, let's build a front page. Most of your portfolio designs require a homepage, either with a large image, some information or other design elements. Currently our sites all open up to blog posts. We can use **front-page.php** to create a separate page to appear at the index of the site.

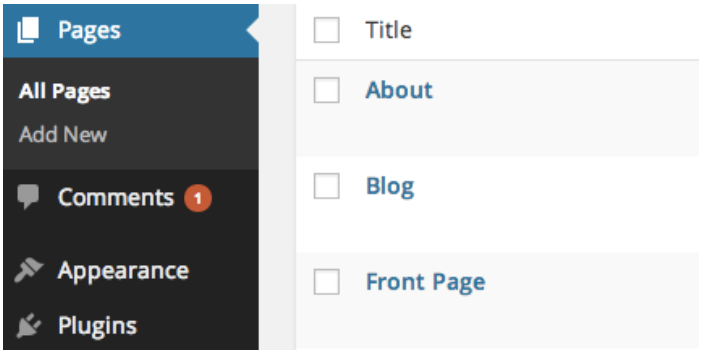
Let's look quickly at the WP template hierarchy to understand how WP decides what page to load at the root of the site.

First, create a page called **Front Page** and a Page called **Blog** from the Dashboard.

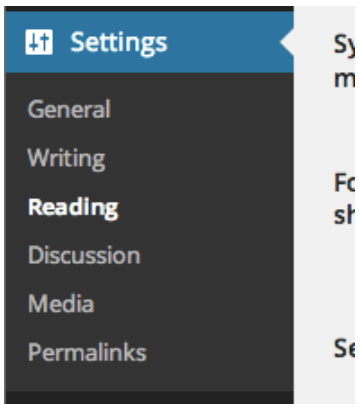


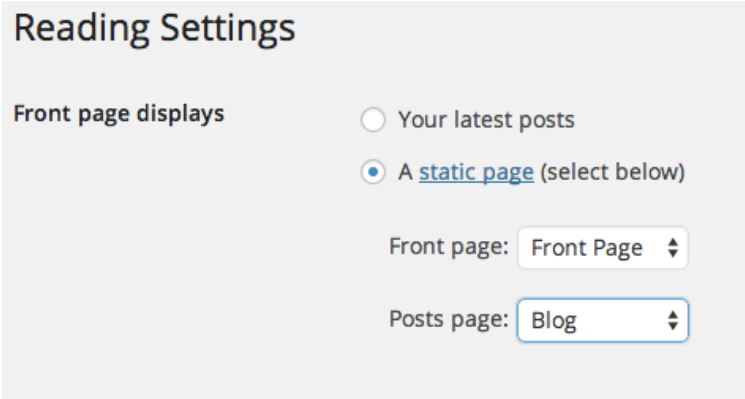


Front Page will be the splash page of the portfolio, and blog will be the new page to display posts. We can leave them blank at first.

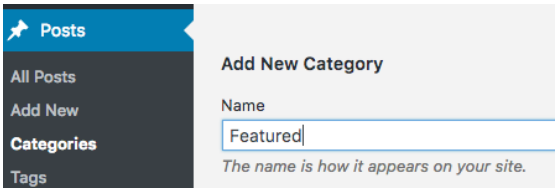


Next go to Settings > Reading and set the front page and posts page to Front Page and Blog.



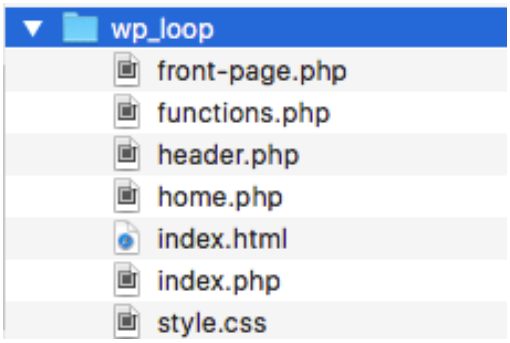


Now we need to add Features posts to appear on the front page. Add a new Category called Featured. In All Posts, Quick Edit a couple of posts and add "Featured".



Now for some PHP fun.

Go to your local WP files. Notice that there are a number of new files, header.php, functions.php, front-page.php and home.php.



Let's talk about header.php first. Header.php is a generic header that can be reused by all of your pages. It is called by the function get_header() which you will see that the top of index.php, front-page.php and home.php.

```
<?php get_header(); ?>
```

```php
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title><?php bloginfo('name'); ?> | <?php bloginfo('description'); ?></title>
    <link href="<?php bloginfo('stylesheet_url'); ?>" rel="stylesheet" type="text/css">
    <?php wp_head(); ?>
</head>
<body>
    <div class="wrapper">
        <header>
            <h1><a href="<?php echo home_url('/'); ?>"><?php bloginfo('name'); ?></a></h1>
            <h2><?php bloginfo('description'); ?></h2>

            <?php $main_menu_top = array(
                    'theme_location' => 'main-menu',
                    'container' => 'nav'
                );
            ?>
            <?php wp_nav_menu($main_menu_top); ?>

        </header>
```

Figure: the contents of header.php

Q: What do you notice about header.php?

Step 1

Front-page.php and home.php are clones of index.php Both files will use the loop to display different content. They are activated in different contexts.

Let's edit **front-page.php** first.

For the front page we're not going to use the title for the post, just the content.

```php
<div class="front-page">

    <?php if(have_posts()) : while(have_posts()) : the_post(); ?>

    <div class="intro">
        <?php the_content(); ?>
    </div>

    <?php endwhile; else: ?>

    <p>
        <?php _e('Sorry, no posts matched your criteria.'); ?>
    </p>
    <?php endif; ?> <!--End of posts -->
</div> <!-- end content -->
```

I want to make sure only the Featured posts appear on the front page, so I'm going to make a custom query using **WP_Query()**. I'm going to create an argument for the query as a variable and save the result in another variable:

Step 2

```
<?php
    $args = array('category_name' => 'featured');
    $featured = new WP_Query($args);
?>
```

and feed that variable into my content loop:

```
<?php if(have_posts()) : while($featured->have_posts()) : $featured->the_post(); ?>
```

Finally, I'm going to take out the **else** statement. If there are no Featured posts, well, I'm going to have to make some to fulfill my design. I also need to add the function **wp_reset_postdata()** to reset my query for later loops.

```
    <?php endwhile;  ?>
    <?php endif; wp_reset_postdata(); ?> <!--End of posts -->
</div> <!-- end content -->
```

Step 3: Displaying the Featured Image as a Thumbnail

Okay, this still looks like a blog. Let's change the template a bit to make it look like a portfolio. We'll need to add some basic styles and change the content a bit.

First, let's set the featured image with a single line:

```
add_theme_support( 'post-thumbnails' );
```

Figure: add theme support for featured image thumbnails to functions.php

Step 4: Modifying home.php for blog posts

Now modify home.php so that it looks like the following:

```
<?php get_header(); ?>
        <div class="posts">
            <?php if(have_posts()) : while(have_posts()) : the_post(); ?>
            <article class="post">
                <header>
                    <h2 class="entry-title"><a href="<?php the_permalink(); ?>">
                    <?php the_title(); ?></a></h2>
                </header>
                <div class="content">
                    <?php the_content(); ?>
                </div>
                <p class="entry-meta">Posted on <?php the_date(); ?> by <span
                class="author"><?php the_author(); ?></span></p>
            </article>
            <?php endwhile; else: ?>
                <p>No posts are found.</p>
            <?php endif; ?>
        </div> <!-- end content -->
        <footer>
            <p class="copyright">&copy; <?php the_date(); ?></p>
            <p class="webdesigner">website by <?php the_author(); ?></p>
        </footer>
    </div> <!-- end wrapper -->
</body>
</html>
```

Extra work if there is time

The files for this exercise are in category_and_sidebar.zip

Footer.php

The file **footer.php** is accessed from **front-page.php** and other pages using:

```
<?php get_footer(); ?>
```

Inside the **footer.php** folder we added a new menu for the footer links. To do this we had to register two menus inside of **functions.php**:

```
register_nav_menus( array (
                    "main-menu" => "Main Menu",
                    "footer-menu" => "Footer Menu"
            )
);
```

And then used the same loop as the Main Menu in **header.php** with some modifications:

```
<nav class="small-12 large-3 columns">
 <?php $footer_menu = array(
  'theme_location' => 'footer-menu'
  );
 ?>
 <?php wp_nav_menu($footer_menu); ?>
</nav>
```

7

We also used three different instances of the **WP Loop**, using different queries to populate different sections of the prototype. We introduced new arguments for the query, such as `posts_per_page`.

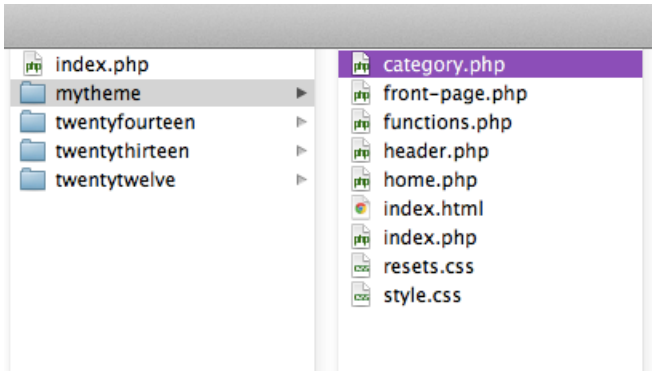
```
<?php
  $args = array(
    "category_name" => "featured",
    "posts_per_page" => 3
  );
  $featured = new WP_Query($args);
?>
```

```
<?php if(have_posts()) : while($featured->have_posts()) : $featured->the_post();
  <div class="footer-link small-12 large-3 columns">
    <?php the_post_thumbnail(); ?>
  </div>
<?php endwhile; ?>
<?php endif; wp_reset_postdata(); ?>
```

More WordPress – single.php, category.php, category.php, page.php, sidebar.php

We're going to go over **single.php**, **category.php, page.php** and **sidebar.php**.

category.php can be used like **front-page.php,** to create an overview of posts, so we'll start there. This time, duplicate **front-page.php** and change the name to **category.php**.



You can use categories to organize your portfolio into different sections, depending on medium, discipline or other factors.

To show which category is being viewed, use the `single_cat_title()` function:

```
<?php echo single_cat_title(); ?>
```

This will print the name of the category using the `echo` function, which outputs a given string. If we want to include more information about the category, we can create conditional statements to determine what category is being called and add contextual content.

```
<?php if (is_category('Web')) : ?>
  <h2>Web Design</h2>
  <p>A sample of previous web design work.</p>
<?php elseif (is_category('Photography')) : ?>
  <h2>Photography</h2>
  <p>View some of my recent photographs.</p>
<?php endif; ?>
```

This is the first time we have seen the `elseif` conditional. This is used to evaluate a series of conditional statements.

For the WP loop, we no longer need the query for featured posts that was being used in **front-page.php** (unless you have another set of featured posts you want to use for each category). So we can revert to the original WordPress loop:

```
<?php if (have_posts()) : while(have_posts()) : the_post(); ?>
```

Those are the primary changes to make. You may need to add or remove HTML content from the loop if you plan to change the style of the category page.

single.php and **page.php**.

Let's take a look at the WordPress Template Hierarchy again and talk about the difference between these two templates. Okay, so **single.php** will be used for a single blog post, while **page.php** will be used for a static page. This time I'm going to duplicate **index.php** again, because I want see the full content of the pages and posts. Depending on how your content is organized, you may not need both of these options, or they might be exactly the same, or they might be different.

So these template files will look almost exactly the same as **index.php** except I want to add custom classes so make it easier to change the style.

In **index.php** I had this after `<?php get_header(); ?>:`

```
<div class="posts">
  <?php if(have_posts()) : while(have_posts()) : the_post(); ?>
  <article class="post">
```

For **single.php** I'll change it to:

```
<div class="single-post">
  <?php if(have_posts()) : while(have_posts()) : the_post(); ?>
  <article class="post">
```

And for **page.php**:

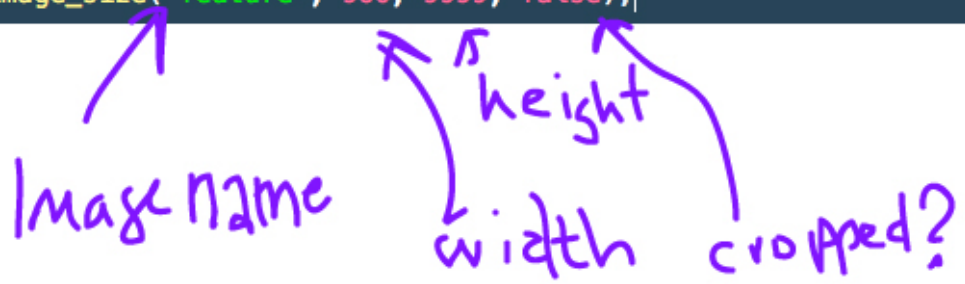
```
<div class="page">
  <?php if(have_posts()) : while(have_posts()) : the_post(); ?>
  <article class="post">
```

The only difference is the CSS classes.

Notice that the posts section is the same in **page.php** and **post.php**. This is confusing, but although pages and posts use different formats, the information is still accessed via the WordPress loop, so the word **post** is still used to describe the content in a page.

For **page.php**, I want to add a featured image. Unlike the thumbnail for the category pages and front page, I want this to be a full sized image. To support this, I need to add something to **functions.php**.

```
add_theme_support( 'post-thumbnails' );
add_image_size('thumb', 200, 200, true);
add_image_size('feature', 960, 9999, false);
```



This adds the option of multiple featured images. The first **add_image_size()** function is for small thumbnails, the second for a large featured image. You can add as many as you like.
Once we've added this, we have to go back and change the image size in earlier versions, including **front-page.php** and **category.php**.

```
<div class="featured-item">
  <h3><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h3>
  <?php the_excerpt(); ?>
```

```
    <?php the_post_thumbnail('thumb'); ?>
  </div>
```

Then we'll update **page.php**.

```
<article class="post">
  <?php the_post_thumbnail('feature'); ?>
  <h2><a href="<?php the_permalink(); ?>"><?php the_title(); ?></a></h2>
  <div class="content">
    <?php the_content(); ?>
  </div>
</article>
```

sidebar.php

Create a blank document and title it **sidebar.php**.

Then add these lines:

```
<div id="sidebar">
  <?php dynamic_sidebar('Sidebar'); ?>
</div>
```

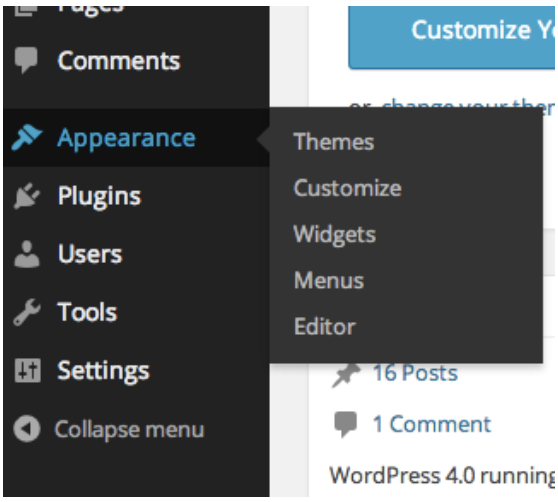
This will import the sidebar from your Dashboard to wherever you place it. Let's add it to **front-page.php:**

```
<?php get_header(); ?>
<?php get_sidebar(); ?>
<div class="posts">
```

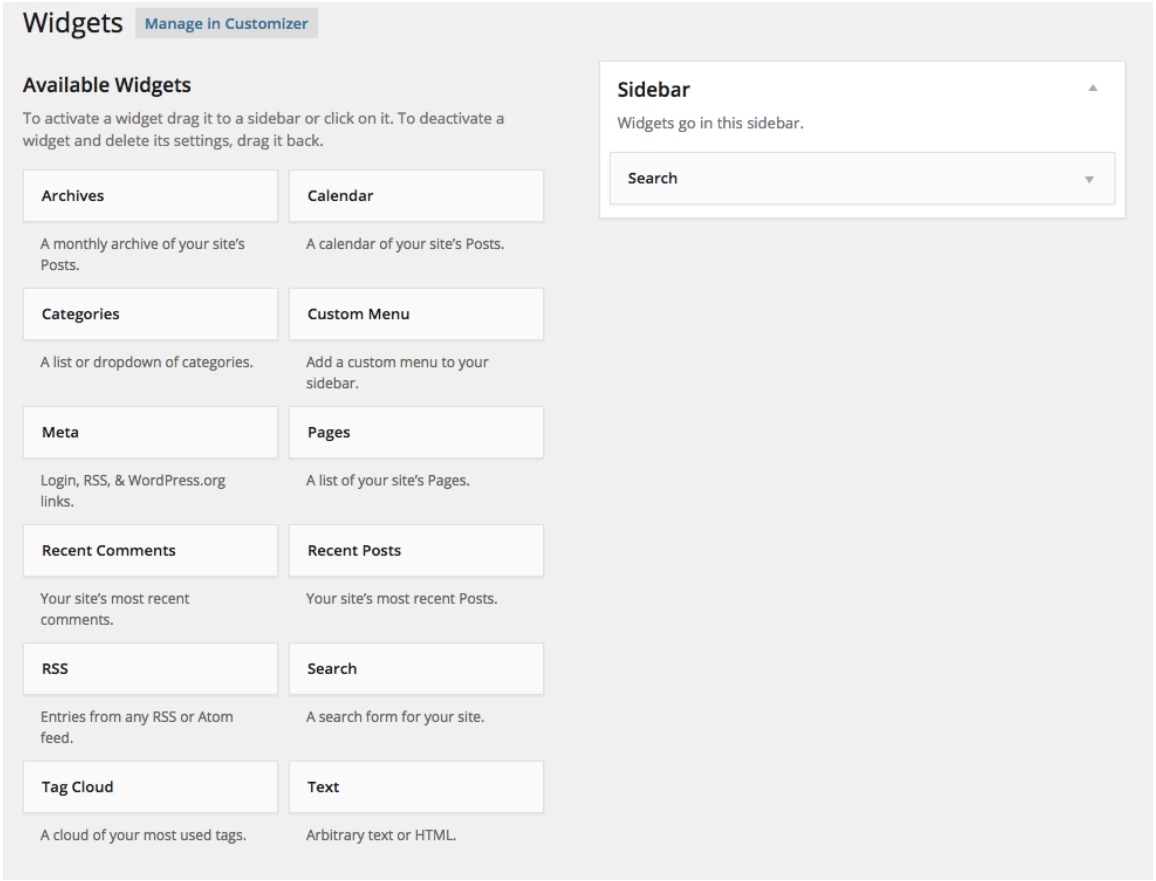
Then we need to add some more lines to **functions.php** to register the sidebar in Dashboard:

```
$sidebar = array(
  'name'    => 'Sidebar',
  'id'     => 'sidebar',
  'description' => 'Place widgets here.',
);
register_sidebar($sidebar);
```

If we want, later multiple sidebars can be added.
Once **functions.php** is uploaded, widgets will appear in the menu.



Then you can dynamically change what appears in the sidebar:



Homework / Readings

Comments
https://codex.wordpress.org/Function_Reference/comments_template

Date and Time Formats

https://codex.wordpress.org/Formatting_Date_and_Time

Tags

http://codex.wordpress.org/Function_Reference/the_tags

Author posts

http://codex.wordpress.org/Function_Reference/the_author_posts_link

Posts and Pages
https://www.webmechanix.com/how-to-add-posts-to-pages-in-wordpress-tutorial/#guide