**CPSC 2030 Course Notes:**

**Creating WordPress Themes Part 2**

**Challenge: make the main title of the site in the header into a link to the homepage. Use a method similar to the method we used to make the post titles into links. However, you will have to use the *bloginfo()* function.**

**Task 4: Template Files**

There are many different types of pages that a WordPress site can display; there’s the index page, a single post, a single page, a list of search results, a list of posts with a certain category or tag, and so on. It turns out that we can specify what happens on those pages by editing different *template files*. For instance, once can set up a single WordPress page by creating and editing the **page.php** file. Or one can create and edit the **archive.php** file to determine what the list of all the posts in a certain category will look like.

**Challenge: even though we have not created any template files other than *index.php*, we can still view pages, archives, search results and single posts using our theme. Why is this? Study the *WordPress Template Hierarchy* to find out:** [**https://developer.wordpress.org/themes/basics/template-hierarchy/**](https://developer.wordpress.org/themes/basics/template-hierarchy/)

Let’s create a file called **single.php**. This file will provide the template for viewing single posts. Copy the content of the **index.php** file into **single.php**. We will use this as a starting point for designing our single posts page. In fact, we’re going to use the exact same code with one change: delete **the\_excerpt()** and replace it with **the\_content()**. This will simply display the full content of the post rather than a small sample, which is usually what we want when we view the full post.

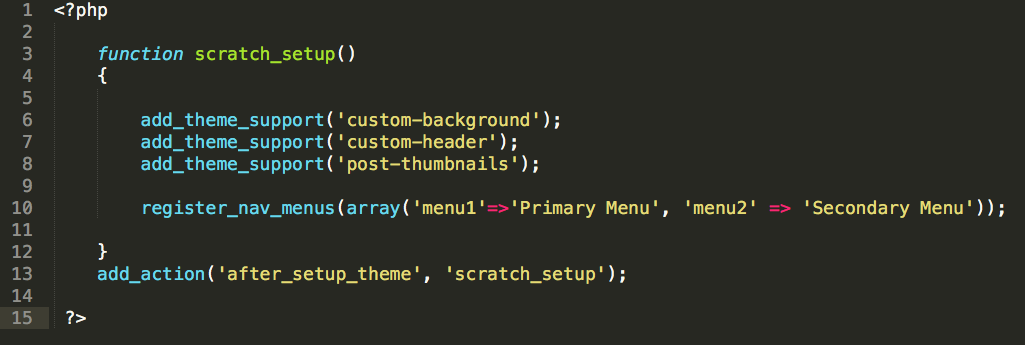
**Challenge: create *page.php* and *archive.php*. Copy the contents of *index.php* into these files, and try making some changes. Experiment with using different template files!**

*\*Tip: we can streamline the template structure of our theme even further by using template parts. Read about template parts here:* [*https://developer.wordpress.org/reference/functions/get\_template\_part/*](https://developer.wordpress.org/reference/functions/get_template_part/)

**Task 5: Adding Support for Theme Features**

You may have noticed that our site is lacking some of the basic functionality that we’ve come to expect from most themes. For example, the Customize bar on our site doesn’t have many things to customize. Also, we don’t seem to have the option to add menus to our site, or custom post thumbnails. The **functions.php** file will help us add these features.

Create a file called **functions.php**, and add the following code:



There’s a lot going on here, but let’s start by looking at lines 6 to 10; what we’re doing here is adding theme support (using the **add\_theme\_support()** function) for custom backgrounds, custom headers, and post thumbnails (or featured images, as they’re now called). We’re also “registering” two menus, which means the user can now set up menus and choose between two locations for them to be displayed. So, we’ve made our theme way better with just a few lines of code!

Of course, there are a few other things that need to be done to get some of these things to work. For starters, we’ll need to actually place an **<img>** tag in our **header.php** file somewhere for the custom header image. We’ll also need to place some code in our theme template files to display the navigation menus that users can now set up. Our new **header.php** file could look like this:

As you can see, we’re adding an **<img>** tag that uses the function **header\_image()** to get the location of the header image the user has chosen. We’re also using the **wp\_nav\_menu()** function to display the menu that we named “menu1” in the **functions.php** file. We could put the menu that we named “menu2” into another location, like maybe the footer.

For more information on adding theme support, look here:

<https://developer.wordpress.org/reference/functions/add_theme_support/>

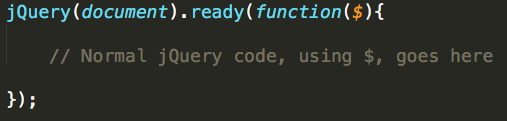
**Task 6: Styles and Scripts**

Noticeably absent from our theme right now is any support for CSS stylesheets or JavaScript. When we were making static web pages using HTML, we would simply place a link or script tag in the <head>. WordPress requires a bit more work than that, but not much. Let’s add this code to our **functions.php**, after the code we added previously:



What we’re doing here is linking our **style.css** stylesheet to the site on line 19, loading jQuery on line 22, (WordPress has jQuery prepackaged) and linking a file called **script.js** on line 25. The **wp\_enqueue\_style** and **wp\_enqueue\_script** functions that we’re using on lines 19 and 25, respectively, both take a string as the first parameter, which is used by WordPress as a handle for that resource, and then take a string for the file path of that file. You can see that to construct the path of each file, we’re using the **get\_template\_directory\_uri** function, which gets the path of the theme directory, and then concatenating it with the file name. (We could also add additional folders to the path this way as well.)

It’s important to mention at this point that there are a few little quirks to using jQuery in WordPress. By default, we won’t be able to use the usual **$** character for our jQuery functions; however, there are a few ways to get around this. The way that I will demonstrate here is simple and adequate for the purposes of this class. Create a file called **script.js** and place this code inside:



Any code inside this function will be able to use the **$** symbol as usual.

One last note on styling WordPress sites using CSS; it may be helpful to be able to target different styles to different types of pages and template files. We can add a function to our header file that will add different classes to the **<body>** tag depending on which page is currently being displayed. In the **header.php** file, replace the **<body>** tag with

**<body <?php body\_class(); ?>>**

(Watch out that you don’t forget the last “>” in that code! We’re closing a php tag and an HTML tag. )

That’s it for this tutorial; these are the basics that are required to make a simple WordPress theme.