Drupal 8 Theming Training

Description

Are you struggling with Drupal 8 theming? New to Drupal or Drupal 8?

Let’s take it from the top!

This hands-on training will take you step-by-step through the process of creating a custom theme in Drupal 8. There will also be time for Drupal Q&A.

Along the way, we’ll cover:

- Setting up Your local development environment
- Working with YAML files
- Adding assets responsibly
- Getting, modifying and display Drupal data in the template
- Cool Twig tips and Tricks
- Preprocessing template functions and hooks
- Basic OOPHP principles
- Kint, theme_debug and other changes to debugging
- Leveraging new Drupal 8 site building paradigms to make theming easier
- Where to find help

Notes:

- All terminal commands are run from the Drupal root.
- `$` indicates a prompt. You do not need to type it into the terminal window.
- `MYDRUPAL` refers to the Drupal root directory or base URL.
- You’ll have an easier time if you configure your editor to use spaces instead of tabs.
- Feel free to ask any of the "Questions you may have …" someone probably asked the question before!
- See a mistake or typo? Submit a pull request on Github!
- Common hiccups are:

- Syntax errors
- Too many or not enough spaces in .yml files
- Cache not cleared
- PHP memory limit not high enough (>= 256)
- settings.php
  - `ini_set('memory_limit', '512M');`
- php.ini
  - `'memory_limit' = '512M'`

Exercises

Exercise 1 - Local Setup

Exercise 2 - Add Content

Exercise 3 - Contrib Themes

Exercise 4 - Dot Info File

Exercise 5 - Libraries

Exercise 6 - Intro to Twig

Exercise 7 - Regions

Exercise 8 - Dot Syntax

Exercise 9 - Twig Classes

Exercise 10 - Twig Filters

Exercise 11 - Twig Blocks

Exercise 12 - Include SVG

Exercise 13 - Preprocess Function

Exercise 14 - Template Suggestions

Exercise 15 - Add Classes with PHP

Exercise 16 - Form Alter

Exercise 17 - Responsive Images

Exercise 18 - Custom Theme Settings 1

Exercise 19 - Custom Theme Settings 2

Style Guides for Contributors

Path to files and directories.

MYDRUPAL/themes Path to files and directories.

Name of file or directory

node.html.twig

Code.

```
$ cd drupal
```

Url

http://MYDRUPAL/admin/config

Basic terminal commands used.

Change Directory.

`cd`

Create File.

`touch`

Create Folder.

`mkdir`

Move file from one location to another.

`mv`

Copy file to a new location.

`cp`

Final Folder Structure

```
MYDRUPAL/themes/custom
                      └── acme
                          ├── acme.breakpoints.yml
                          ├── acme.info.yml
                          ├── acme.libraries.yml
                          ├── acme.settings.yml
                          ├── acme.theme
                          ├── css
                          |   ├── css-stuff-print.css
                          |   ├── css-stuff.css
                          |   └── custom-widget.css
                          ├── images
                          |   └── mysvg.svg
                          ├── js
                          |   └── custom-widget.js
                          ├── templates
                          |   ├── block
                          |   |   └── block--system-powered-by-block.html.twig
                          |   ├── html
                          |   |   └── html.html.twig
                          |   ├── node
                          |   |   └── node.html.twig
                          |   └── page
                          |       └── page.html.twig
                          └── theme-settings.php
```

Done ☺

Exercise 2:

Populate some content

It's hard to theme the content of a site when there is no content. So, before we dive into the good stuff, let's start creating some content.

If you're comfortable creating content in Drupal, skip to "Auto Generate Content".

1. In the admin menu, click `Content` > `Add content` > `Basic page`.
    1. Title: `About`
    2. Body: `This is the awesome site that I'm making with Chapter Three.` (Or filler text from wherever you want.)
    3. Under **Menu Settings** check the box to **Provide a menu link** and leave all defaults.
    4. Click the **Save** button.
2. Click `Content` > `Add content` > `Article`.
    1. Title: `An Interesting Article`
    2. Body: `We're preparing to launch a new Drupal website full of interesting articles like this one.` (Or filler text from wherever you want.)
    3. Tags: `website, Drupal`
    4. Upload an image of your choice with alternative text of your choice.
    5. Click the `Save` button.

Auto Generate Content

The above process is fine for one or two items, but what if you need to test out what 50 nodes look like in a view? What if you need to know how the page is going to look with a couple dozen comments all at different levels? Are you going to create each one of those items by hand? Hopefully not. Luckily, we have a handy module that has already been ported to D8 and is (mostly) able to take care of this work for us. Enter the "Devel" and "Devel Generate" module.

1. Enable the **Devel generate** module if you have not already.

For vocabularies and taxonomy terms

1. Click "Configuration" in the Admin menu.
2. In the `Development` group select `Generate vocabularies`.

1. Number of vocabularies?: `2`
2. Maximum number of characters in vocabulary names: `20`
3. Click the **Generate** button.

3. Click "Configuration" in the Admin menu
4. In the `Development` group select `Generate terms`.

    1. Vocabularies: **Select any one vocabulary**
    2. Number of terms?: `10`
    3. Maximum number of characters in vocabulary names: `15`
    4. Click the **Generate** button.

For content

1. Click "Configuration" in the Admin menu.
2. In the `Development` group select `Generate content`.

    1. Select “Content Type: `Article`
    2. How many nodes would you like to generate?: `25`
    3. How far back in time should the nodes be dated?: `1 year`
    4. (If comments enabled) Maximum number of comments per node: `5`
    5. Maximum number of words in titles: `10`
    6. Leave the rest at their default values
    7. Click the **Generate** button.

For users

1. Click "Configuration" in the Admin menu
2. In the `Development` group select `Generate users`.

    1. How many users would you like to generate?: `10`
    2. Which roles should the users receive?: **Select a role or leave unchecked depending on what role/setup you would like to test**.
    3. Password to be set?: **Set password or leave blank**
    4. How old should user accounts be?: `1 week`
    5. Click the **Generate** button.

For menus

1. Click "Configuration" in the Admin menu
2. In the `Development` group select `Generate menus`.

    1. Generate links for these menus?: `Create new menu` & `Main navigation`
    2. Number of new menus to create?: `1`
    3. Number of links to generate?: `5`
    4. Maximum number of characters in menu and menu link names?: `12`

5. Leave defaults for the rest or tweak to your desired needs
6. Click the **Generate** button.

Questions you may have…

-

Done ☺

Exercise 3 - Contrib Themes is next!

[Drupal 8 Theming]

Exercise 3:

Contrib themes

This is where most themers will start their career. Let someone else do the work for you and download something generic that can be modified to fit your needs. Contributed themes can be a great resource for learning new techniques and functionality, which you can implement in your custom themes.

You can find themes at http://drupal.org/project/themes, as well as some external sites. Make sure the theme is compatible with Drupal 8.

If you run into problems, check the theme's issue queue and search the forums. If your problem hasn't already been addressed, post a question, and someone will try to help you out.

Install our first contrib…

1. Download and add the latest recommended release of *Radix*: http://drupal.org/project/radix

   ```
   $ composer require drupal/radix
   ```

2. Also, download and add the latest recommended release of the *Mayo* theme: https://www.drupal.org/project/mayo

   ```
   $ composer require drupal/mayo
   ```

3. Read `README.txt` inside of the *mayo* folder to know what to expect.

4. Enable the *Mayo* theme:

   1. Click on **Appearance** at the top.
   2. Find *Mayo* near the bottom and click **Install and set default**.

5. Clear your cache.

> ### Typical structure of a theme
>
> - **MYTHEME.info.yml** - A theme must contain an .info.yml file to define the theme. Among other

things, the .info.yml files define metadata, style sheets, and block regions. This is the only required file in the theme.

- **MYTHEME.libraries.yml** - The .libraries.yml file is used to define JavaScript and CSS libraries that can be loaded by the theme.
- **MYTHEME.breakpoints.yml** - Breakpoints define where a design changes in response to different devices. While the theme can use this file, its info can also be used by Drupal core to make adjustments to data it sends to the theme.
- **MYTHEME.theme** - The .theme file is equivalent to the old Drupal 7 template.php file. It is a PHP file that contains conditional logic and data (pre)processing of the output.
- **screenshot.png** - If a screenshot.png file is found in the theme folder, it will be used on the Appearance page. You can also define a screenshot image in .info.yml file.
- **logo.svg** - New to Drupal 8, logos are best saved as svg files and placed in the main level of your theme. Logos can also be uploaded at Appearance > Settings.
- **css/** or **styles/** - Drupal 8 core themes organize CSS files following the SMACCS style guide. For CSS files to be loaded, they must be defined in your .libraries.yml file. You can override or remove core and module CSS in your themes .info.yml file.
- **js/** or **script/** - JavaScript files are stored in the 'js' folder. For a theme to load JavaScript files, they must be defined in .libraries.yml file.
- **img/** or **images/** - It is good practice to store images in the 'images' subfolder.
- **templates/** - Templates provide HTML markup and some presentation logic. It is customary to place template files into subcategory folders, such as **templates/node/** for node based templates.

… then we configure our theme

1. Revisit the Appearance page and click **Settings** under the Mayo theme.
2. Play with some of the configuration settings.
3. Try to complete the following tasks:
    1. Set it so that both sidebars appear on the left side on big screens.
    2. Change the base font selection to be `Verdana, Geneva, Arial, ...`
    3. Set sidebar to use round corners.
    4. Tweak the color scheme by utilizing the color wheel.

4. Many settings are theme-specific. Visit the settings pages of other themes, like *Seven* to compare.

Questions you may have…

- How do I choose a theme?

Done ☺

Woo hoo! Time for Exercise 4 - Dot Info File!

[Drupal 8 Theming]

Exercise 4:

The .info.yml file

To create a new theme for Drupal 8, the only real requirement is to make sure you have implemented the .info.yml file (YaML, like Camel). Drupal 8 runs off of YaML files much the way that Drupal 7 ran off of .info files.

As long as you have your .info.yml file in place with a few keys in place, you have a theme (although if that's all you have in place, it's not going to be a pretty theme).

Create the theme folder and "YaML" file

1. Locate the **MYDRUPAL/themes** folder in your Drupal installation. In Drupal 8, Contrib and Custom modules and themes, are saved at the root level **MYDRUPAL/modules** and **MYDRUPAL/themes** folders.
2. Create a new folder called `custom` and inside it a folder called `acme`

```
$ cd MYDRUPAL
$ mkdir themes/custom
$ mkdir themes/custom/acme
```

3. In that folder create a new file. It should be called **acme.info.yml**

```
$ touch themes/custom/acme/acme.info.yml
```

4. Add the following lines to that file:

```
name: Acme
description: My first Drupal 8 theme.
type: theme
base theme: classy
core: 8.x
version: VERSION

regions:
  header: Header
  primary_menu: 'Primary menu'
  secondary_menu: 'Secondary menu'
  breadcrumb: Breadcrumb
  help: Help
  hightlighted: Highlighted
  content: Content
  sidebar_first: 'Left sidebar'
  sidebar_second: 'Right Sidebar'
  footer: Footer
  page_top: 'Page top'
  page_bottom: 'Page Bottom'
```

5. Clear cache.
6. Go to /admin/appearance and under the Acme theme choose `Install and set as default`.
7. Go to the home page and observe your beautiful new theme.

Things we might find in a .info.yml file

The following keys are items that we will often find in a `*.info.yml` file. Some are optional; some are required. These keys provide metadata about your theme and define some of the basic functionality.

1. `name` **Required** The human-readable name will appear on the Appearance page, where you can activate your theme.
2. `description` **Required** The description is displayed on the Appearance page.
3. `type` **Required** The type key indicates the type of extension, e.g., module, theme or profile. For themes this should always be set to `theme`.
4. `base theme` The theme can inherit the resources from another theme by defining it as a base theme. Not declaring this, will default to using "Stable" as the base theme.

5. `core` **Required** The core key specifies the version of Drupal core that your theme is compatible with.

6. `version` For modules hosted on drupal.org, the version number will be filled in by the packaging script. You should not specify it manually, but leave out the version line entirely.

7. `regions` Regions are declared as children of the regions key. You are required to have a content region. The regions we just declared are also the default regions that are enabled by core if you do not declare any regions in your .info.yml file.

Other items

1. `regions_hidden` will allow you to remove default regions from the output if you don't specifically declare any regions.

2. `screenshot: IMAGE_NAME.png` With the screenshot key you define a screenshot that is shown on the Appearance page. If you do not define this key, then Drupal will look for a file named 'screenshot.png' or 'screenshot.svg' in the theme folder to display.

3. The `libraries` key can be used to add asset libraries to all pages where the theme is active.

```
libraries:
    - THEMENAME/global-styling
    - THEMENAME/LIBRARY-NAME
```

Theme libraries contain CSS and/or javascript. Theme libraries are declared in another type of YaML file (THEME.libraries.yml). We will cover libraries in later exercises.

1. The `libraries-override` and `libraries-extend` keys can be used to take control over the components of a library, remove items or the complete library, or add additional elements to a library.

2. The `stylesheets-remove` key is used to stop the addition of CSS components for core and contrib modules. However, it is better to do this with `libraries-override`.

```
stylesheets-remove:
 - core/assets/vendor/normalize-css/normalize.css
 - '@classy/css/components/tabs.css'
```

3. `ckeditor_stylesheets` will allow you to have custom ckeditor styles attached to your theme

> 4. `quickedit_stylesheets` will allow you to have custom styles and javascript attached to the quickedit functionality

Questions you may have…

- What happens if I don't declare any regions?
- Why are some regions in `''` and others not?

Done ☺

Exercise 5 - Libraries awaits!

[Drupal 8 Theming]

Exercise 5:

Drupal asset libraries: adding CSS and JavaScript to your site

In Drupal 8, "Asset Libraries" are used for loading stylesheets (CSS) and JavaScript (JS). The libraries system that you use in your theme is the same framework used by modules and core. Asset libraries can contain one or more CSS assets, one or more JS assets, and one or more JS settings.

Drupal follows this high-level principle: CSS and JavaScript are only loaded if you tell Drupal it should load them. Drupal has stopped assuming that it should load all assets on all pages, because this is bad for front-end performance.

The biggest difference from Drupal 7 is when it comes to JavaScript loaded on pages. By default Drupal doesn't need JavaScript on most pages that anonymous users can see. This means that jQuery is not automatically loaded on all pages anymore. If your theme does need it, you can declare it in one of your theme's libraries, and it will be added to every page.

The general process for adding CSS and JavaScript

The basic process breaks down into 3 steps

1. Save the CSS or JS to a file.
2. Define a "library" in a `*.libraries.yml` file, which contains a reference to the CSS and/or JS files.
3. "Attach" the library to a render array in a hook using the `#attached` attribute, by including it in a template, or add the library as a dependency to the theme's `*.info.yml` file.

Create a library

In the following steps, we will create a libraries.yml file, declare our library some CSS files and create the CSS files in out theme.

1. Navigate to your theme's root directory

```
$ cd MYDRUPAL
```

2. Create a file called **acme.libraries.yml** and open that file in your preferred code editor.

```
$ touch themes/custom/acme/acme.libraries.yml
```

3. Add the following code to that file:

```
css-stuff:
  version: VERSION
  css:
    theme:
      css/css-stuff.css: {}
      css/css-stuff-print.css: { media: print }
      //fonts.googleapis.com/css?family=Abhaya+Libre|Open+Sans: { type: external }
```

We just declared our library called `css-stuff`. We can give it any name we want as long as another module or theme hasn't declared it. A best practice for theme libraries is to use some word(s) to describe what the library will do. A name like `global-styles` would be more accurate for this library, but I'm a free spirit and I like to run with scissors.

4. Navigate back to your theme's root directory, and create a folder called **"css"** (if one doesn't already exist).

```
$ mkdir themes/custom/acme/css
```

5. Create two files in that directory. One called **css-stuff.css** and one called **css-stuff-print.css**

```
$ touch themes/custom/acme/css/css-stuff.css
$ touch themes/custom/acme/css/css-stuff-print.css
```

6. Open them both and add any high level css you want. For **css-stuff.css** you could add:

```
body {
  background-color: #e7e7e7;
  font-family: 'Open Sans', serif;
  padding: 30px;
}

h2 a {
  background-color: #0c89af;
  color: white;
  font-family: 'Abhaya Libre', serif;
  padding: 3px;
  text-decoration: none;
}

h4 {
  background-color: #3baf8a;
  font-size: 1.5em;
  padding: 3px;
}
```

and for **css-stuff-print.css**, you could add:

```
body {
 font-family: sans-serif;
 color: #888;
}
```

Go crazy if you want!

7. Navigate back to your theme root directory and open your **acme.info.yml** file
8. We need to make our theme aware of our newly declared library before anything happens.
9. Add the following code to your **acme.info.yml** file

```
libraries:
  - acme/css-stuff
```

10. Clear your caches and you should now see your css styles in place.

    If everything worked, you should see output like the following in the head section in the source code on all pages of your theme.

```
@import url("/themes/acme/css/css-stuff.css?of7sd1");
...
<style media="print">
    @import url("/themes/acme/css/css-stuff-print.css?of7sd1");
</style>
```

Attach Library to a template

In Drupal 8, it is recommended that libraries only be applied when needed. Unless a style is being used on every page, we should use the `attach_library()` function in twig to add JS or CSS.

1. Create a folder called **js** in your theme root and add a JavaScript file called **custom-widget.js**

```
$ mkdir themes/custom/acme/js
$ touch themes/custom/acme/js/custom-widget.js
```

1. Create a CSS file in the **css** folder called **custom-widget.css**.

```
$ touch themes/custom/acme/css/custom-widget.css
```

1. Add some sample CSS and js to each file:

   custom-widget.js:

   ```
   console.log('It Works');
   ```

   custom-widget.css:

   ```
   article {
     background: white;
     padding: 10px;
   }
   ```

2. Define your new library in **acme.libraries.yml**

```
widget:
  version: VERSION
  js:
     js/custom-widget.js: {}
  css:
    theme:
      css/custom-widget.css: {}
  dependencies:
     - core/drupal
     - core/jquery
```

3. Navigate to **MYDRUPAL/core/themes/classy/templates/content/** and look for **node.html.twig**.

4. Make a copy of **node.html.twig** and paste it in at **MYDRUPAL/themes/custom/acme/templates/node**. Then, because you have added a new file, you'll need to clear cache.

```
$ mkdir themes/custom/acme/templates
$ mkdir themes/custom/acme/templates/node
$ cp core/themes/classy/templates/content/node.html.twig themes/custom/acme/templates/r
$ drush cr
```

1. Add the following to your theme's new **node.html.twig** near the top

```
{% if node.bundle == 'article' %}
  {{ attach_library('acme/widget') }}
{% endif %}
```

2. Verify that **custom-widget.js** and **custom-widget.css** have loaded on the article pages but NOT on basic pages.

Overriding and removing libraries

Removing CSS or javascript

If we are using a base theme, it is very possible that some CSS or javascript is coming over from the base theme that we just do not want to use in our theme. Or sometimes there stylesheets coming from core that we just don't want. Even though Drupal tries to follow the rule of only "load stuff only if needed," you may find yourself in a situation where a contrib module or other library is just adding too much. We can remove stylesheets from output at the theme level by using the `stylesheets-remove` key in our *.info.yml* file.

The following method is for removing individual CSS or javascript files

1. Take a look at the CSS that is being loaded on your site as a logged in user. Note that **/core/assets/vendor/jquery.ui/themes/base/dialog.css** and **/core/themes/classy/css/components/breadcrumb.css** are being loaded.

2. Navigate to your theme root and open the **acme.info.yml** file.

3. Place the following code in your **acme.info.yml** file.

   ```
   stylesheets-remove:
     - core/misc/active-links.css
     - '@classy/css/components/breadcrumb.css'
   ```

   In this example, we are removing the extra CSS file that core tries to add with one of its JQuery libraries. We are also removing a stylesheet from our parent (base) theme, classy. You probably notice the `@` symbol at the start of the third line. This is a placeholder token. @classy, or @node, or @whatever is the equivalent to `drupal_get_path()` in Drupal 7. Note that `stylesheets-remove` is technically deprecated and will be removed in Drupal 9.

Overriding CSS or javascript

Sometimes we don't want to remove a file or a library, but we do have a better file for it to use. We can override whole libraries or components of them. Below is an example of libraries-override from drupal.org. You would place this in your *.info.yml file.

Note: The methods for modifying libraries have changed during the development of Drupal 8. Check Drupal.org's Theming Guide for the latest documentation.

> **This code is just for show.**

```
libraries-override:
 # Replace an entire library.
 core/drupal.collapse: mytheme/collapse

 # Replace an asset with another.
 subtheme/library:
   css:
     theme:
       css/layout.css: css/my-layout.css

 # Replace a core module JavaScript asset.
 toolbar/toolbar:
   js:
     js/views/BodyVisualView.js: js/views/BodyVisualView.js

 # Remove an asset.
 drupal/dialog:
   css:
     theme:
       dialog.theme.css: false

 # Remove an entire library.
 core/modernizr: false
```

Libraries Extend Example

```
# Extend drupal.user: add assets from classy's user libraries.
libraries-extend:
 core/drupal.user:
   - classy/user1
   - classy/user2
```

Questions you may have…

- Why is the word `theme` in the library definition?
- Is `//fonts.googleapis.com/...` a comment?
- Why do some lines start with `-` in .yml files and others don't?

Done ☺

[Drupal 8 Theming]

Exercise 6:

Intro to Twig

Twig is a modern, advanced, templating language for PHP. Twig is also the new templating engine for Drupal 8. It replaces the old and antiquated phptemplate engine. Twig is fast, secure, and incredibly flexible, but one of its best attributes is that it does not allow some of the bad habits in Drupal theming that phptemplate allowed in the past. (I'm referring to stuff like database queries and data preprocessing in the template files. We've all done it at some point, and yes, we are all ashamed.)

If debugging is not enabled, please see "Exercise 1"

Basic Twig

There are 3 Basic Syntaxes of Twig. Mostly, we will use just 2 of them.

The "Say Something" Syntax: {{ … }}

The double-curly-brace (`{{`) is always used to **print** something. If whatever you need to do will result in something being printed to the screen, then you'll use this syntax. I call this the "say something" tag, because it's how you "speak" in Twig.

- Printing a variable

```
{{ content }}
```

The "Do Something" Syntax: {% … %}

The curly-percent (`{%`) is the other syntax, which I call the "do something" syntax. It's used for things like **if** and **for** tags as well as other things that "do" something. There are only a handful of things that can be used inside of it. If you go to Twig's website, click Documentation, and scroll down, you can see a full list of everything in Twig. The "tags" header shows you everything that can be used inside of a "do something" tag, with more details about how each of these works. The only ones you need to worry about now are **if** and **for**.

We'll talk about a bunch more of these later.

- Running a function (in this case, check if the variable 'logo' is set, if so print the logo)

```
{% if logo %}
  {{ logo }}
{% endif %}
```

- Runing a loop and print a value (in this case, for each product item in products array, it'll print out the product item content)

```
{% for product in products %}
  <h2>{{product}}</h2>
{% endfor %}
```

The Comment Syntax: {# … #}

Actually, There is a third syntax, used for comments: `{#` . Just like with the "say something" and "do something" syntaxes, write the opening {# and also the closing #} at the end of your comments:

- An example comment

```
{# This is a comment for you to enjoy :) #}
```

What's in a twig template

1. Open your theme's **node.html.twig** file in a text editor and add one of the following lines somewhere at the end of the twig template `{{ dump(date) }}` or `{{ kint(content_attributes) }}`
2. Visit a node page, and lets see what it gives us.

Use Twig

Print out the bundle type for the node:

1. Inspect the **node.html.twig** template, review the comments and variables.
2. Add a line to output the the node's bundle type before

`<div{{ content_attributes.addClass('node__content') }}>` :

```
<h4>{{ node.bundle}}</h4>
```

3. Clear cache, visit (or refresh) a node.

Questions you may have

- Can I modify a variable through Twig?

Done ☺

Get your Exercise 7 - Regions on.

Exercise 7:

Adding a new region (Let's use Twig)

The client wants a new region for special messages at the top of the footer, so we'll give them a new region called `Footer top`. Eventually, it would be great to style this region to look a little nicer, but for now, lets just put the content in place.

1. Update **acme.info.yml** to include the new Footer top region:

```
regions:
  ...
  footer_top: 'Footer top'
  ...
```

2. Now that Drupal knows that a new region is available, we need to print that region to the page by modifying **page.html.twig**.

3. Navigate to the Classy theme in **MYDRUPAL/core/themes/classy/templates/layout** and locate **page.html.twig**.

4. Make a copy of **page.html.twig** and put it in the **acme** theme folder. We like to keep it organized, so we'll put it in **MYDRUPAL/themes/custom/acme/templates/page**.

```
$ cd MYDRUPAL
$ mkdir themes/custom/acme/templates/page
$ cp core/themes/classy/templates/layout/page.html.twig themes/custom/acme/templat
```

5. Edit your theme's new version of **page.html.twig**: Look for and copy the following lines of code around line 81:

```
...

{% if page.footer %}
<footer role="contentinfo">
  {{ page.footer }}
</footer>
{% endif %}

...
```

6. Above this footer callout code, paste your copy.

7. In your pasted code, Replace `footer` with `footer_top`, It should look like this when you're finished:

```
...

{% if page.footer_top %}
<footer role="contentinfo">
  {{ page.footer_top }}
</footer>
{% endif %}

{% if page.footer %}
<footer role="contentinfo">
  {{ page.footer }}
</footer>
{% endif %}

...
```

8. Flush your cache and try putting a block in the new region.

Questions you may have…

- Can I add more regions? Can I remove ones I don't need?
- How do I style my new region?

Done ☺

Sometimes you feel like a nut; sometimes you Exercise 8 - Dot Syntax.

Exercise 8:

Manipulating variables in the template

Look at the comments at the top of the **node.html.twig** template. The comments detail the variables that are available to this particular template. In twig, we can use certain filters to manipulate the output of variables.

Use `kint()` to inspect the content variable.

1. Add `{{ kint(content) }}` to the bottom of your **node.twig.html** field.
2. Go to an Article node page.
3. Inspect the content variable. Note that body and field_image are available.

   Make sure your php memory limit is high (>= 256), or you may see a white screen of death.

Print content without certain fields.

1. Delete the kint statement.
2. Verify that the article you're looking at has Tags.
3. In **node.html.twig**, change `{{ content }}` to `{{ content|without('field_tags') }}`
4. View an article node page. Now all fields are printed except for the tags field.

Print fields individually.

1. Find `{{ content | without('field_tags') }}` from the previous step and change it to `{{ content | without ('field_tags', 'field_image') }}`

   This removes the tags and the image fields.

2. Add the following directly above `{{ content | without ('field_tags', 'field_image') }}` .

```
<div class="sidebar">
  {{ content.field_tags }}
  {{ content.field_image }}
</div>
```

You should now see your Tags and Image fields inside a div with class `sidebar`.

3. Add a little styling to **css-stuff.css** to get the full effect.

```
.node__content {
    display: grid;
    grid-template-columns: 1fr 3fr;
}
```

Explore.

The `.` syntax in twig is a shorthand for some PHP methods and functions. Below is a list of methods Twig will check in the order they will be checked in.

```
{{ sandwich.cheese }}
```

```
// Array key.
$sandwich['cheese'];
// Object property.
$sandwich->cheese;
// Also works for magic get (provided you implement magic isset).
$sandwich->__isset('cheese'); && $sandwich->__get('cheese');
// Object method.
$sandwich->cheese();
// Object get method convention.
$sandwich->getCheese();
// Object is method convention.
$sandwich->isCheese();
// Method doesn't exist/dynamic method.
$sandwich->__call('cheese');
```

1. Spend a few moments trying to print out variables and their children.

Questions you may have…

- What if I only want the body text without the surrounding markup?
- Why do you keep telling me to delete my kint statements?

Done ☺

Next stop: Exercise 9 - Twig Classes

[Drupal 8 Theming]

Exercise 9:

Drupal has some handy functions specifically designed for the manipulation of HTML elements.

Manipulating Classes with Twig

Inspect the `attributes` array.

1. Add `{{ kint(attributes) }}` in your theme's **node.html.twig**. Review the object and its properties and methods. Note that `addClass()` is an available public method.

Add a class.

1. Change `<article{{ attributes.addClass(classes) }}>` to `<article{{ attributes.addClass(classes).addClass('myclass') }}>`. Observe the new value in kint as well as the new class in your site's markup.

Add multiple classes to the body tag.

1. Remove all kint() statements from **node.html.twig**.
2. Copy classy's **html.html.twig** into your theme.

```
$ cd MYDRUPAL
$ mkdir themes/custom/acme/templates/html
$ cp core/themes/classy/templates/layout/html.html.twig themes/custom/acme/templat
```

3. Clear Cache
4. Above the DOCTYPE declaration and below the comments, add the following code.

```
{% set myclasses = ['red', 'green', 'blue'] %}
```

5. Find the line `<body{{ attributes.addClass(body_classes) }}>` and change it to

```
<body{{ attributes.addClass(body_classes).addClass(myclasses) }}>
```

You should now see classes `red`, `green` and `blue` attached to the body tag.

Create a new custom variable.

Let's create a special body class for the user role.

1. Note that `logged_in` is one of the variables available in **html.html.twig** according to its comments.
2. Use `{{ kint(user) }}` to inspect the user variable in **html.html.twig**. Search the kint for `account`. Note that the method `getAccount()` is public. If we look back at Exercise 8, we see that we can use `{{ user.account }}` because of that sweet, sweet Twig magic.
3. Use `{{ kint(user.account }}`. Search for `roles`. Look at that! Our friends at D.O. also made `getAccount()` public. Now we can do the thing.
4. Below `{% set myclasses = ['red', 'green', 'blue'] %}` add

```
{% if logged_in %}
    {% set roles = user.account.roles %}
{% endif %}
```

1. Change the body tag to

```
<body{{ attributes.addClass(body_classes).addClass(myclasses).addClass(roles) }}>
```

1. Compare the `<body>` tag for logged in and logged out users. What are the results when you remove the `if` statement?

Questions you may have…

- How do I remove a class?
- Where can I find other cool twig functions?
- What happened to preprocess functions and the template.php file?
- Why do we copy our template files from the classy theme?
- How did you know the `user` variable was available?

Drupal 8 Theming

Exercise 10:

Advanced Twig

Using filters

Twig filters allow you to make changes to markup right in the template. Let's use our node title to test some string filters.

1. First, we want to get the node title as a string.
2. Try `{{ kint(node) }}` in your theme's **node.html.twig** file. Under `Available methods` observe `getTitle()`. It is a public function so we can use it in our twig template.
3. Near your other variable declaration in your theme's **node.html.twig** add:

```
{%  set title_string = node.label %}
```

4. Test it with the following filters.

```
clean_class: {{ title_string|clean_class }} <br/>
upper: {{ title_string|upper }} <br/>
length: {{ title_string|length }} <br/>
```

5. If you have time, try creating arrays, loops and object to test some of the other amazing filters for twig See: https://twig.symfony.com/doc/2.x/filters/index.html.

Questions you may have…

- When should one use Twig filters instead of CSS or PHP?
- Why didn't you use node.title instead of node.label?

Done ☺

Exercise 11:

Twig blocks and Drupal blocks

Twig introduces another "block" concept within Drupal. However, a Twig "block" is not the same as a Drupal block.

A Twig "block" is an element in the template file that can be overridden, independently from its inherited (parent) template. This means I can reuse the majority of a parent template and just change the one part I need to change. In Drupal 7, you would have to copy the whole template into a new file and be stuck trying to manage two independent templates or use PHP to conditionally include a new file. Twig "blocks" are used for inheritance and act as placeholders and replacements at the same time.

Twig block official documentation http://twig.sensiolabs.org/doc/tags/extends.html

Using twig block inheritence

In Drupal 8 core, we will use the original example Drupal block, "Powered by Drupal" block, to show you an example of using twig blocks. The "Powered by Drupal" Block utilizes the default block template. This template contains a twig block inside of it named `content` .

We are going to copy the default block twig template into our theme, rename it and set it to only alter the value of the twig block. This way our new template will follow the default template, but only change the value inside of our Twig block.

1. First thing is to make sure that the "Powered by Drupal" block is displayed. If not, go to the block admin page and place the "Powered by Drupal" block into any of your theme's regions.
2. **Save** the configuration, and visit a page on the front end of your site. **Make sure you can see the block somewhere on your site**.
3. Locate and copy the **block.html.twig** template located at **MYDRUPAL/core/themes/classy/templates/block/block.html.twig** and copy it into your **/templates/block** folder of your custom theme.

```
$ cd MYDRUPAL
$ mkdir themes/custom/acme/templates/block
$ cp core/themes/classy/templates/block/block.html.twig themes/custom/acme/template
```

4. Open the new **block.html.twig** file in your preferred code editor. You should see the following code:

```
{%
  set classes = [
    'block',
    'block-' ~ configuration.provider|clean_class,
    'block-' ~ plugin_id|clean_class,
  ]
%}
<div{{ attributes.addClass(classes) }}>
  {{ title_prefix }}
  {% if label %}
    <h2{{ title_attributes }}>{{ label }}</h2>
  {% endif %}
  {{ title_suffix }}
  {% block content %}
    {{ content }}
  {% endblock %}
</div>
```

5. We are going to copy and rename this file using information from our debug setup. View the source code on a page where the "Powered by Drupal" block is visible. Locate the code for the block in your browser's dev tools. You should see some additional code that looks like this:

```
<!-- FILE NAME SUGGESTIONS:
* block--acme-powered.html.twig
* block--system-powered-by-block.html.twig
* block--system.html.twig
x block.html.twig
-->
```

   These are the template suggestions for different components of the page. These come from our theme and twig debug enabling. Drupal is telling you exactly how you can name your new template files to make sure they affect that component. The `x` at the start of a name refers to the template that is currently being used to build that component. Items lower on the list override higher ones.

6. So that it only affects the "Powered by Drupal" block, rename the `block.html.twig` file to

`block--system-powered-by-block.html.twig`

```
$ cd MYDRUPAL
$ mv themes/custom/acme/templates/block/block.html.twig themes/custom/acme/template
```

7. Open the `block--system-powered-by-block.html.twig` file and replace all the code in it with the following code:

```
{% extends '/core/themes/classy/templates/block/block.html.twig' %}

{% block content %}
<div class="d8-power">{{ 'I am powered by Drupal 8, haha!'|t }}</div>
{% endblock %}
```

8. Clear cache.

We only override the content inside the twig block name **"content"**, The rest of the items are still controlled by the default block.html.twig template file. If we had to make a change to all Drupal blocks, like add a wrapping div or a class to all blocks, we could make that change in our default template, and it would still apply to our overridden template.

Questions you may have…

- What is the `|t` in our twig template?
- What is the `{% trans %}` component in twig?
- Can I use all the Twig functions from SensioLabs?

Done ☺

… Exercise 12 - Include SVG …

[Drupal 8 Theming]

Exercise 12:

Include an SVG inline.

Often we want to include SVGs in our project because they load fast and can be easily manipulated using CSS. In this example, we use the Twig `include` function to add svg code to our template inline. This method can be used to include any snippet of HTML that needs to be accessed across templates.

1. Create a folder called **images** in your theme root.

```
$ cd MYDRUPAL
$ mkdir themes/custom/acme/images
```

1. Copy and paste svg code into a file called **mysvg.svg** inside your newly created **images** folder

```
$ touch themes/custom/acme/images/mysvg.svg
```

You can use an svg on your machine or the example below.

```
<svg>
  <rect x="10" y="10" height="100" width="100"style="stroke:#ff0000; fill: #0000ff"/>
</svg>
```

1. Paste the following code into .twig file.

```
{% include active_theme_path() ~ '/images/mysvg.svg' %}
```

2. View a page that include the template you modified.

Questions you may have...

- Can other types of files can be included?

- Where does the function `active_theme_path()` come from?

Done ☺

Onward! Exercise 13 - Preprocess Functions

Exercise 13:

.theme file and preprocess_page

Don't make yourself responsible for updating every site on January 1st. Let the computer do the work for you and automatically set items, like the copyright date, to the current year. We will begin to dive into some more advanced elements of Drupal theming including preprocess functions and template variables.

What is a preprocess function? A preprocess function is a function that allows us to manipulate (add, edit, or remove) our data before our template processes it. Hence, the "pre" in preprocess. It's powered by `hook_preprocess()` function. `hook_preprocess_HOOK()` allows modules to preprocess theme variables, but for a specific theme hook, like node, page, or menu_link. For our example, we will manipulate the data before it goes to the `page` twig template **page.html.twig**, so we will use "hook_preprocess_**page**()"

Create our .theme file

1. Create a file at the root of the Acme theme named **acme.theme** and open in your preferred code editor

   ```
   $ cd MYDRUPAL
   $ touch themes/custom/acme/acme.theme
   ```

2. Write the following code in your new **acme.theme** file:

   ```
   <?php
   function acme_preprocess_page(&$variables) {
       $variables['copyright'] = "It works!";
   }
   ```

   Note that there is not a closing ?> at the end, but there should be a line of whitespace.

3. Open up **page.html.twig** and near the closing footer element, print out our new variable using the following code:

```
{% if copyright %}
  <div class="copyright">
  {{ copyright }}
  </div>
{% endif %}
```

4. Clear cache and verify your copyright is on the page.

What year is it?

Great, it's printing our variable, but we're still not much better off than simply adding a block to say the copyright date manually. Let's start adding some logic in **acme.theme**.

1. Open **acme.theme** and add some php around our new variable so that it can print the current year instead of our filler message:

```
<?php
function acme_preprocess_page(&$variables) {
  $variables['copyright'] = t("Copyright @date",
    array('@date' => date('Y'))
  );
}
```

Note that there is not a closing ?> at the end, but there should be a line of whitespace.

Questions you may have…

- Where did you find that function name? I would never have guessed that.
- What would happen if I didn't check if a variable is set before printing it?
- Why don't I close the PHP tags?
- Why does the new variable have a t() function around it?
- What's the best way to add markup and styles to our variable?
- Could I have done all this on the twig template?

Done ☺

Exercise 14 - Template Suggestions is ready to bat.

Exercise 14:

Create new template suggestions

In Drupal 7, we used to create new template suggestions in our preprocess functions. In Drupal 8, we use two new functions to create new suggestions for Twig templates.

These functions are `hook_theme_suggestions_alter()` function and the more targeted, `hook_theme_suggestions_HOOK_alter()` function. The `HOOK` refers to the theme hook being used, like `node` or `page` or `menu_link`. This `HOOK` helps to focus in your suggestions to a specific theme component. In this exercise, we will create new suggestions for our 'node' templates based off if the user is logged in or not.

1. Open the **acme.theme** file and add the following function:

```
function acme_theme_suggestions_node_alter(&$suggestions, $variables, $hook) {
    // Our code will go here.
}
```

2. Inside of our new function, add the following code:

```
...
if (\Drupal::currentUser()->isAuthenticated()) {
    $bundle = $variables['elements']['#node']->bundle();
    $mode = $variables['elements']['#view_mode'];
    $view_mode = strtr($mode, '.', '_');

    $suggestions[] = 'node__' . $bundle . '__logged_in';
    $suggestions[] = 'node__' . $view_mode . '__logged_in';
}
...
```

   This will create a new suggestion for content types and node view modes for when the viewer is logged in.

3. Clear your caches and visit a node page. View the source code and see if you can find your new template suggestions at the start of your node output.

Questions you may have…

- What is a theme suggestion HOOK?
- What do I do with these new theme suggestions?

Done ☺

Yes, there's more. Exercise 15 - Add Classes with PHP

Exercise 15:

Adding classes in .theme

A major job of any theme is to give us control over classes. We use classes to control styling, layouts, and javascript. Drupal and Twig give us many different ways to do this.

In this exercise, we are going to use `hook_preprocess_html` to gain control over the classes attached to our body tag.

Add classes to our body tag

1. Navigate to your theme root, and open your **acme.theme** file.
2. Add the following code:

```
function acme_preprocess_html(&$variables) {
// Our code will go here
}
```

3. We want to add our own classes to the body tag. This first one will be for all pages. The second one will set a class for each region that has content. Once completed, save and clear caches. Then view the source code of a page and confirm that our new class is applied to the body tag.

```
// Add 'my-class' to all pages.
function acme_preprocess_html(&$variables) {
  $variables['attributes']['class'][] = 'my-excellent-class';
}
```

4. Next, add the following line at the top of your **acme.theme** below `<?php` .

```
use Drupal\Component\Utility\Html;
```

5. Then add the following code inside of your `preprocess_html` function, below the `my-class` line

```
...
// This is the D7 equivalent of "global $theme"
$theme = \Drupal::theme()->getActiveTheme()->getName();

$regions = system_region_list($theme);

foreach ($regions as $region => $region_name) {
  $region_class = Html::getClass($region);
  if (!empty($variables['page'][$region])) {
    $variables['attributes']['class'][] = $region_class . '-active';
  }
}
...
```

We are going to use the `Html` PHP Class and its `getClass` method to process our values and make sure they are properly formatted class names. This is the replacement to drupal_html_class() and drupal_clean_css_identifier(). Because we use this, we have to make sure to declare our dependency on that class. That is why we added the code in part 4. Otherwise, PHP would not know what you are talking about and we would get a delightful PHP fatal error.

Go back to your website and refresh. Check out our body tag. Do you see the active region classes? This may be helpful if you want to trigger javascript or CSS in one region based on if there is stuff in another region.

Questions you may have…

- How would I know to use `use Drupal\Component\Utility\Html;` ?

Done ☺

Can you believe it? Exercise 16 - Form Alter

Exercise 16:

Simple form altering

A big part of Drupal are forms. For example, content entry forms to add and manage content, search forms to find content, and settings forms to control the numerous parts of our site. Default forms are pretty awesome, but sometimes they just aren't enough. Now and then we need to alter a label, default value, or add libraries for styling or cool javascript functionality.

Form API and Drupal form alters are still in Drupal 8 and are still incredibly powerful tools to control Drupal. When using a form_alter for theming it is best to keep it to simple alterations such as changing labels, adding/removing classes, editing div structure, and adding HTML5 placeholders like in our following examples. Anything else should really be done in a module form alter.

For this exercise, make sure the search block is in a region and visible.

Simple form altering with `hook_form_alter` function

1. Navigate to your theme root
2. Locate and open your **acme.theme** file, and add the following code.

```
function acme_form_alter(&$form, \Drupal\Core\Form\FormStateInterface $form_state,
    if ($form_id == 'search_block_form') {
        // Add placeholder text
        $form['keys']['#attributes']['placeholder'] = t('Search');
    }
}
```

3. Clear cache, and observe your search block.

Simple form altering with `hook_form_FORM_ID_alter` function

This function is refined to only affect the form with the matching **FORM_ID** in the function name.

1. Replace the above function with the following code:

```
function acme_form_search_block_form_alter(&$form, \Drupal\Core\Form\FormStateInter
    // Add placeholder text
    $form['keys']['#attributes']['placeholder'] = t('Search this site.');
}
```

Adding a javascript via a library to a form

Let's say we want to leverage one of the libraries from core or contrib when our form is visible.

1. Search and remove any other instances of kint() in your theme. Make sure the search_kint module is enabled.
2. View your site as an anonymous user by opening a new browser or private window. Verify that `form.js` is not included.
3. Add the following to the `acme_form_search_block_form_alter` function in **acme.theme**

```
kint(\Drupal::service('library.discovery')->getLibrariesByExtension('core'));
```

4. Search for `form.js` in the the search form provided by kint. You'll see it highlighted under parent `drupal.form` .
5. Navigate back to your theme root directory and open your **acme.theme** file.
6. Add the following code to your `acme_form_search_block_form_alter` function:

```
// Attach a library from our theme to a form.
$form['#attached']['library'][] = 'core/drupal.form';
```

7. Clear your caches.

If everything worked, the form.js file will be loaded for whenever the search block is present even for anonymous users.

Questions you may have

- What is the benefit of using **hook_form_FORM_ID_alter** rather than just **hook_form_alter**?

- What else can I do with form alters?
- Can I use `[#attached]` in any function?

Done ☺

You're on fire! Don't stop now! Exercise 17 - Responsive Images

Exercise 17:

Breakpoints and setting up responsive images

Using the core Breakpoint module, you can now define your theme's breakpoints in code. There is no UI for doing this, but we have a solid API from the Breakpoint module that allows modules and themes to define breakpoints and breakpoint groups, as well as resolution modifiers that come in handy when targeting devices with HD/Retina displays. To keep this exercise a little simpler, we won't worry about modifiers for HD displays.

Find **toolbar.breakpoints.yml** for and observe how the breakpoints correspond to the administration menu styles.

Make sure Breakpoint (breakpoint) and Responsive Image (responsive_image) modules are enabled

1. Visit your modules page, under the "Extend" button in the admin menu.
2. Enable the Breakpoint and Responsive Image module if they are not enabled

Create the breakpoints

1. Navigate to your theme root directory
2. Create a new file called **acme.breakpoints.yml**

```
$ cd MYDRUPAL
$ touch themes/custom/acme/acme.breakpoints.yml
```

3. Open the file in your preferred code editor and add the following.

```
acme.mobile:
  label: mobile
  mediaQuery: '(min-width: 0px)'
  weight: 0
  multipliers:
    - 1x
acme.tablet:
  label: tablet
  mediaQuery: '(min-width: 1040px)'
  weight: 1
  multipliers:
    - 1x
acme.desktop:
  label: desktop
  mediaQuery: '(min-width: 1200px)'
  weight: 2
  multipliers:
   - 1x
```

4. Clear cache.

Together, all of these breakpoints are referred to as a **breakpoint group**. The weight of these is crucial so that the proper image styles get swapped in depending on viewport size and screen resolution. A breakpoint's weight should be listed from smallest min-width to largest min-width. Also, note the "multipliers" section. Breakpoint allows for different pixel density multipliers for displaying crisper images on HD/Retina displays: 1x, 1.5x and 2x.

Create your Responsive image style

We now need to bring the breakpoint group information and the image styles all together. We do this with a **Responsive image style**.

1. Navigate to **/admin/config/media/responsive-image-style** and "Add responsive image style"
2. Create the style using the following information:
    - Label: `Acme Image`
    - Breakpoint group: select `Acme`
    - For each breakpoint, choose: `Select a single image style` and set the image style that you want along with each breakpoint.
        - 1X DESKTOP [(MIN-WIDTH: 961PX)] => Large (480x480)

        - 1X TABLET [(MIN-WIDTH: 601PX)] => Medium (220x220)
        - 1X MOBILE [(MIN-WIDTH: 0PX)] => Thumbnail (100x100)
    - Fallback image style: Medium (220x220) .

We now have a Responsive image style, with the breakpoints matched to an image style. Now it is time to apply our Responsive image style to an image field

Configure the display of our image field

1. Navigate to the Manage display page for the "Article" content type.
2. Change the format of the `Image` field to use `Responsive image`.
3. Select `Acme Image` as the Responsive image style we want to use.
4. Click **Update**, and then click **Save**
5. Clear caches, and then visit an article node with an image.

If you have responsive testing tools in your browser, use those to emulate different screen sizes. If not just make your window smaller until you can start to see the images changing size.

Done ☺

Almost there! Exercise 18 - Custom Theme Settings 1

Exercise 18:

Creating a theme settings file

Often we need to create small, simple little settings for our theme that can make it more reusable and customizable across sites. We can create new theme settings on our theme's settings page that we can use to help other users configure and customize our theme. For this, we will utilize two additional files. **acme.settings.yml** and **theme-settings.php** file.

1. Go to MYDRUPAL.LOCAL/admin/appearance/settings/acme in your browser to see what settings are available by default.
2. Create a new file called **acme.settings.yml** in your theme root and open in your preferred code editor.

```
$ cd MYDRUPAL
$ touch themes/custom/acme/acme.settings.yml
```

3. Add the following code:

```
features:
  copyright_holder: ''
  search_placeholder: 'Search site'
```

4. Create a new file in your theme directory called **theme-settings.php**

```
$ cd MYDRUPAL
$ touch themes/custom/acme/theme-settings.php
```

5. Add the following code to that file.

```php
<?php

function acme_form_system_theme_settings_alter(&$form, $form_state, $form_id = NULL

    $form['copyright_holder'] = array(
        '#type' => 'textfield',
        '#title' => t('Copyright Holder'),
        '#default_value' => theme_get_setting('copyright_holder'),
        '#description' => t('This appears in the footer'),
        '#weight' => -10,
    );
    $form['search_placeholder'] = array(
        '#type' => 'textfield',
        '#title' => t('Search Placeholder'),
        '#default_value' => theme_get_setting('search_placeholder'),
        '#description' => t('This appears in the footer'),
        '#weight' => -10,
    );
}
```

6. Clear cache, then go to our **Appearance** page. Click on **Settings** for your custom theme. You should see two new options for the settings we added.

7. Feel free to customize those values. We will use them in the next exercises.

Questions you may have…

- What is `features` in the ***.settings.yml**?
- Where do I find documentation for creating forms?

Done ☺

One more to go! Exercise 19 - Custom Theme Settings 2

Exercise 19:

Apply our theme settings to our custom variables

Now that we have created some custom theme settings and we can customize them through the UI, let's put them to use. We will capitalize on the functionality we created in an earlier exercise. We will alter our **preprocess_page()** to include the "copyright_holder" variable that will be customizable with one of our new theme settings.

Customize the copyright holder

1. In our **acme.theme** file, locate the `acme_preprocess_page()` function.
2. Modify the code to include the copyright_holder setting.

```
$variables['copyright'] = t("Copyright @date @holder",
  array(
    '@date' => date('Y'),
    '@holder' => theme_get_setting('copyright_holder')
  )
);
```

1. Clear caches and see if it worked. Change the value on the theme settings page if you haven't already (otherwise nothing will really change).

We have declared our copyright_holder in our theme's settings, and it will override the site name as being the default copyright holder. A simple example, but powerful.

Questions you may have…

- How can I get one of the default theme settings, like the logo?

Done ☺