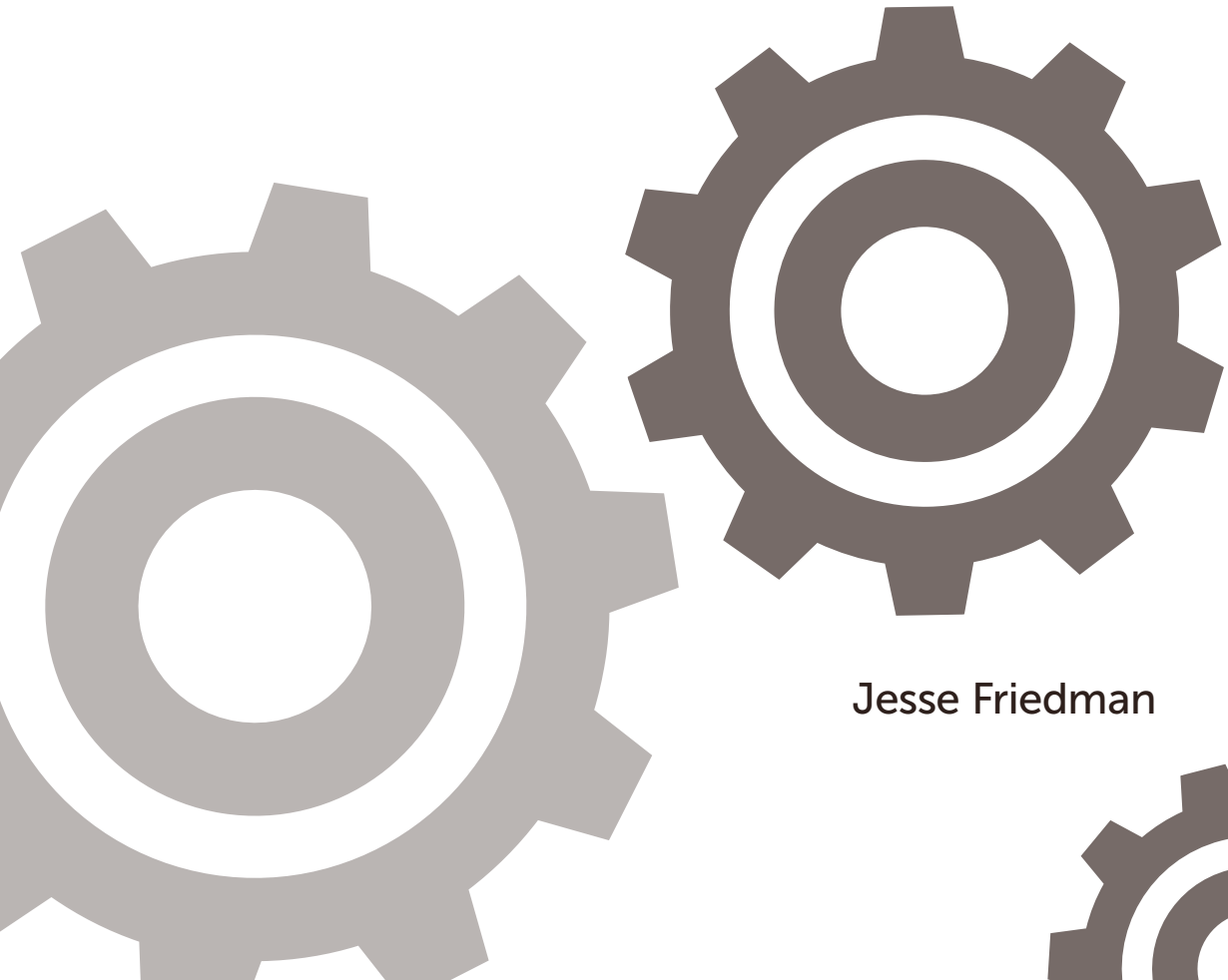
web designer's guide to wordpress

PLAN • THEME • BUILD • LAUNCH

Jesse Friedman

web designer's guide to wordpress

PLAN • THEME • BUILD • LAUNCH



Jesse Friedman

Web Designer's Guide to WordPress: Plan, Theme, Build, Launch
Jesse Friedman

New Riders
1249 Eighth Street
Berkeley, CA 94710
510/524-2178

Find us on the Web at: www.newriders.com
To report errors, please send a note to errata@peachpit.com

New Riders is an imprint of Peachpit, a division of Pearson Education.

Project Editor: Michael J. Nolan
Development Editor: Margaret S. Anderson/Stellarvisions
Technical Editor: Jonathan Desrosiers
Production Editor: David Van Ness
Copy Editor: Gretchen Dykstra
Proofreader: Patricia Pane
Indexer: Joy Dean Lee
Cover Designer: Charlene Charles-Will
Interior Designer/Compositor: WolfsonDesign

ISBN 13: 978-0-321-83281-8
ISBN 10: 0-321-83281-7

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

For my loving wife, whose unparalleled tenacity motivated me to finish this book, and for my son, who teaches me more than I could ever teach him.

Download all the code and resources for this book at
http://wdgwp.com/downloads

Acknowledgments

I'd like to take this opportunity to thank not only those individuals who directly contributed to the making of this book, but also those who have motivated, taught, and inspired me over the years.

IN ORDER OF APPEARANCE

To my family, thank you for instilling in me the value of education and the importance of knowledge. Jake, you're an amazing friend, whom I know I can count on for anything.

Hilary Mason, thank you for the years of motivating, coaching, and teaching. You are a great mentor and friend.

Patrick McNeil, you gave me an opportunity to contribute to your amazing books, which both humbled me and open many doors. I appreciate our friendship and your continued support.

Jeff Golenski, I'm proud to have mentored you in the past and thank you for your massive contribution to this book. Your designs and photographs are amazing, as was your willingness to work with me at Starbucks into those late nights and even early mornings.

Michael Nolan, just months ago we were talking about this book over a box lunch and now it's a reality. Thank you for starting that conversation and for all your help with making this book a great resource. Margaret Anderson, I really appreciate your commitment to managing this process and all the late night and weekend hours dedicated to this project. To everyone else at New Riders who had a hand in this book, you're all amazing and I appreciate all your hard work.

Jon Desrosiers, you're a fantastic developer and, as it turns out, a great tech editor. Thanks for finding all the little nuances that I would have otherwise missed.

Sara Cannon, thank you for all your contributions to the WordPress community and for writing the foreword to this book.

To all the industry veterans, WordPress power users, friends, and colleagues who wrote letters for this book, thank you very much. To the rest of the WordPress community who literally made all this possible, I cannot begin to thank you enough. To everyone who has written a patch, contributed to the codex, developed a plugin, or run an event, you're an invaluable member of a global team. And finally, here's a special shout-out to @nacin, @markjaquith, @jjj, @janeforshort, @otto42 and, of course, @photomatt.

TO MY WIFE

Joy, this is as much your book as it is mine. Thank you for everything.

Foreword

In the past few years, I've been honored to speak at dozens of WordCamps, lead WordCamp Birmingham, and contribute to WordPress Core with the user interface group. I am honored that Jesse asked me to write this foreword. I love the WordPress community.

WE ARE PIONEERS

In this age of ever-changing technology, it's important to be forward-thinking. As web designers, developers, and builders, we need tools that are reliable, faster, better, and sustainable. We desire standards, yet also crave innovation. We don't want to sit idly while technology passes us by—we want to predict what lies ahead and make it.

WE SHAPE THE FUTURE

We need to protect ourselves and our content, and forge our own paths. We have to react to a constantly shifting landscape—proprietary systems only tie us down. To be forward-thinking, we need a platform we can mold into anything we want it to be.

WE USE WORDPRESS

We use WordPress because it's secure, reliable, and adaptable. It can fit anyone's needs, large or small—from high-traffic WordPress.com VIP clients to local urban farms. WordPress isn't just for personal blogs—it's the backbone of large businesses, tight-knit communities, web apps, and everything in between. Most importantly, WordPress is free and open source. Thanks to its license, the GPL, WordPress will remain free forever. The GPL lets you build on the shoulders of others, granting you the freedom to reuse whatever you want, shape it, and publish it at will.

WORDPRESS CAN DO ANYTHING

Not many platforms can do what WordPress does. Not many have the flexibility to scale and adapt. Some have proprietary code that you can't touch and remake into your own. With WordPress, you can remake and reuse—and you're not alone. WordPress is a community: there are millions of people just like you sharing their code, ideas, and innovations.

Jesse Friedman is one of these people. In this book, he takes you step-by-step through how to leverage WordPress and its intricacies, sharing the knowledge he's gathered from years of working with this tremendous platform. Jesse not only provides you with practical standards for WordPress design and development, but also dives deeper with his own insights.

Jesse has a knack for thinking about not only the outcome, but also ongoing usage. He shows great attention to how people interact with the WordPress dashboard, and provides insight into the user-admin experience. Pioneering the future requires more than a make-it-then-leave-it mentality—it takes thoughtfully crafting experiences for everyone, while keeping an eye on the future.

This book is a tremendous resource to our community.
—Sara Cannon

Sara Cannon is an artist, designer, and developer based in Birmingham, Alabama. She is the cofounder and creative director of Range—A Design and Development Shop.

Contents

The Letters

Dear WordPress Tenderfoot,

WordPress has evolved from a simple blogging tool to a feature-rich content management system, and even a web application platform. You can use WordPress to do almost anything, although it fits certain projects more than others. I've worked with many companies and individuals, architecting and implementing websites powered by WordPress since 2005. What follows are my recommendations for deciding whether WordPress is the right tool for the task and for approaching new WordPress projects.

When evaluating WordPress as a platform for a new project, start with the *information architecture* or the data layer. It's imperative to learn as much as possible about the information to be stored, how it will be organized, updated, and searched, and the relationships between data objects.

For simple websites, posts and pages are sufficient. However, more sophisticated websites often require custom post types. For example, an event will have a title, description, date, and number of tickets available. Post types may need to relate to one another—an event is associated with a venue, which has a name, address, and phone number. Custom post types often warrant custom taxonomies such as specific product categories. Custom post types, taxonomies, themes, and additional metadata can be added to WordPress fairly easily, but it's important to map out how they will be implemented and ensure that they meet the needs of the business.

Carefully review the website's *functional requirements*. An understanding of WordPress features will help you spot gaps where additional development or plugins are needed. When choosing plugins, it's important to consider: age (newer plugins may be less stable than competing, veteran plugins), compatibility (plugins should maintain current WordPress version compatibility), support (developer should be diligent in fixing bugs and responding to issues), and documentation (installation guide should be included, as well as usage information).

If, after evaluation, you find your project resisting "the WordPress way," you may wish to explore other solutions. This includes information architecture not conforming to the post type and taxonomy paradigm, custom theme and plugin development exceeding budget or timeline, or requiring more than 50 percent customization of the core.

As WordPress becomes more extensible, it will be a faster, easier, and more economical tool to solve increasingly complex client problems. May you find success and enjoyment in your WordPress projects.

James Coletti
jamescoletti.com | @jamescoletti

WordPress Theming Basics

4

I'm often surprised to learn how complicated it is to work with other CMSs. As we've seen, WordPress has a very low barrier to entry, which means you can learn the system and build themes faster and more efficiently. At this point you know your HTML, CSS, and probably JavaScript. The only difference between a static website and a WordPress theme is stripping away that static content and replacing it with dynamic CMS calls.

I'm also surprised to meet web designers working in WordPress who don't realize they are writing PHP. PHP is the server-side web development language behind WordPress. Even if you're well aware of what it does and how it works, you probably haven't written much PHP. Well, guess what? Today we're diving right in.

What you're about to learn

- WordPress theme requirements
- Theming basics
- Dynamic header calls
- Menu nav creation
- Content formatting

Five-Minute Theme Build

You need only two template files (index.php and style.css) to create a functional WordPress theme. Index.php is used to make WordPress calls to display content, while the style.css file houses site styles and defines the theme name, description, and other details. In this chapter, we'll create a very simple WordPress theme using some basic required files.

> **NOTE**
>
> While you can technically create a WordPress theme with just two files, it is not recommended. In fact, in the future other files like header.php, footer.php, and comments.php will be required.

Theme Requirements and Declarations

Let's start by creating a blank style.css file and putting it in the theme folder. Name your theme folder something simple yet unique and don't use any spaces, numbers, or special characters.

```
my-basic-theme
    style.css
```

If you haven't installed a local server application yet, that's OK. For now we're just doing some very basic programming. However, to test the theme you'll need to install WordPress somewhere.

> **BEST PRACTICE**
>
> No two themes can have the same declaration details, as this will cause conflicts in the WordPress admin. Unique naming conventions are paramount.

The absolute first thing in the style.css file has to be the theme declarations, which are commented out to prevent interaction with actual site styles. This section is called the "stylesheet header." Below is the stylesheet header for our first basic theme. Be aware that changing this information on an activated theme is likely to cause a slight glitch and require reactivation.

```
/*
Theme Name: My Basic Theme
Theme URI: http://webdesignerguidetowordpress.com/
Description: My first WordPress theme
Author: Jesse Friedman
Author URI: http://jesserfriedman.com/
Version: 1.0

Tags:

License:
License URI:

General comments (optional).
*/
```

Feel free to replace the text in maroon with your information. The black text must be absolutely perfect and mirror what you see above. Changing "Theme Name:" to "ThemeName:" will result in a broken theme.

The next step would be to add site styles, but we're building a very basic theme so we won't be inputting any styles at the moment.

We don't actually have to add anything to the index.php file right now. Let's start by simply creating a blank file and placing it in the same theme folder as the style.css file above:

Theme Installation and Activation

That's it! You've created a WordPress theme. Now let's install it by adding it to the themes folder on the server. Upload your files via FTP to the wp-content/themes directory on the server. You can avoid FTP by "zipping" up the theme and uploading it under the "Add New" tab in Appearance and Themes.

Once the theme is uploaded you can go to Appearance → Themes and see the theme ready and awaiting activation. It's missing a thumbnail, but since we don't really have anything to take a screenshot of at the moment, we can leave it blank. We will cover how to add a screenshot to your finished theme in Chapter 19, "Test and Launch." You'll also notice that all the theme details from the stylesheet header are there, too.

Activate the theme and then navigate to the front of the website. You'll see a very simple site, with no content. Who says we shouldn't be proud of a plain white screen?

The Next Half Hour

The theme is still quite bare but that's OK—we're going to add to it right now. By default, all WordPress installs start with one post, one category, one page, and one comment. It makes sense at this point to go in and add a few extra pages, posts, and other content to make it easier to test the theme's functionality.

Now that we have some content to work with, let's identify some WordPress theme basics. A typical website, WordPress–powered or not, will have branding, navigation, and site content, and all of these will be written in HTML.

If you head over to http://wdgwp.com/downloads you can download a very simple HTML file that has some basic markup and content we can use to create our theme. Copy and paste this file's contents into the index.php.

> **NOTE**
>
> Anytime you edit the index.php or any other template files, you'll have to upload them to the server unless you're working locally. That's one benefit of running a local server application.

There's no need to reactivate the theme each time you change the template files. Refreshing the page will show your changes. Now that you've uploaded the index.php, let's visit the front of the site. You'll see the content in place (remember, it won't be pretty just yet). Technically, we have a working web page at this point, but we still haven't made anything dynamic.

NOTE

Begin the theming process by building a static version of the theme with all the HTML, CSS, and JavaScript in place. Next, instead of building every page, focus on page templates (pages that have a different layout or unique functions, or in some way require a separate template). Finally, replace static content with WordPress calls and, just like that, you have a working theme.

The file has elements you're familiar with: title, navigation list, headings, and text within HTML tags. Next we'll replace the content, such as "Jesse Friedman | Developer," with dynamic PHP calls.

So, the document title "Jesse Friedman | Developer" will be replaced with a call to display the site title and description. Again, for now this is all placeholder text. Once you replace this content with dynamic calls, the content will automatically be replaced with content from your WordPress installation.

The navigation list will be powered by a simple menu, which we'll define shortly. After that you'll see several posts displaying only the title and content with a link to the full article, all of which will be replaced by the infamous WordPress Loop.

Now that we've defined the content, we can get to work replacing it all with dynamic calls. Let's start at the top of document in the <head> section and work our way down. In the head we have to make some minor tweaks. The <title> tag defines the title of the page you're currently viewing in the browser window.

<title>

The HTML we copied from the supplied HTML document currently looks like this:

```
<title>Jesse Friedman | Developer</title>
```

The first thing to realize about converting static content to WordPress calls is that we're simply calling PHP functions that will be replaced by content. It's easy to learn WordPress calls without really having a full understanding of PHP. This is why it's easy for web designers to build WordPress themes without really knowing that they're writing PHP.

BEST PRACTICE

WordPress performs amazingly well on the SEO front. This is attributed to the heightened level of control of all HTML elements with dynamic WordPress calls. It's a great advantage to be able to create a template for dynamic data to replace the <title></title> tag content. This ensures rich, SEO-friendly title tags, metadata, and more on every web page.

To make the title dynamic we'll delete the content between the <title> tags and replace it with the bloginfo() function.

```
<title><?php bloginfo(); ?></title>
```

Any and all PHP functions, scripts, or code in general must start and end with <?php ?>. In some cases you can get away without the closing ?> but for now let's keep it in place. The bloginfo() function requires a parameter so it knows what you're asking for. Once it receives that parameter it will "echo" it. Below is a list of parameters that bloginfo() accepts and what they will return:

```
name = Site Title
description = Site Description
admin_email = admin@example.com

url = http://example/home (however you should use home_url('/') function
      instead)
wpurl = http://example/home/wp (however you should use site_url('/') function
      instead)

stylesheet_directory = location of theme files in wp-content
stylesheet_url = http://example/home/wp/wp-content/themes/child-theme/
      style.css
template_directory = http://example/home/wp/wp-content/themes/parent-theme
template_url = http://example/home/wp/wp-content/themes/parent-theme

atom_url = http://example/home/feed/atom
rss2_url = http://example/home/feed
rss_url = http://example/home/feed/rss
pingback_url = http://example/home/wp/xmlrpc.php
rdf_url = http://example/home/feed/rdf

comments_atom_url = http://example/home/comments/feed/atom
comments_rss2_url = http://example/home/comments/feed

charset = UTF-8
html_type = text/html
language = en-US
text_direction = ltr
version = 3.1
```

Now all we have to do is enter the parameter and we'll be done. The parameter goes between the parentheses in single quotes.

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

The above code displays the site name set in the General Settings section of the WordPress admin.

NOTE

In PHP "echo" refers to outputting content so it's visible by a user on the screen. It simply prints the content wherever the echo is called, so it's still your job to place it between HTML tags for proper formatting.

You can read more about the bloginfo() function and learn all the possible parameters at http://wdgwp.com/bloginfo. As we continue through this book you'll get the hang of using WordPress/PHP functions and parameters.

Let's hop over to the front end of the site and refresh it. The title in the browser window will now mirror what you've entered into the settings.

<style>

The next thing to do is link the stylesheet in the header. Since the theme can be used on any domain, don't try to link to the stylesheet through an absolute link. Instead, replace its location with a WordPress call like we did above. Currently, the style.css call looks like this:

```
<link rel="stylesheet" type="text/css" href="style.css" >
```

All you need to do is replace the text in the href=" " with the call to the stylesheet. Since every theme requires a style.css file, you can link to it directly using <?php bloginfo('stylesheet_url'); ?>. If you want to link to additional stylesheets, JavaScript, or other files in the theme, you can use <?php bloginfo('stylesheet_directory'); ?> followed by the location and name of the file in the theme:

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

<header>

In the body there are a few elements that need replacing. Again, this results in a very simple theme, so we don't have sidebars or even a footer in this example. A quick glance at the static content shows that we have to replace the site name in the <header> ➜ <h1> along with the <nav> with a dynamic menu. Following that we have the ten most recent posts, each in its own <article>.

Below is the static content used in the <header>:

```
<header>
  <h1>Jesse Friedman | Developer</h1>
  <nav>
    <ul>
      <li><a href="">Home</a></li>
      <li><a href="">About Us</a></li>
      <li><a href="">Services</a></li>
      <li><a href="">Portfolio</a></li>
      <li><a href="">Contact Us</a></li>
    </ul>
  </nav>
</header>
```

Let's start by replacing the text within the <h1> with dynamic calls as we did above. The first half uses the 'name' parameter and the second half of the <h1> is replaced with the site 'description.' At this point you should be getting used to replacing HTML static content with dynamic calls. It's a very straightforward process—don't let it scare you.

```
<header>
  <h1><?php bloginfo( 'name' ); ?> | <?php bloginfo( 'description' ); ?></h1>
  <nav>
    <ul>
      <li><a href="">Home</a></li>
      <li><a href="">About Us</a></li>
      <li><a href="">Services</a></li>
      <li><a href="">Portfolio</a></li>
      <li><a href="">Contact Us</a></li>
    </ul>
  </nav>
</header>
```

Next, we'll call a menu by its name to replace the list of navigational items. There are lots of parameters you can use to customize this section, but for this example let's keep it simple.

Menus

The one caveat with menus is that you have to turn them on. To do this, we'll have to deviate from our index.php file and create a functions.php file. The functions.php file lives in the

theme in the same directory as the index.php and style.css. This is important: As with many template files, they must reside in the main theme directory.

Put the code below in your functions.php file.

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

Once you've implemented this code you'll see menus in the Appearance section in the admin. If you haven't already, go into menus and create a new navigation menu. Call it "Main Nav," add some pages to it, and save it.

Now we'll replace the <ul> with a function to call the menu by name. Later in the book we'll look at what this means in greater detail, but for now, just know that we're calling wp_nav_menu() function and passing it an array of parameters, in this case, the menu name.

```
<header>
  <hl><?php bloginfo( 'name' ); ?> | <?php bloginfo( 'description' );?> </hl>
  <nav>
    <ul>
      <?php wp_nav_menu( array( 'menu' => 'Main Nav' ) ); ?>
    </ul>
  </nav>
</header>
```

The above code replaces the static <header> content with dynamic content. Go into General Settings and Menus, change the content, rearrange some nav items, and get used to seeing the content change dynamically.

The Loop

The Loop is one of the more complex elements to learn, but fear not—we'll cover it in detail now. If you take a look at the static content, you can see that the same structure is repeated over and over: opening tags, title, content, closing tags. In other words, the HTML tags for each post are exactly the same, the only difference is the *content*.

```
<article>
  <h2><a href="" title=""><!-- title --></a></h2>
  <p><!-- content --></p>
</article>
```

So, instead of a repetitive list of umpteen posts with the same structure, our template will have one loop that presents the content dynamically.

The Loop is widely known among WordPress developers as the engine behind WordPress blogs (http://wdgwp.com/loop). It runs through the markup structure, template tags, and PHP code for every available post (based on your location in a site) and formats and displays them. Any HTML or PHP placed inside the loop is repeated instead of duplicated (included) as many times as the loop runs. In most places, The Loop lists up to ten posts, but this can be changed in the reading settings or in a more advanced solution right in The Loop (we'll discuss this further later on).

For now, think of The Loop as a PHP *while* loop (which it is) that calls functions along the way. If you're on the home page, it will display the ten most recent posts in the blog. If you're on a category page, it will simply display the ten most recent posts from that category. While what is being displayed changes, The Loop itself does not, because the visitor's location, or better yet the URL, dictates what will be shown.

Here's a look at a basic WordPress loop:

```
<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
  <!-- content here -->
<?php endwhile; else: ?>
      <p><?php _e( 'Sorry, no posts matched your criteria.' ); ?></p>
<?php endif; ?>
```

Let's break this down:

<?php if(have_posts()) is utilizing a simple PHP if statement to test whether the have_posts() function will return a value. If it does, then we know we have posts and we move on.

: while(have_posts()) initiates the PHP while loop using the number returned by the have_posts() function. So if the have_posts() function returns the number 10, then the loop will run ten times. Finally, we call the the_post() function, which retrieves the post data and other things.

The PHP while loop loops all the content and calls all the functions we place inside it. After that we end the loop with <?php endwhile; then call the else: ?> statement, which gives us an opportunity to do something if we don't have any posts to display.

In this case, the else statement simply echoes, "Sorry, no posts matched your criteria."

NOTE

The _e(); function echoes its parameter passed through the translation filters. Read more about _e() at http://wdgwp.com/_e.

Now that we've broken down The Loop, let's put it back together. We'll start by replacing <!-- content here --> with static HTML content as seen below:

```
<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
  <article>
    <h2><a href="<?php the_permalink(); ?>" title="">This is an Article
        Title</a></h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi nulla
        nisi, adipiscing eu laoreet vitae, venenatis vitae velit. Phasellus
        euismod dapibus velit in laoreet. Vivamus ornare justo vehicula felis
        scelerisque non aliquam nisl semper. Curabitur nisl mauris, posuere
        sed imperdiet vel, cursus id dolor. Suspendisse varius consequat
        lorem ac luctus. Maecenas consectetur neque at turpis elementum vitae
        eleifend sem blandit. Nullam auctor, risus nec porta lacinia, ante
        sapien bibendum massa, a semper tortor odio in nunc.</p>
  </article>
<?php endwhile; else: ?>
    <p><?php _e( 'Sorry, no posts matched your criteria.' ); ?></p>
  <?php endif; ?>
```

Now let's replace the static content with WordPress calls, starting with the content within the <h2>:

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

The first step was to replace the article title with the_title();. This function displays the post title that we're currently looping through. Next we'll link to the article using the the_permalink(); function:

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

We also need to input something in the title attribute of the <a>. I like to use a mix of static content with the post title. Here I want the title to be "For More Info on <!-- article title --> Click Here":

```
<h2><a href="<?php the_permalink(); ?>" title="For More Info on
        <?php the_title_attribute(); ?>"><?php the_title(); ?></a></h2>
```

The last thing to do is call the article content. Currently, we're using a static <p> to house the content. In all likelihood, the content area will be made up of all sorts of content and HTML tags, including images and videos. Anything we put into the content editor will be displayed here when we call the the_content() function:

```
<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
  <article>
    <h2><a href="<?php the_permalink(); ?>" title="For More Info on <?php
        the_title_attribute(); ?>"><?php the_title(); ?></a></h2>
    <?php the_content(); ?>
</article>
<?php endwhile; else: ?>
      <p><?php _e( 'Sorry, no posts matched your criteria.' ); ?></p>
  <?php endif; ?>
```

That's it! We've completed our loop and our theme is now functioning and dynamic. The index.php should now look like exactly like this:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title><?php bloginfo(); ?></title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
       <link rel="stylesheet" type="text/css" href="<?php bloginfo(
       'stylesheet_url' ); ?>" >
  </head>
<body>
<header>
  <hl><?php bloginfo( 'name' ); ?> | <?php bloginfo( 'description' ); ?> </hl>
  <nav>
    <ul>
      <?php wp_nav_menu( array(' menu' => 'Main Nav' ) ); ?>
    </ul>
  </nav>
</header>
  <section>
  </section>
<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
  <article>
    <h2><a href="<?php the_permalink(); ?>" title="For More Info on <?php
        the_title_attribute(); ?>"><?php the_title(); ?></a></h2>
    <?php the_content(); ?>
```

continues on next page

```
</article>
<?php endwhile; else: ?>
      <p><?php _e( 'Sorry, no posts matched your criteria.' ); ?></p>
  <?php endif; ?>
    </section>
</body>
</html>
```

Once you understand what each WordPress call does, you can make more sense of the above code. At that point, you're like Neo from *The Matrix*, seeing Matrix code rather than people. This is a very clean and easy-to-read template, and the beauty of The Loop is that it displays the right content for each and every page you're currently viewing. Don't believe me? Start navigating your site—you'll see the content change based on the URL and the index.php template page will power everything, whether you're on the home page, a single post, or even on a Search Results page.

Let's take a break. In the next chapter, it'll be time to buckle up because this was the easy stuff.

What's Next

In the next chapter, we are going to take an in-depth look at all the template design and development files we'll be using for our theme. Instead of working with functions completely out of context, the following chapter starts us down a path of building out a complete theme from beginning to end.

Dear WordPress Pupil,

At Big Sea, we work with businesses big and small for which WordPress is an ideal content management solution. We've established the following processes to help our clients make the most of their websites:

1. During the wireframing process, we indicate which sections of the site will be editable through WordPress, so clients don't assume they can change *everything*. We use color-differentiation in our wireframes to show exactly what they'll be able to change later.

2. We use custom post types liberally. Before WP version 3, we had to direct our clients to use categories and tags—or worse, plugins—to create content like employee profiles, testimonials, or locations. Now, we set up simple custom post types with appropriate names and our clients know exactly where to go to edit their Employees or Locations information.

3. We use Mark Jaquith's WP Help plugin to provide screencasts and documentation. This simple plugin creates a custom post type that displays a Publishing Help tab under the Dashboard in the WordPress admin. We create short screencasts to walk through any custom features. We upload them to our Vimeo+ account and embed them in a post in the Publishing Help. This provides access to that refresher anytime the client needs it.

4. We record "general" screencasts that we can reuse for all clients. These cover basics like creating new posts, editing content in the visual editor, setting a featured image, and even best blogging practices.

5. We include documentation and a one-hour walk-through in every project proposal. We do the walk-through when the system is about 90 percent done, so the client can add content while we finish the design details that come with filling out the site's actual content.

These steps take a little time up-front, but they save time, hassle, and frustration in the end. Arm your clients with as much help as you can, so they actually use their websites instead of letting them sit stagnant.

Andi Graham
bigseadesign.com | @BigSea

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y