Mastering the Fundamentals of WordPress Themes

How Themes Work, and How to Make Them Work For You

Brought to you by

WPSHOUT

Sponsored by

PRESS UP

good people making great websites

Contents

Introduction

WordPress is the most powerful website builder and content management system in the world. There, we said it.

Much of that power comes from WordPress's thorough and ingenious use of themes, which can give users a huge head start toward creating exactly the site look-and-feel they're after.

But there's more to themes than choosing them, setting them up, and selecting theme options. The theme paradigm offers a *lot* of power and flexibility to people with a bit of technical savvy: WordPress developers and empowered WordPress site owners.

Mastering the Fundamentals of WordPress Themes aims to help you become one of them. **We'll be lifting the hood on WordPress themes**—so you can start to see how they work, how they fit together, and how you can build and modify them to meet your own needs.

We'll be covering the following:

- **Core principles:** what WordPress themes are, what they do, and how they work in their fundamentals.
- **Key features:** elements like post template tags and the `get_template_part()` function that will hugely improve and streamline your work with themes.
- **Best practices:** basic do's and don't's that will help you make the right decision for themes you build or modify.

Let's get started!

Demystifying “The Loop” in WordPress

One of the first things that typically stumps people trying to make WordPress themes is the concept of “The Loop.” It’s *the core concept* of WordPress, so wrapping your head around it is definitely worth the upfront time cost. Our goal here is to make the concept clear first, and then to move on and show you how the code works.

“The Loop” in Plain English

In this section, I promise, no programming. We’ll just go over basic concepts.

The main building blocks of of WordPress are “posts.” The terminology here gets a bit confusing, because WordPress by default has multiple post types, one of which is called “Posts,” and the other called “Pages.” For the rest of this piece, I’m going to use “posts” to mean it in the WordPress sense of “a long block of HTML saved in a database,” which includes Posts, Pages, and any other post type you can dream up.[1]

WordPress is built around the idea that you’re writing and storing posts (just one last time: “Pages” are also posts) and it’s going to display your posts to the world on your website. This is where The Loop comes in. **The WordPress Loop is the method WordPress uses to display posts.**

Inside “The Loop,” you specify how you want those posts to be laid out. WordPress will reuse the format you specify for all the posts it’s going to display on a given webpage. If there’s only one post on the webpage—which is what we get when someone clicks onto a section of your site containing either a page or just a single blog post—WordPress will still use the core concept of The Loop to process the (single) post into what you see when the webpage loads.

[1] As a last, hopefully unnecessary, language note: a “webpage” is not the same thing a WordPress post. Rather, it is a single complete screen of HTML content as loaded by your web browser. The Google homepage, for example, is a webpage—as is the homepage of your WordPress site.

Read the rest of this chapter and then come back to the following statement: “‘The Loop’ processes WordPress posts in order to display them to users as webpages.” It should make sense.

The use of The Loop is more obvious on your homepage, or your main blog page, where WordPress cycle through, say, ten blog posts, reusing the format we've specified. The WordPress Loop is an "iterator"—programming jargon for something that repeatedly does something. Specifically, **The Loop iterates over all the posts brought back from the database.** No matter how many posts there are going to be—1 or 100—you'll always put the basic skeleton of the iterator in place, and WordPress will repeat it across all the relevant posts.

The Minimal Loop

```
<?php
if ( have_posts() ) {
	while ( have_posts() ) {
		the_post();
		//
		// Post content here
		//
	} // end while
} // end if
```

```
<?php
if ( have_posts() ) :
	while ( have_posts() ) :
		the_post();
		//
		// Post Content here
		//
	endwhile;
endif;
```

Two examples at once? Have I gone crazy?! No. I list them side by side because both examples above are identical in functionality, they just use different symbols to communicate it. You'll see both formats frequently as you poke around WordPress themes, so I've included them both. Essentially, both `if () {}` and `if () : endif;` are ways we program conditions. Like many languages, PHP supports different syntaxes to specify conditional behavior. In the left example we're saying, "If the condition inside the parenthesis is true, do what's between the curly brackets." In the right case, we're saying "If the condition in the parentheses is true, do what comes after the colon and before we tell you to stop with `endif`."

From the top—computers read just like we do—the first question of The Loop is a simple one: "Are there posts to display?" If there aren't, WordPress will move on and ignore the rest of The Loop (jumping past the `}` or `endif;`). If there are, then we proceed onward, where we encounter `while`. A `while` and an `if` have a lot of common, so if you understood the last paragraph you're most of the way there.

The difference about `while` is that unlike an `if`, which asks its question once and then continues on, `while` continues to do what's between the curly braces (or, in the example on the right, between `:` and `endwhile`) until its condition stops being true. What that means, in English, is that now that WordPress knows it has at least one post(s), it will keep "iterating" the stuff in the `while` loop until it's out of posts.

OK, one last issue: the stuff inside the `while` loop. There's not much there in our example, because we're keeping it light and only one thing *must* be in your `while` loop. That's the `the_post()` "function call." A stack of posts has come into the page, was verified to exist by the `if`, and verified to have stuff in it by the `while`. Now, `the_post()` is the worker who takes each post in that stack, one by one, and makes it ready to use.

In our giant factory, `the_post()` is the truck-unloader who pulls one giant chunk of raw steel out so that we can start to fashion it into our giant gear. Because he's only called once, he just unloads one piece, makes it ready for work, and sends the truck back around to the `while` loop. If the `while` inspector finds something more to work on in the truck, she'll send the truck back around and `the_posts()` will take off another piece. If not, the truck will continue on and exit "The Loop." This cycle will always continue until the truck is empty.

An Example of The Loop in Action

My hope is, after reading the functioning of the The Loop in programming-jargon-less terms, and then with the minimal programing jargon, you'll be able to understand a real-world example. This example might be found in a *very basic* theme's `index.php` file.

```
<?php if (have_posts()) { ?>
	<?php while (have_posts()) : the_post(); ?>
		<li><a href="<?php the_permalink(); ?>"><?php the_title();
?></a></li>
	<?php endwhile;
} ?>
```

One of the first things to realize about above example is that in the above section, everything we had was PHP, but now we're putting some HTML right along with it. Things that are between `<?php` and `?>` are considered valid PHP logic when run on the server; things that aren't are just controlled by the surrounding PHP logic. That is: **If the PHP that surrounds our HTML says the HTML should show up on the final page, it will. If it doesn't, it won't.**

Now a few things about the example are a bit weird. First, for learning's sake, I've declared my `if` condition with curly braces and my `while` condition with a colon and an `endwhile`. This will function perfectly, it's just a bit confusing to the reader, so best not to do it in real life. Similarly, in the last section we had `the_post()` on its own line and here it's sharing the line with the existing `while` condition. Again, this is totally functional PHP and will work without problem. It's just an example of the ways in which real-world uses of The Loop may vary.

The truly new stuff though, is that middle line, the one that starts `<li>`. The HTML tags `<li></li>` surround “list items,” just as `<a></a>` surrounds a link.

Essentially, in WordPress, you use functions like `the_permalink()` and `the_title()` to get data about the current post. These functions are called template tags (check out our later chapter on them!), and they just output the data that they describe right into the HTML that surrounds them. So once filled in by The Loop, each post that is being iterated over in this loop would spit out something like:

```
<li><a href="http://pressupinc.com/blog/2013/07/demystifying-the-loop-in-
wordpress/">Demystifying "The Loop" in WordPress</a></li>
```

This is a list (`<li></li>`) containing a link (`<a></a>`) that points to the post using `the_permalink()`, and displays `the_title()`. Many other WordPress template tags do exist, like `the_content()`, which displays the contents of the post, and `the_excerpt()`, which displays either the first part of the post, or the excerpt that is manually set for the entry.

Onward to New Heights

There are no doubt details about the The Loop I failed to explain sufficiently. But I hope I gave you a good simple overview of the how the whole thing works to transform posts into webpages. Understanding The Loop is one of the most powerful pieces in starting to understand how WordPress works.