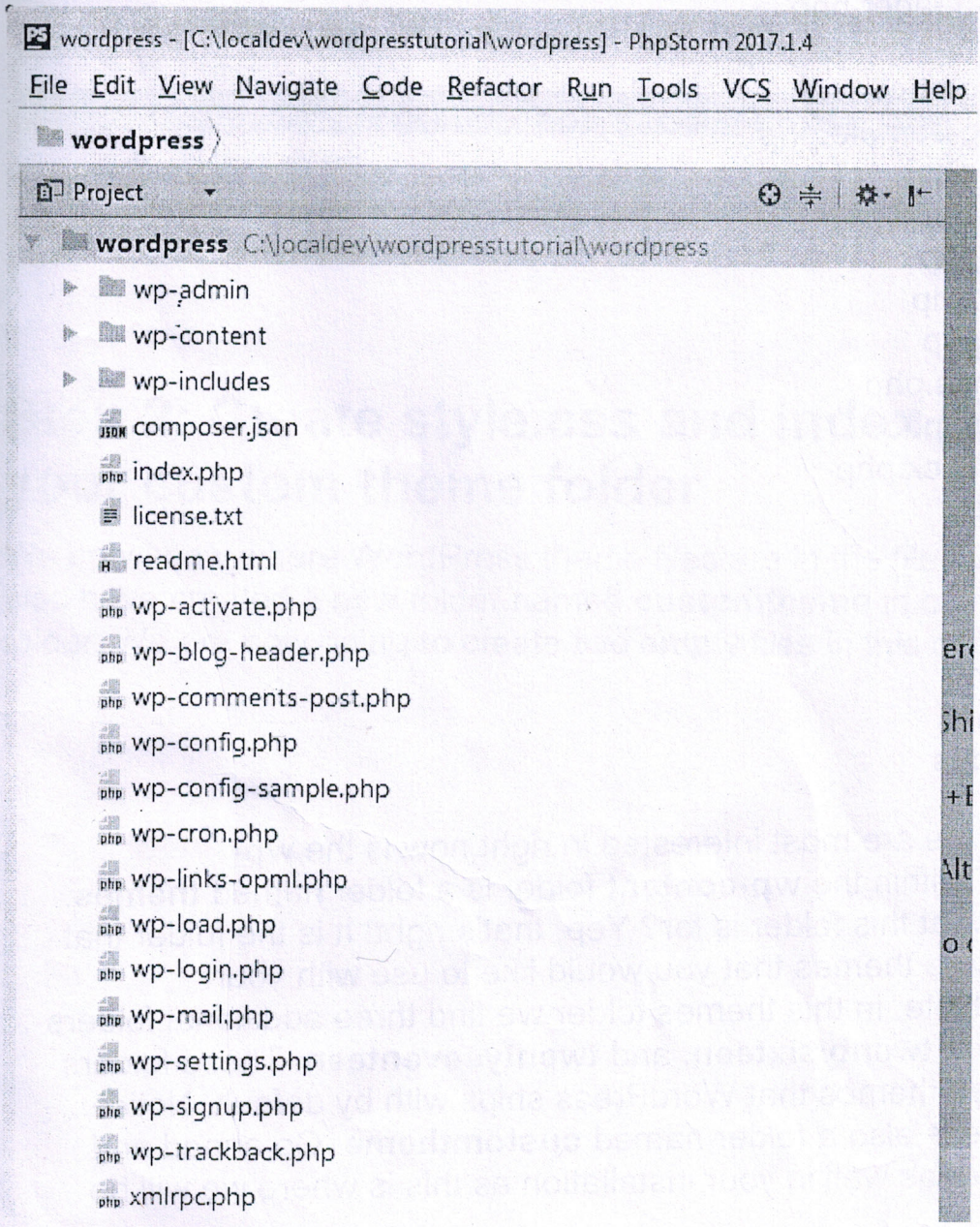
Step 1: Create a folder to hold your theme files.c

If we are going to be building themes, we need to know where the files that make up a WordPress theme live in a WordPress Installation. This is pretty easy. We know that a WordPress installation typically has a root directory named `wordpress`. Here is what our root directory looks like in PHP Storm.



This directory contains the following files and folders:

Files

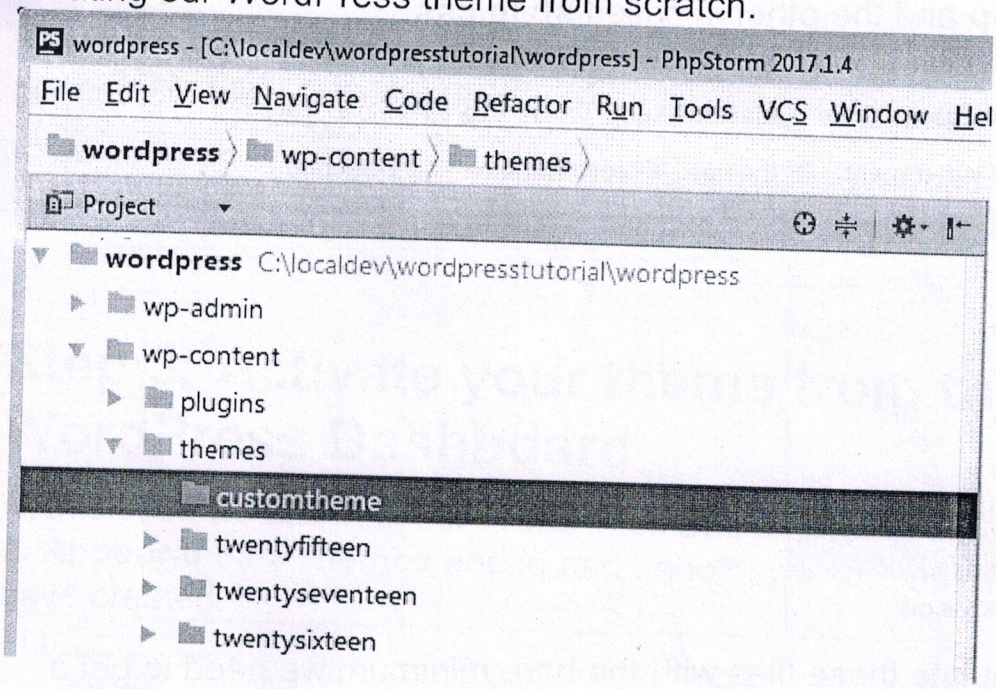
- composer.json
- index.php
- license.txt
- readme.html
- wp-activate.php
- wp-blog-header.php
- wp-comments-post.php
- wp-config.php
- wp-config-sample.php
- wp-cron.php
- wp-links-opml.php
- wp-load.php
- wp-login.php
- wp-mail.php
- wp-settings.php
- wp-signup.php
- wp-trackback.php
- xmlrpc.php

Folders

- wp-admin
- wp-content
- wp-includes

The folder that we are most interested in right now is the **wp-content**folder. Within the **wp-content** folder is a folder named **themes**. Do you know what this folder is for? Yep, that's right! It is the folder that holds one or more themes that you would like to use with your WordPress website. In this themes folder we find three additional folders of **twentyfifteen**, **twentysixteen**, and **twentyseventeen**. These folders contain the three themes that WordPress ships with by default. Notice below that there is also a folder named **customtheme**. Go ahead and create that folder as well in your installation as this is where we will be

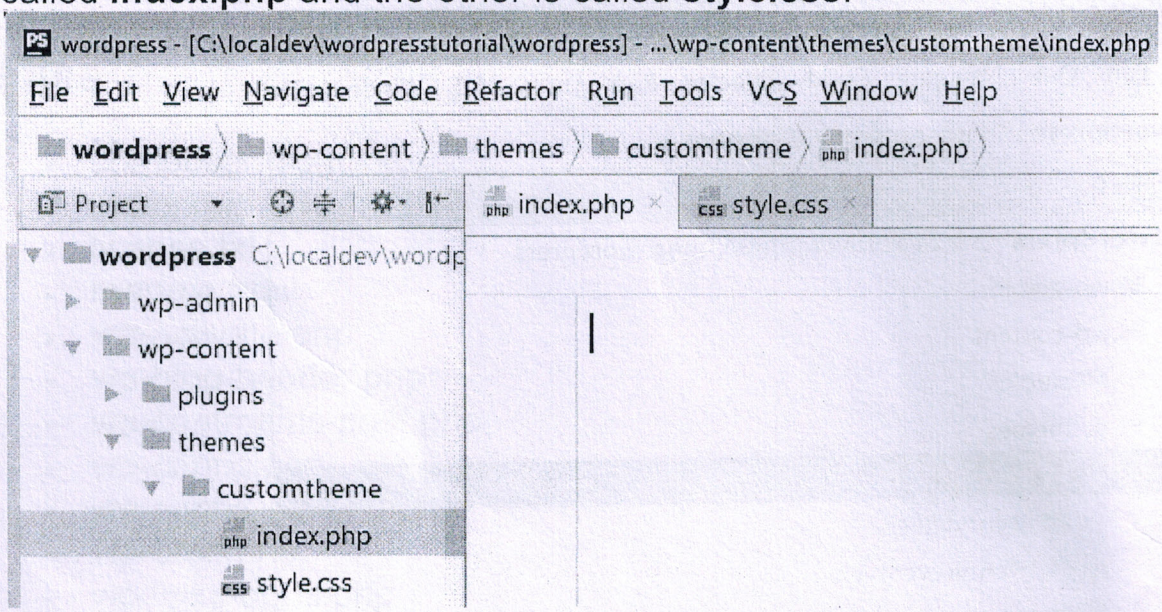
creating our WordPress theme from scratch.



Step 2: Create style.css and index.php in your custom theme folder

We now know where WordPress theme files are in the file system. We also have created a new folder named **customtheme** in our **themes** folder. We are now going **to create two empty files in this directory. One**

called **index.php** and the other is called **style.css**.



Let us now populate these files with the bare minimum we need to get a **new theme going in WordPress.**

style.css

WordPress actually reads the comments that you place in the style.css file. This is where you specify specific information about the theme you are building.

The style.css is a stylesheet (CSS) file required for every WordPress theme. It controls the presentation (visual design and layout) of the website pages.

In our snippet here we simply assign a Theme Name, the Author, the Author URI, and the Version number of our theme.

```
1 /*
2 Theme Name: customtheme
3 Author: Vegibit
4 Author URI: http://vegibit.com
5 Version: 1.0
6  */
```

index.php

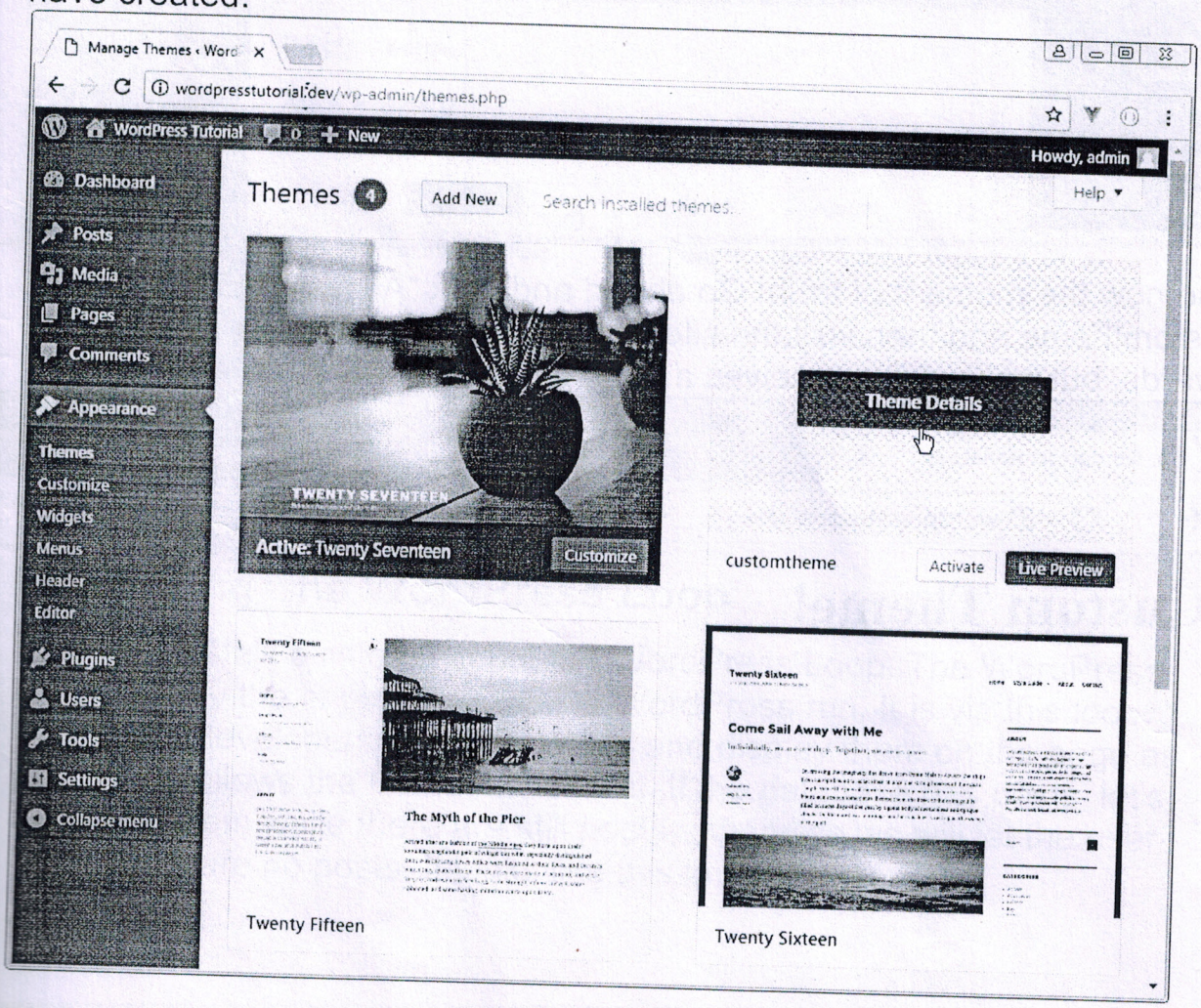
In this file, we just want to output something to the screen to prove that our custom theme is working.

```
1 <h1>Custom Theme!</h1>
```

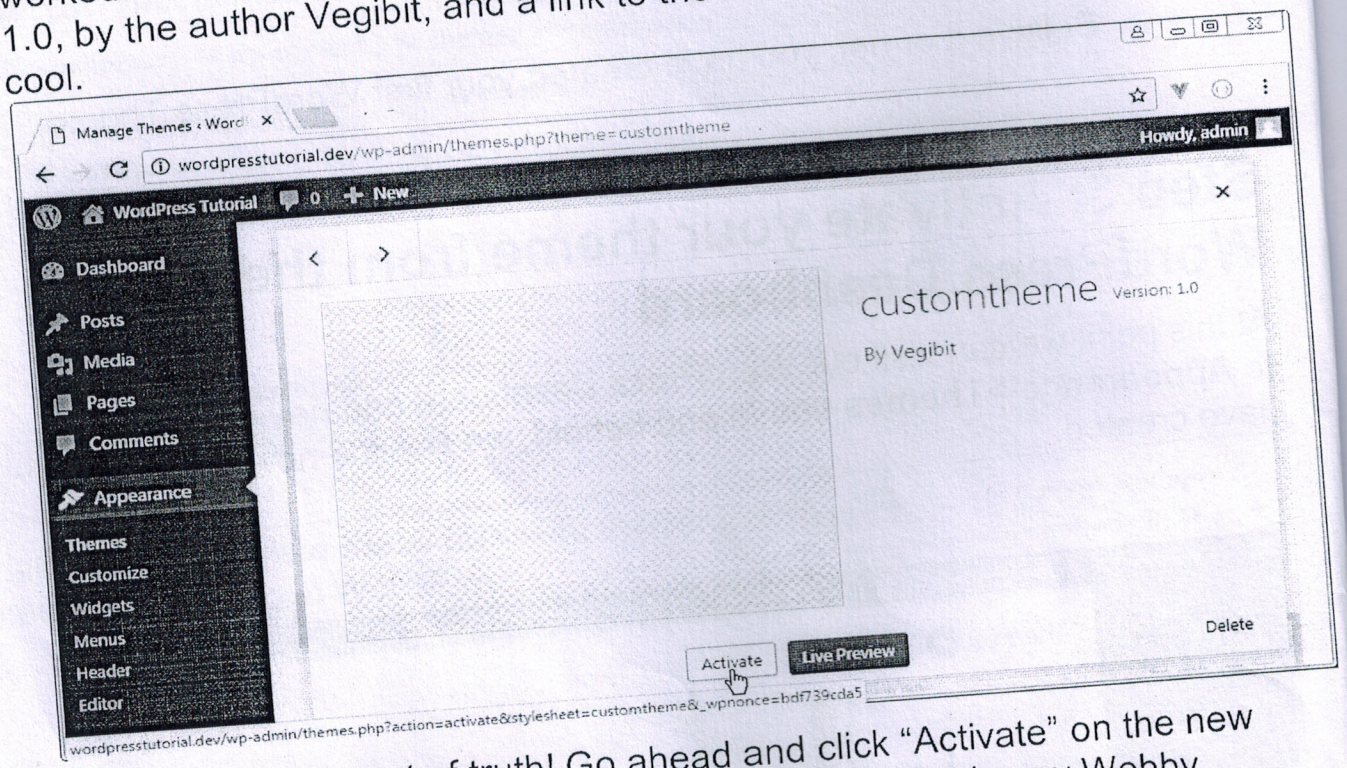
Believe it or not, you have created your first WordPress Theme.

Step 3: Activate your theme from the WordPress Dashboard

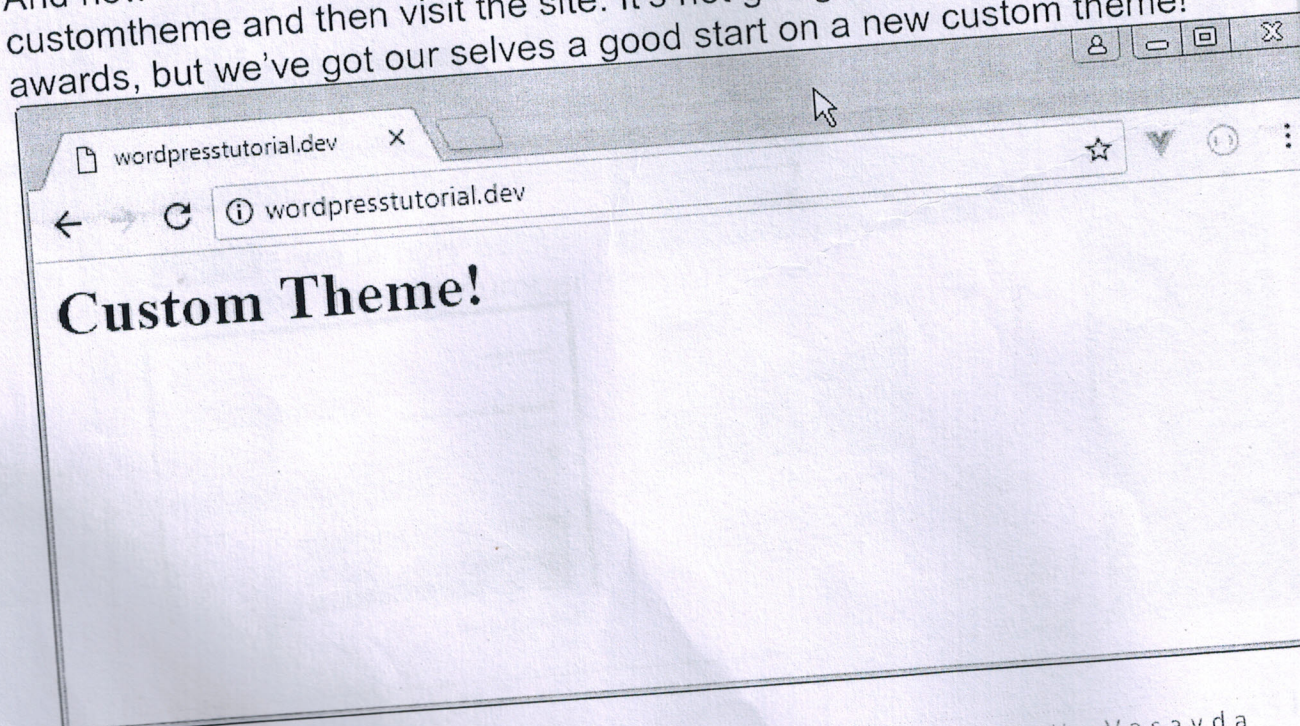
At this point we can visit our WordPress Dashboard and navigate to **Appearance->Themes** and lo and behold, we see the new theme we have created.



We can click on “Theme Details” to drill down on our custom theme and find that the information that we had entered into the style.css file has worked. We can see the them has a name of customtheme, with Version 1.0, by the author Vegibit, and a link to the URI we had provided. Very cool.

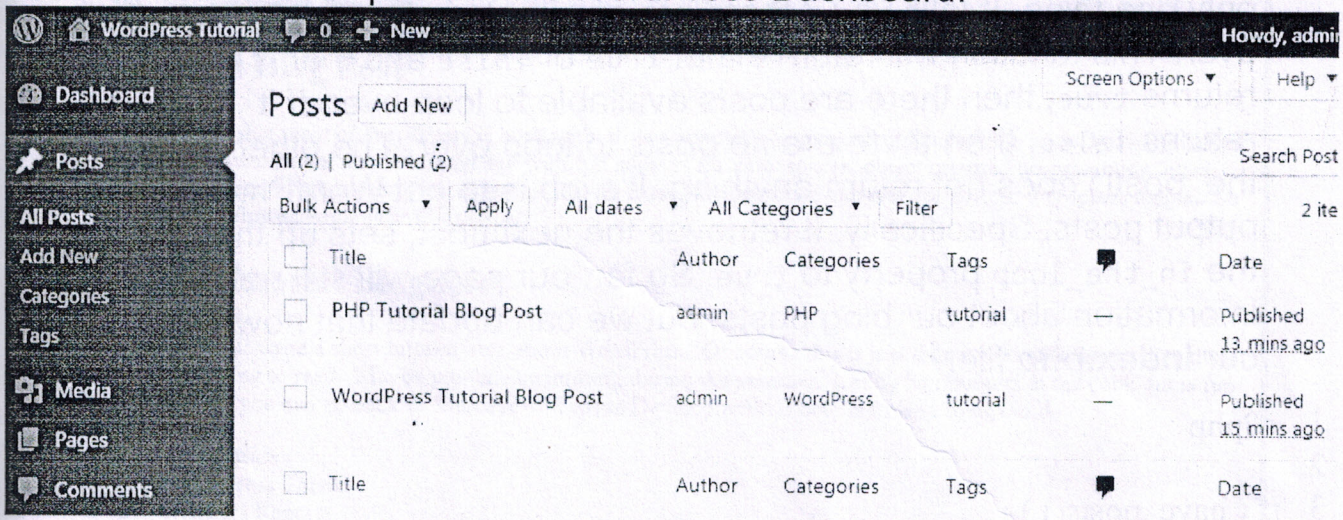


And now the moment of truth! Go ahead and click “Activate” on the new customtheme and then visit the site. It’s not going to win any Webby awards, but we’ve got our selves a good start on a new custom theme!



Step 4: Add Code to Output The Post Title and Post Text

We've take the liberty to populate a couple of example posts in the database so we can work with that data during this tutorial. Right now, our theme just outputs **Custom Theme!** to the page when we visit our site no matter how many posts are in the database. Let us now move to fetching some data from the database, and outputting it to the page. Specifically, we want to fetch the Post Title and Post Content of all posts and view them on the homepage. Let's give that a shot now. First let's see what we have for posts in the WordPress Dashboard.



Leveraging The WordPress Loop

Now we can talk a little bit about the WordPress Loop. The WordPress Loop is really the engine that makes WordPress run. It is via this loop, that theme developers check for posts and display them on the page as needed. If follows the format as follows. If the database has posts, let's loop over them while there are still posts, otherwise we will let the user know there are no posts. It looks like this in PHP code.

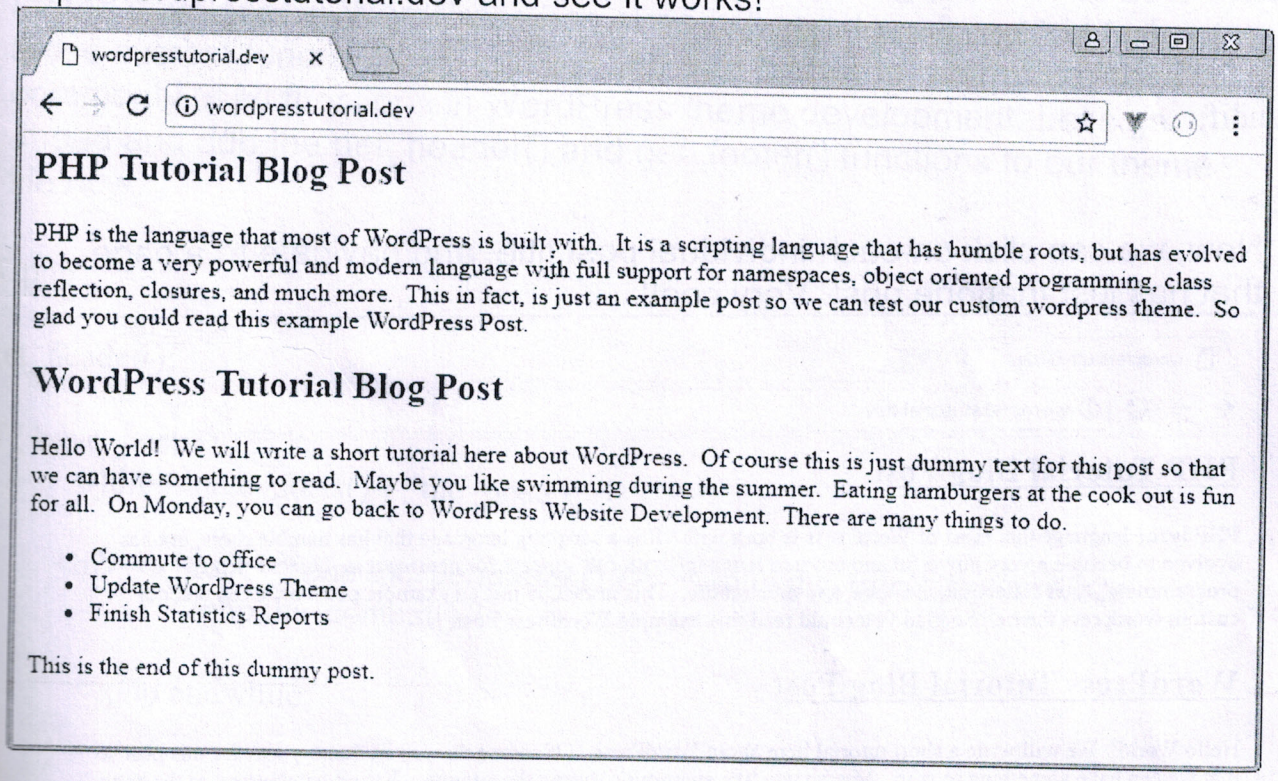
```
<?php
```

```
3  if ( have_posts() ) :
4          while ( have_posts() ) : the_post(); ?>
5
6          <?php endwhile;
7
8  else :
9          echo '<p>There are no posts!</p>';
10
11 endif;
12
13 ?>
```

Notice that The Loop makes use of two functions in it's most basic form. Those are have_posts() and the_post(). The have_posts() function does only one thing. It tells you if there are any posts in the database to loop over. This function will return either `true` or `false` and *that is it.* If it returns `true`, then there are posts available to loop over. If it returns `false`, then there are no posts to loop over. The other function, the_post() does not return anything. It's job is to get WordPress ready to output posts. Specifically, it retrieves the next post, sets up the post, sets the `in_the_loop` property to `true`. So far, our page will still not output any information about our blog posts, but we can update that now in our **index.php** file.

```
1  <?php
2
3  if ( have_posts() ) :
4          while ( have_posts() ) : the_post(); ?>
5
6          <h2><?php the_title() ?></h2>
7                  <?php the_content() ?>
8
9          <?php endwhile;
10
11 else :
12         echo '<p>There are no posts!</p>';
13
14 endif;
15
16 ?>
```

Ok cool. We have now made use of two additional WordPress functions, the_title() and the_content(). Most often, these functions are used inside the loop and what they do is to fetch the title and the content of each post as the loop iterates over each one in the database. So as the loop runs, it will come across the first post. At that time the_title() function will output the title of the post to the page and the_content() will output the body of that post to the page. On the next loop these functions will again fetch the next title and content and output them to the page. So with these in place, we should now see some information about our posts getting sent to the screen. Let's try it! We visit http://wordpresstutorial.dev and see it works!

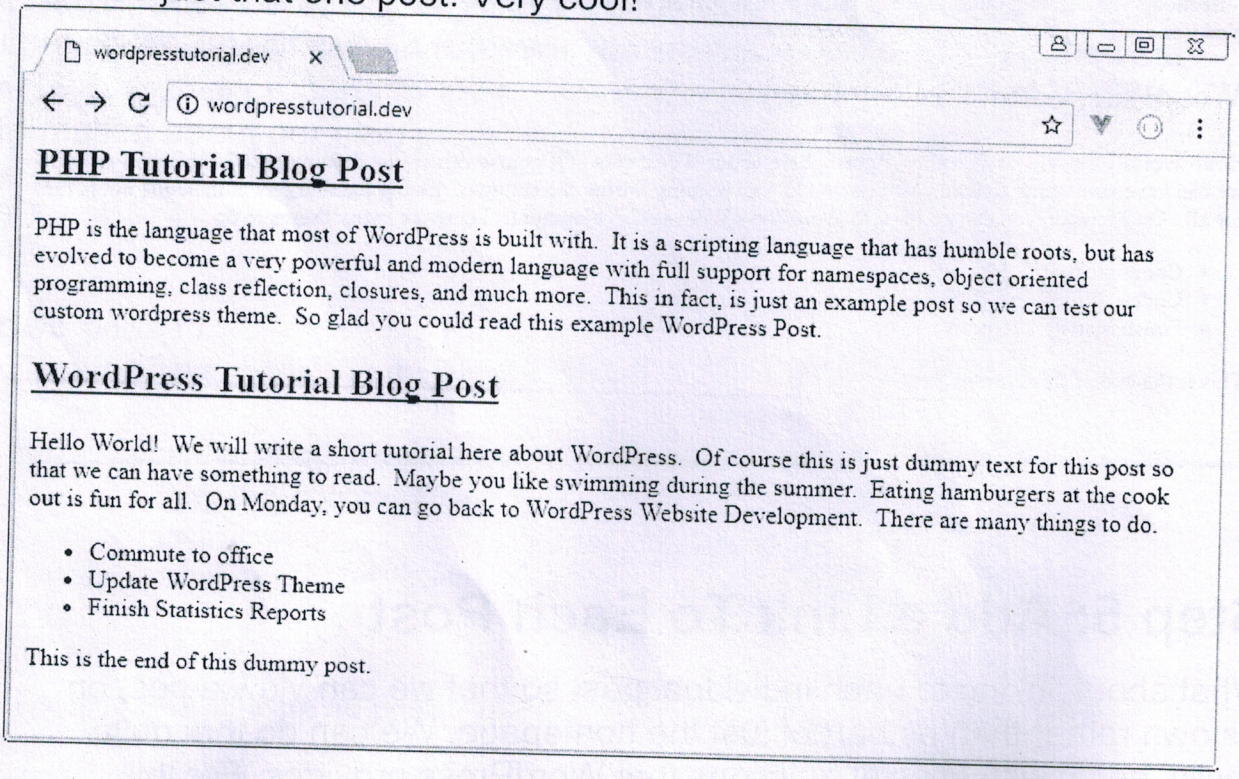


Step 5: Add a Link To Each Post

What about linking to each individual post so that we can view a post on it's own rather than as part of just the homepage. We can do that quite easily, again with special functions that WordPress provides. For this task, we can make use of the the_permalink() function. We can update our code like so:

```
1  <?php
2
3  if ( have_posts() ) :
4          while ( have_posts() ) : the_post(); ?>
5
6         <h2><a href="<?php the_permalink() ?>"><?php the_title() ?></a></h2>
7                  <?php the_content() ?>
8
9          <?php endwhile;
10
11 else :
12         echo '<p>There are no posts!</p>';
13
14 endif;
15
16 ?>
```

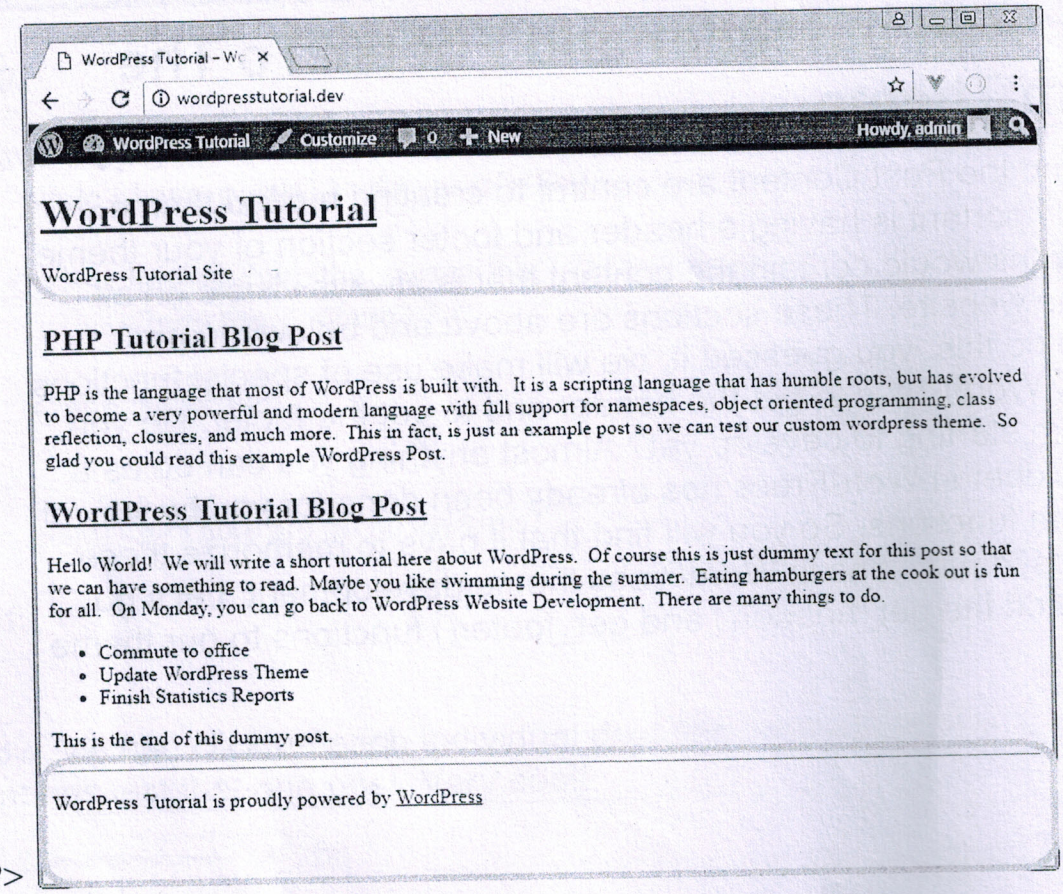
Now, we can click on each individual post title, and navigate to a page that has just that one post. Very cool!



Step 6: Add a Header and Footer To The Custom Theme

The Title and the Post Content are central to creating a good theme. Almost as important is having a header and footer section of your theme. These sections would contain the content that is always visible on all pages of the website. These sections are above and below the post content. To do this, you guessed it, we will make use of special functions provided by WordPress to get the header and to get the footer. Do you see a pattern starting to develop yet? Almost anything you can do as a theme developer in WordPress has already been done for you by way of these custom functions. So you will find that it pays to memorize these commonly used functions in WordPress theme development. Let's go ahead and add the get_header() and get_footer() functions to our theme file now.

```
<?php

get_header();

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

      <h2><a href="<?php the_permalink() ?>"><?php the_title() ?></a></h2>
            <?php the_content() ?>

        <?php endwhile;

else :
        echo '<p>There are no posts!</p>';

endif;

get_footer();
```



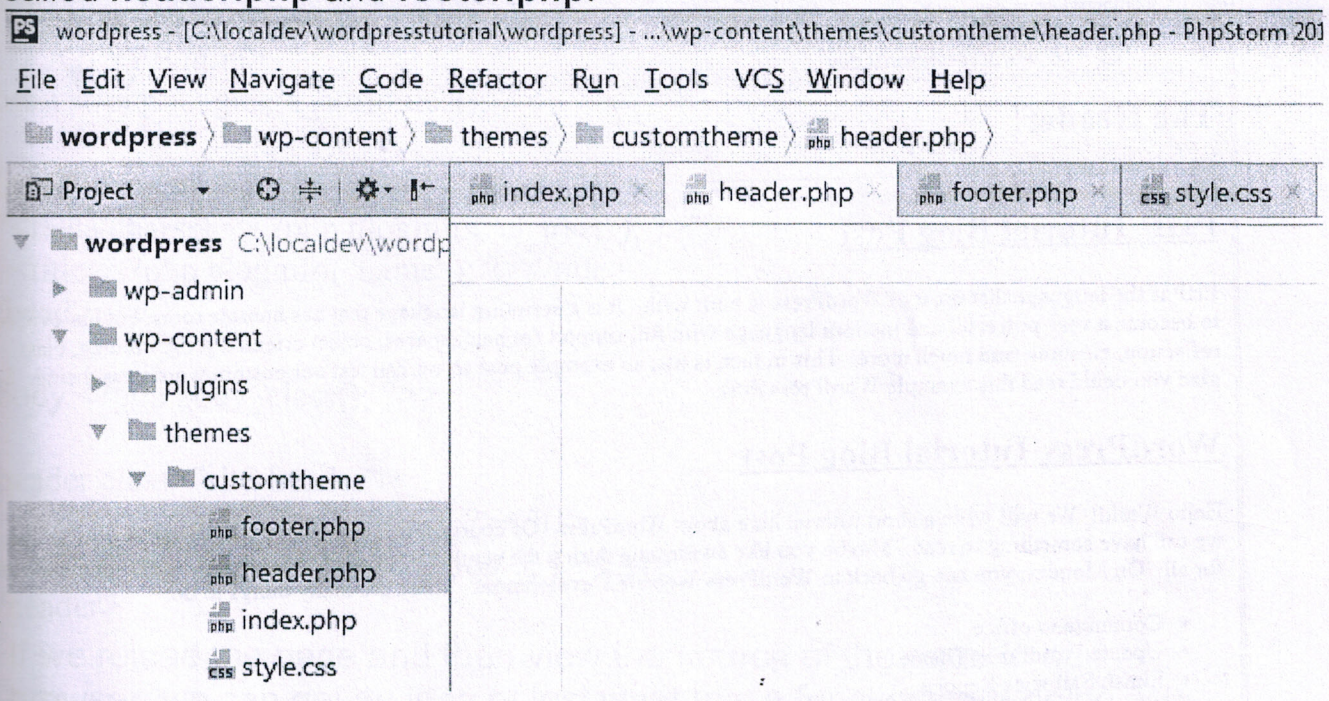
?>

Well would you look at that?! We can see that our custom theme now has a header area as well as a footer area. In the header is the name and tagline of the site while in the footer we see the familiar text, *WordPress Tutorial is proudly powered by WordPress*. These are the default header and footer options when using these functions. What about when we want to have a custom header and footer? Coming right up!

From 2 Theme Files to 4

So far in this tutorial, we have two files that live in our **customtheme**folder **(which itself is in the themes** folder**). Those files are style.css** and **index.php**. At this point, we are going to **need to add** more files to get further **along. Go ahead and create two new files in** the **customtheme**folder. **These files will be conveniently**

called **header.php** and **footer.php**.



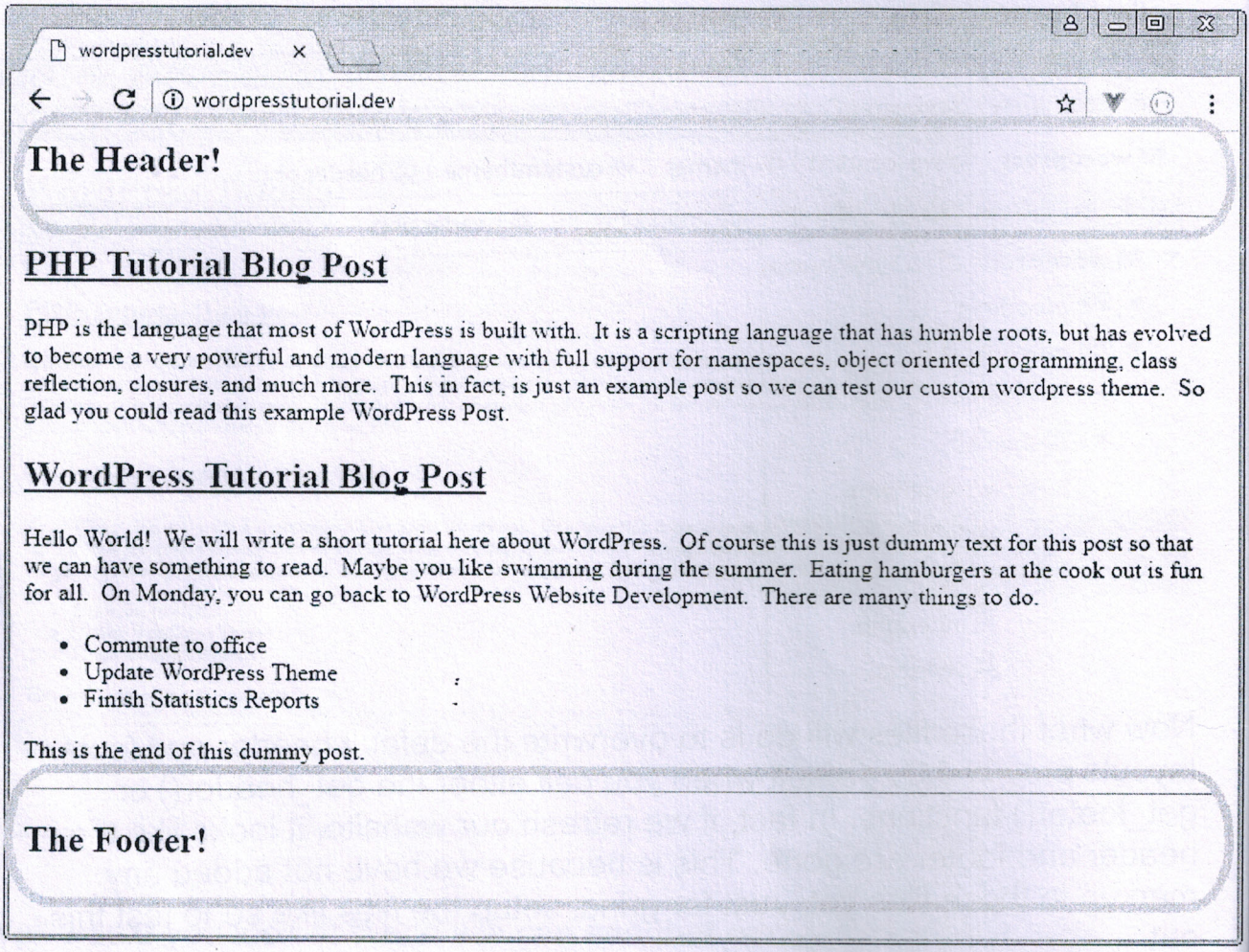
Now what these files will do is to overwrite the default header and footer layouts provided by default when you call either the get_header() or get_footer() functions. In fact, if we refresh our website, it looks like the header and footer are gone. This is because we have not added any markup to those files yet. Just for grins, setup the files like so to test this out.

header.php

```
<h2>The Header!</h2>
<hr>
```

footer.php

```
<hr>
<h2>The Footer!</h2>
```



Working with header.php

Our example above worked great, and it shows us how this file works at it's most basic level. The header.php file is actually quite important however, so let's not gloss over the details of it too quickly! This is where you include code that all pages on your site will need access to in one way or another. To start with, all HTML pages will have a doctype. You would specify that in this file. Additionally, all pages will have an opening html tag, a head section, and an opening body tag. All of this can go in the header.php file. Let's quickly add some of these things that all web pages would make use of. We will make use of a few new WordPress functions here as well. Those will be language_attributes(), bloginfo() and body_class().

header.php

```
DOCTYPE html>
tml <?php language_attributes(); ?>>

ead>
<meta charset="<?php bloginfo( 'charset' ); ?>">
<title><?php bloginfo( 'name' ); ?></title>
ead>

ody <?php body_class(); ?>>

eader class="site-header">
<h1><?php bloginfo( 'name' ); ?></h1>
<h4><?php bloginfo( 'description' ); ?></h4>
eader>
```

f we reload the page and then view the source of the page in our rowser, we can get an idea of just what these functions are doing. We ighlight the lines that have output which came from those functions elow:

```
DOCTYPE html>
tml lang="en-US">
ead>
eta charset="UTF-8">
tle>WordPress Tutorial</title>
ead>

ody class="home blog logged-in admin-bar no-customize-support">
eader class="site-header">
1>WordPress Tutorial</h1>
4>WordPress Tutorial Site</h4>
eader>
><a href="http://wordpresstutorial.dev/2017/06/12/php-tutorial-blog-
t/">PHP Tutorial Blog Post</a></h2>
PHP is the language that most of WordPress is built with.  It is a scripting
guage that has humble roots, but has evolved to become a very powerful and
dern language with full support for namespaces, object oriented programming,
s reflection, closures, and much more.  This in fact, is just an example post so
can test our custom wordpress theme.  So glad you could read this example
```

```
20 WordPress Post.</p>
21 <h2><a href="http://wordpresstutorial.dev/2017/06/12/wordpress-tutorial-blog-
22 post/">WordPress Tutorial Blog Post</a></h2>
23 <p>Hello World!  We will write a short tutorial here about WordPress.  Of course
24 this is just dummy text for this post so that we can have something to read.  Maybe
   you like swimming during the summer.  Eating hamburgers at the cook out is fun for
   all.  On Monday, you can go back to WordPress Website Development.  There are
   many things to do.</p>
   <ul>
    <li>Commute to office</li>
    <li>Update WordPress Theme</li>
    <li>Finish Statistics Reports</li>
   </ul>
   <p>This is the end of this dummy post.</p>
   </body>
   </html>
```

Again, we can see the very liberal use of WordPress functions when developing your own themes in WordPress. In fact, we have not written any custom code at all yet. We are simply learning what the various WordPress functions can offer us, and then putting them to work in our custom theme.

Including wp_head()

wp_head() is kind of a special function when you're working with WordPress Themes. It's not quite as simple as all the others we have looked at so far. The purpose of this function is to finalize the output in the <head> section of your header.php file. In fact it is meant to go just prior to the closing </head> tag. This becomes important when you start adding various plugins to your site. It prints scripts or data in the head tag on the front end. It is a good practice to always include wp_head() in your themes as many other plugins may rely on this hook to add styles, scripts, or meta elements into the <head> area of the site. We will add it as such here:

header.php

```
1 <!DOCTYPE html>
```

```
<html <?php language_attributes(); ?>>

<head>
    <meta charset="<?php bloginfo( 'charset' ); ?>">
    <title><?php bloginfo( 'name' ); ?></title>
        <?php wp_head() ?>
</head>

<body <?php body_class(); ?>>

<header class="site-header">
    <h1><?php bloginfo( 'name' ); ?></h1>
    <h4><?php bloginfo( 'description' ); ?></h4>
</header>
```

Completing footer.php

We have finished adding the basics of what we will need in the header.php file. Let's now go ahead and round out the footer.php file. The are a few things we need to do. Recall that in our header.php file we have opening html and body tags. Those need to be closed at some point. The footer.php file is a perfect place to do that. So we will add closing </html> and </body> tags in addition to making a call to the wp_footer() function.

footer.php

```
<footer class="site-footer">
    <p><?php bloginfo( 'name' ) ?></p>
</footer>

<?php wp_footer() ?>
</body>
</html>
```

Changing Site Information In The WordPress Dashboard

You might be wondering why we had to make use of all these fancy functions to build up our theme. For example, when we wanted to list out the name and tagline of our site, we made use of the bloginfo() function passing parameters of name and description. The reason for this is because generally, you never want to hard code these values into your site. This is information that might change. Additionally, if you make your theme available to the public, they will have their own name and tagline for their website. They should be able to simply visit the admin dashboard in WordPress and update their General Settings to see this data populate automatically.

Making The Site Title Link To The Homepage

Most themes will offer the ability to click on the title text of the website, and direct the user to the homepage of the site. This way, no matter where the user may be on the site, they can always click that title text and go right back to the main page of the website. Let's add that link now in header.php.

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>

<head>
   <meta charset="<?php bloginfo( 'charset' ); ?>">
   <title><?php bloginfo( 'name' ); ?></title>
      <?php wp_head() ?>
</head>

<body <?php body_class(); ?>>
<header class="site-header">
   <h1><a href="<?php echo home_url(); ?>"><?php bloginfo( 'name' );
?></a></h1>
   <h4><?php bloginfo( 'description' ); ?></h4>
</header>
```

Step 7: Add a functions.php file to your theme

At this point, we have four files in our custom theme. Those are **index.php**, **style.css**, **header.php**, and **footer.php**. Probably the next most important file we need to have is the **functions.php** file.

The functions.php file in WordPress does many things for your theme. It is the file where you place code to modify the default behavior of WordPress. You can almost think of the functions.php as a form of a plugin for WordPress with a few key points to remember:

- Does not require unique Header text
- Stored in the folder that holds your theme files
- Executes only when in the currently activated theme's directory
- Applies only to the current theme
- Can call PHP functions, WordPress functions, or custom functions

One thing we need badly in our theme is some better styling! Let's create a function in our functions.php file to include the style.css file into our theme. Here is how we can achieve that goal.

```
<?php

function custom_theme_assets() {
        wp_enqueue_style( 'style', get_stylesheet_uri() );
}

add_action( 'wp_enqueue_scripts', 'custom_theme_assets' );
```

This piece of code will include, or make active, the stylesheet of our custom theme. Now you might be wondering why we are using a custom function, when it seems like we could just as easily manually link to the stylesheet ourselves in the header.php file. Well, this comes down to doing a little more work up front for a bigger return on your effort later. As themes get more complex and more assets are added, you will be happy to have this one function that can handle all the heavy lifting for you.

Now it's time to makes things look a little more pretty. First, let's add a wrapping with a class of container. The opening will be in

header.php, while the closing will be in footer.php. We'll also wrap the post output in index.php with an <article> tag that has a class of post. This will give us classes to target in our style.css file so that we can set page width among other things. We'll also add some better styling to style.css in this step.

Step 8: Add Some Style To Your Theme

header.php

Adding an opening to the header.php file.

```
1  <!DOCTYPE html>
2  <html <?php language_attributes(); ?>>
3
4  <head>
5     <meta charset="<?php bloginfo( 'charset' ); ?>">
6     <title><?php bloginfo( 'name' ); ?></title>
7              <?php wp_head() ?>
8  </head>
9
10 <body <?php body_class(); ?>>
11 <div class="container">
12    <header class="site-header">
13       <h1><a href="<?php echo home_url(); ?>"><?php bloginfo( 'name' ); ?></a></h1>
14       <h4><?php bloginfo( 'description' ); ?></h4>
15    </header>
```

footer.php

Adding a closing to the footer.php file.

```
1 <footer class="site-footer">
2    <p><?php bloginfo( 'name' ) ?></p>
3 </footer>
4 </div> <!-- closes <div class=container"> -->
5
6 <?php wp_footer() ?>
7 </body>
8 </html>
```

index.php

Wrapping the post output with an <article> tag

```
1  <?php
2
3  get_header();
4
5  if ( have_posts() ) :
6          while ( have_posts() ) : the_post(); ?>
```

```
7
8       <article class="post">
9         <h2><a href="<?php the_permalink() ?>"><?php the_title() ?></a></h2>
10                    <?php the_content() ?>
11      </article>
12
13       <?php endwhile;
14
15 else :
16        echo '<p>There are no posts!</p>';
17
18 endif;
19
20 get_footer();
21
22 ?>
```

style.css

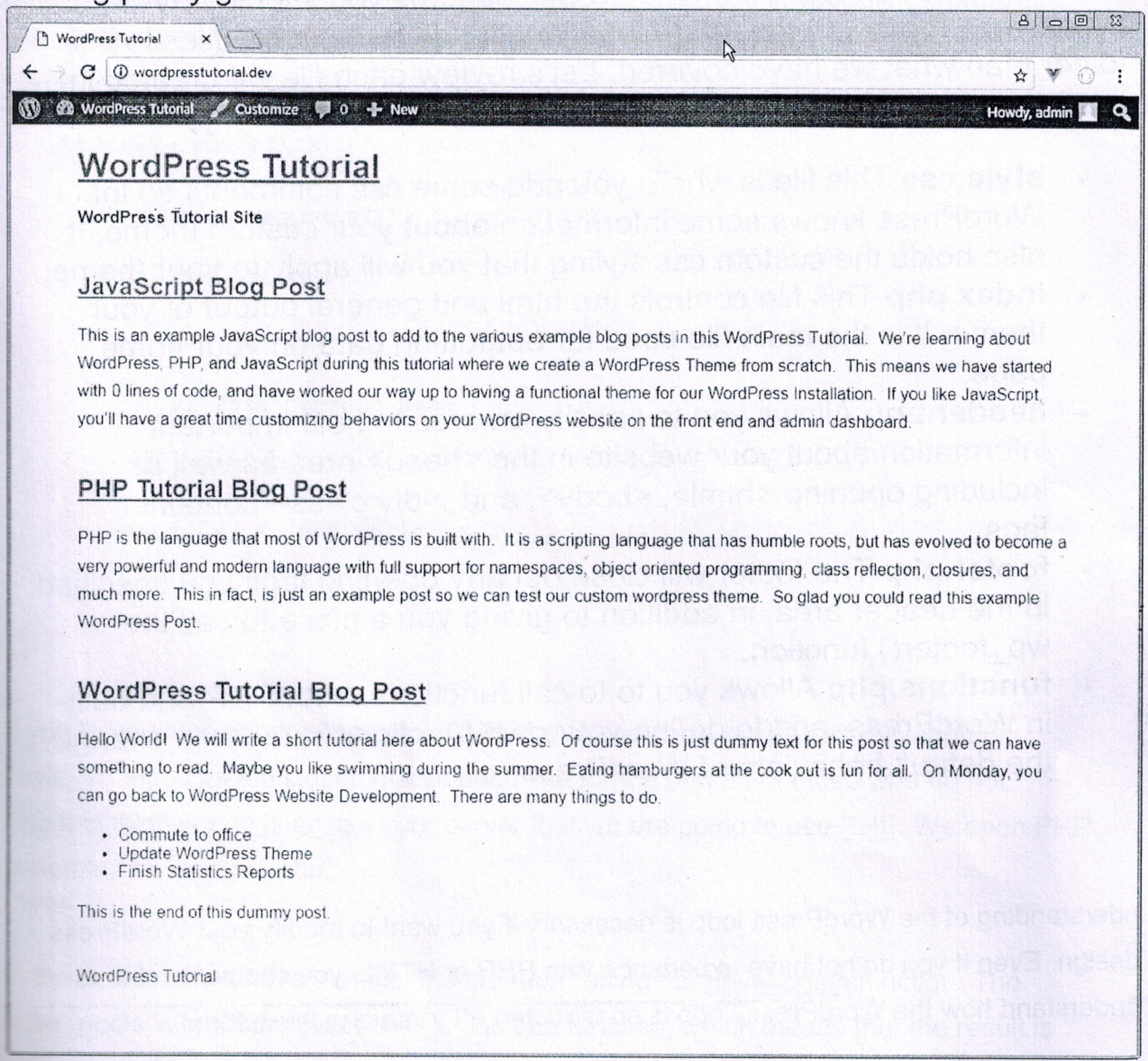
Finally, we add some various CSS style improvements to make the theme look a bit nicer.

```
1  /*
2  Theme Name: customtheme
3  Author: Vegibit
4  Author URI: http://vegibit.com
5  Version: 1.0
6   */
7
8  body {
9     font-family: Arial, sans-serif;
10    font-size: 16px;
11    color: #545454;
12 }
13
14 a:link, a:visited {
15    color: #4285f4;
16 }
17
18 p {
19    line-height: 1.7em;
20 }
```

```
21
22 div.container {
23     max-width: 960px;
24     margin: 0 auto;
25 }
26
27 article.post {
28     border-bottom: 4px dashed #ecf0f1;
29 }
30
31 article.post:last-of-type {
32     border-bottom: none;
33 }
34
35 .site-header {
36     border-bottom: 3px solid #ecf0f1;
37 }
38
39 .site-footer {
40     border-top: 3px solid #ecf0f1;
41 }
```

When we visit our test website now in the browser, we can see that the WordPress Theme that we have developed step by step in this tutorial is

looking pretty good!



WordPress Theme Development Tutorial Step By Step Summary

Let's review everything that we've learned in this basic step by step WordPress Theme tutorial for beginners. We've learned how to create our first custom theme in WordPress by making our own folder in side of the themes folder of our WordPress installation. In this folder, we added different files that correspond to different sections of your website. In our

tutorial, we have started with the bare minimums you should have in a WordPress theme. In the future, you would add many more files to this folder than what we have covered. Let's review each file we created in this tutorial, and what they did for us.

- **style.css** This file is where you add some css comments so that WordPress knows some information about your custom theme. It also holds the custom css styling that you will apply to your theme.
- **index.php** This file controls the html and general output of your theme. It is the main file used for outputting data on your home page.
- **header.php** Allows you to specify an area to hold important information about your website in the <head> area as well as including opening <html>, <body>, and , tags.
- **footer.php** The footer will close out any opening tags you specified in the header area, in addition to giving you a place to call the wp_footer() function.
- **functions.php** Allows you to to call functions, both PHP and built-in WordPress, and to define your own functions in order to change the default behaviors of WordPress

nderstanding of the WordPress loop is necessary if you want to modify your WordPress design. Even if you do not have experience with PHP or HTML, you should be able to understand how the WordPress Loop is constructed after reading this tutorial.

Understanding the WordPress Loop

The best way to learn about the WordPress Loop is to look at a basic example of how it is used in a WordPress theme. Therefore, let us look at some simple code initially and then I can break down each line to give you a better understanding of what each line does.

Below is an example of a simple WordPress Loop. The code in your own WordPress theme for the loop may be much longer, however it follows the same structure as this.

```
<?php

    if ( have_posts() ) :

        while ( have_posts() ) :

            the_post();

            //

            // Post Content here

            //

        endwhile; // end while

    endif; // end if

?>
```

If you have some experience using PHP, the above code will be self-explanatory; however, let us take a closer look at each line for the benefit of those who do not.
The first thing we do is advise your server that we are going to use PHP. We open PHP statement by using <?php.

```
<?php
```

In the next line, we have a basic "*if statement*" using the have_posts function. The have_posts WordPress function is a boolean function; which means that the result is either true of false.
Therefore, the following line of code effectively says "If there are some posts, display this line of code, if not, do nothing".

```
if ( have_posts() ) :
```

In the next line, we use a while loop. A while loop will execute a piece of code as long as something is true. In this case, we are saying that while there are posts to be displayed, execute the following line of code.

Therefore, if you had configured your WordPress reading settings to display five posts on the home page, the while function would execute the statements contained within the while loop five times and then stop.

```
while ( have_posts() ) :
```

We then call the data from the next post by using the WordPress function the_post. This sets up the post and allows us to retrieve any part of the post including the content, the publication date, the author, the category it was published in, and much more later on.

```
the_post();
```

Once we have called up our post, we can display anything we want from it. There are over one hundred template tags available that **can only be used** within the WordPress Loop.
Examples include the_title for displaying the post title, the_content for displaying the post itself, and the_category for displaying the post category.

```
//

// Post Content here

//
```

After we have confirmed the information that we want displayed with every post, we close the while loop.

```
endwhile; // end while
```

We then close the if statement.

```
endif; // end if
```

Finally, we end by closing PHP.

```
?>
```

As you can see, when you break down the WordPress Loop, it is very easy to understand.

General tags

1)get header()

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

Parameters

$name

(*string*) (*optional*) Calls for `header-name.php`.

Default: *None*

Description

Includes the `header.php` template file from your current theme's directory. If a name is specified then a specialised header `header-{name}.php` will be included.

If the theme contains no `header.php` file then the header from the default theme `wp-includes/theme-compat/header.php` will be included.

2) get footer()

Usage

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

Parameters

$name

(*string*) (*optional*) Calls for `footer-name.php`.

Default: *None*

Description

Includes the `footer.php` template file from your current theme's directory. if a name is specified then a specialised footer `footer-{name}.php` will be included.

If the theme contains no `footer.php` file then the footer from the default theme `wp-includes/theme-compat/footer.php` will be included.

3) get_sidebar(*string* $name = null)

Description

Includes the sidebar template for a theme or if a name is specified then a specialised sidebar will be included.

For the parameter, if the file is called "sidebar-special.php" then specify "special".

Parameters

$name

(string) (Optional) The name of the specialised sidebar.
Default value: null

```
<?php get_sidebar('nice-bar'); ?>
```

4) get_search_form(*bool* $echo = true)

Display search form.

Description

Will first attempt to locate the searchform.php file in either the child or the parent, then load it. If it doesn't exist, then the default search form will be displayed. The default search form is HTML, which will be displayed. There is a filter applied to the search form HTML in order to edit or replace it. The filter is 'get_search_form'.
This function is primarily used by themes which want to hardcode the search form into the sidebar and also by the search widget in WordPress.

There is also an action that is called whenever the function is run called, 'pre_get_search_form'. This can be useful for outputting JavaScript that the search relies on or various formatting that applies to the beginning of the search. To give a few examples of what it can be used for.

Parameters

$echo

*(*bool*) (Optional)* Default to echo and not return the form.
Default value: true

- 'url' – The Site address (URL) (set in Settings > General)
- 'admin_email' – Admin email (set in Settings > General)
- 'charset' – The "Encoding for pages and feeds" (set in Settings > Reading)
- 'version' – The current WordPress version
- 'html_type' – The content-type (default: "text/html"). Themes and plugins can override the default value using the 'pre_option_html_type' filter
- 'text_direction' – The text direction determined by the site's language. is_rtl()should be used instead
- 'language' – Language code for the current site
- 'stylesheet_url' – URL to the stylesheet for the active theme. An active child theme will take precedence over this value
- 'stylesheet_directory' – Directory path for the active theme. An active child theme will take precedence over this value
- 'template_url' / 'template_directory' – URL of the active theme's directory. An active child theme will NOT take precedence over this value
- 'pingback_url' – The pingback XML-RPC file URL (xmlrpc.php)
- 'atom_url' – The Atom feed URL (/feed/atom)
- 'rdf_url' – The RDF/RSS 1.0 feed URL (/feed/rfd)
- 'rss_url' – The RSS 0.92 feed URL (/feed/rss)
- 'rss2_url' – The RSS 2.0 feed URL (/feed)
- 'comments_atom_url' – The comments Atom feed URL (/comments/feed)
- 'comments_rss2_url' – The comments RSS 2.0 feed URL (/comments/feed)

6) wp_title();

wp_title(*string* $sep = '»', *bool* $display = true, *s*
ring $seplocation = '')

Description

By default, the page title will display the separator before the page title, so that the blog title will be before the page title. This is not good for title display, since the blog title shows up on most tabs and not what is important, which is the page that the user is looking at.

There are also SEO benefits to having the blog title after or to the 'right' of the page title. However, it is mostly common sense to have the blog title to the right with most browsers supporting tabs. You can achieve this by using the seplocation parameter and setting the value to 'right'. This change was introduced around 2.5.0, in case backward compatibility of themes is important.

Parameters

$sep

(string) (Optional) default is '»'. How to separate the various items within the page title.
Default value: '»'

$display

(bool) (Optional) Whether to display or retrieve title.
Default value: true

$seplocation

(string) (Optional) Direction to display title, 'right'.
Default value: ''

Return

(string|null) String on retrieve, null when displaying.

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

7) Singal post title()

Description

Displays or returns the title of the post when on a single post page (permalink page). This tag can be useful for displaying post titles outside The Loop.

Usage

```
<?php single_post_title( $prefix, $display ); ?>
```

Default Usage

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

Parameters

$prefix

(string) (*optional*) Text to place before the title.

Default: *None*

$display

(*boolean*) (*optional*) Should the title be displayed (`TRUE`) or returned for use in PHP (`FALSE`).

Default: `TRUE`

Example

```
<h2><?php single_post_title( 'Current post: ' ); ?></h2>
```

8) comments_template();

comments_template(*string* $file = '/comments.php', *bool* $separate_comments = false)

Description

Will not display the comments template if not on single post or page, or if the post does not have comments.

Uses the WordPress database object to query for the comments. The comments are passed through the 'comments_array' filter hook with the list of comments and the post ID respectively.
The `$file` path is passed through a filter hook called 'comments_template', which includes the TEMPLATEPATH and $file combined. Tries the $filtered path first and if it fails it will require the default comment template from the default theme. If either does not exist, then the WordPress process will be halted. It is advised for that reason, that the default theme is not deleted.

Parameters

$file

(string) (Optional) The file to load.
Default value: '/comments.php'

$separate_comments

(bool) (Optional) Whether to separate the comments by comment type.
Default value: false

Usage

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

9) add_theme_support();

add_theme_support(*string* $feature)

Description

Must be called in the theme's functions.php file to work. If attached to a hook, it must be 'after_setup_theme'. The 'init' hook may be too late for some features.

Parameter

$feature

(string) (Required) The feature being added. Likely core values include 'post-formats', 'post-thumbnails', 'html5', 'custom-logo', 'custom-header-uploads', 'custom-header', 'custom-background', 'title-tag', 'starter-content', etc.

$args,...

(mixed) (Optional) extra arguments to pass along with certain features.

Return

(void|bool) False on failure, void otherwise.

```
add_theme_support( 'custom-background' );
```

10)wp_footer()

Fire the wp_footer action.

Description

See 'wp_footer'.

```
<body>
    <!-- All the document's HTML goes first. -->
    <!-- Then last, before closing the body tag, add: -->
```

```
    <?php wp_footer(); ?>
</body>
```

```
    * Always have wp_footer() just before the closing </body>

    * tag of your theme, or you will break many plugins, which

    * generally use this hook to reference JavaScript files.
```

11)get_template_directory_uri()

Retrieve theme directory URI.

Return

(string) Template directory URI.(Uniform Resource Identifier)

Using get_template_directory_uri() to link a static image with its correct path in html :

Example:-

```
<img src="<?php echo get_template_directory_uri(); ?>/images/logo.png"
width="" height="" alt="" />
```

12) body_class();

body_class(*string|array* $class = '')

Display the classes for the body element.

This function gives the body element different classes and can be added, typically, in the header.php's HTML body tag.

Parameters

$class

(string|array) (Optional) One or more classes to add to the class list.

Default value: ''

The following example shows how to implement the body_class template tag into a theme.

```
1    <body <?php body_class(); ?>>
```

13)wp_head()

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

Fire the wp_head action.The **wp_head action hook** is triggered within the <head></head> section of the user's template by the wp_head() function. Although this is theme-dependent, it is one of the most essential theme hooks, so it is widely supported.

```
*Always have wp_head() just before the closing </head>

* tag of your theme, or you will break many plugins, which

* generally use this hook to add elements to <head> such

* as styles, scripts, and meta tags.
```

Category tags

1)category_description();

Description

Returns the description of a category defined in the category settings screen for the current category (Posts > Categories).

If used in the archive.php template, place this function within the is_category() conditional statement. Otherwise, this function will stop the processing of the page for monthly and other archive pages.

Usage

```
<?php echo category_description( $category_id ); ?>
```

Parameters

$category_id

(*integer*) (*optional*) The ID of the category to return a description.

Default: Description of current query category.

Example

Default Usage

Displays the description of a category, given its id, by echoing the return value of the tag. If no category given and used on a category page, it returns the description of the current category.

```
<div><?php echo category_description(3); ?></div>
```

Result:

WordPress is a favorite blogging tool of mine and I share tips and tricks for using WordPress here.

Note: if there is no category description, the function returns a br tag.

```
<div><strong><?php single_cat_title('Currently browsing'); ?>
</strong>: <?php echo category_description(); ?></div>
```

2)single_cat_title()

Description

Displays or returns the tag title for the current archive page.

Usage

```
<?php single_tag_title( $prefix, $display ); ?>
```

Default Usage

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

Parameters

$prefix

(*string*) (*optional*) Text to output before the title.

Default: *None*

Note: The $prefix argument is currently ignored if the $display argument is false.
See http://core.trac.wordpress.org/ticket/16632

$display

(*boolean*) (*optional*) Display the title (`TRUE`), or return the title to be used in PHP (`FALSE`).

Default: `TRUE`

Examples

This example displays the text "**Currently browsing** " followed by the tag title.

```
<p><?php single_tag_title('Currently browsing '); ?>.</p>
```

Currently browsing WordPress.

This example assigns the current tag title to the variable $current_tag for use in PHP.

```
<?php $current_tag = single_tag_title("", false); ?>
```

3)the _category()

Description

Displays a link to the category or categories a post belongs to. This tag must be used within The Loop.

Usage

```
<?php the_category( $separator, $parents, $post_id ); ?>
```

Parameters

`$separator`

(*string*) (*optional*) Text or character to display between each category link. By default, the links are placed in an HTML unordered list. An empty string will result in the default behavior.

Default: empty string

`$parents`

(*string*) (*optional*) How to display links that reside in child (sub) categories. Options are:

- `'multiple'` - Display separate links to parent and child categories, exhibiting "parent/child" relationship.
- `'single'` - Display link to child category only, with link text exhibiting "parent/child" relationship.

Default: empty string
Note: Default is a link to the child category, with no relationship exhibited.

$post_id

(*int*) (*optional*) Post ID to retrieve categories. The default value `false` results in the category list c the current post.

Default: false

Examples

Separated by Space

List categories with a space as the separator. `<?php the_category( ' ' ); ?>`

Separated by Comma

Displays links to categories, each category separated by a comma (if more than one).
`<?php the_category( ', ' ); ?>`

Separated by Arrow

Displays links to categories with an arrow (>) separating the categories. **Note:** Take care when using this since some viewers may interpret a category following a > as a subcategory of the one preceding it.
`<?php the_category( '> ' ); ?>`

Separated by a Bullet

Displays links to categories with a bullet (•) separating the categories.
`<?php the_category( '&bull;' ); ?>`

Link tages:=

1)the_permalink();

Description

Displays the URL for the permalink to the post currently being processed in The Loop. This tag must be within The Loop, and is generally used to display the permalink for each post, when the posts are being displayed. Since this template tag is limited to displaying the permalink for the post that is being processed, you cannot use it to display the permalink to an arbitrary post on your weblog. Refer to get_permalink() if you want to get the permalink for a post, given its unique post id.

Usage

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

Parameters

Before 4.4.0, this tag has no parameters. Since 4.4.0: Added the `$post` parameter.

Examples

Display Post URL as Text

Displays the URL to the post, without creating a link:

```
This address for this post is: <?php the_permalink(); ?>
```

As Link With Text

You can use whatever text you like as the link text, in this case, "permalink".

```
<a href="<?php the_permalink(); ?>">permalink</a>
```

Used as Link With Title Tag

Creates a link for the permalink, with the post's title as the link text. This is a common way to put the post's title.

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

2)get_permalink()

get_permalink(*int|WP_Post* $post, *bool* $leavename = false)

Retrieves the full permalink for the current post or post ID.

Parameters

$post

(int|WP_Post) (Optional) Post ID or post object. Default is the global `$post`.

$leavename

(bool) (Optional) Whether to keep post name or page name.
Default value: false

Return

(string|false) The permalink URL or false if post does not exist.

3)home_url()

Description

The **home_url** template tag retrieves the home URL for the current site, optionally with the `$path`argument appended. The function determines the appropriate protocol, "https" if `is_ssl()` and "http" otherwise. If the `$scheme` argument is "http" or "https" the `is_ssl()` check is overridden.

In case of WordPress Network Setup, use network_home_url() instead.

Usage

```
<?php home_url( $path, $scheme ); ?>
```

Default Usage

```
<?php echo esc_url( home_url( '/' ) ); ?>
```

Parameters

$path

(string) (*optional*) Path relative to the home URL.

Default: *None*

$scheme

(string) (*optional*) Scheme to use for the home URL. Currently, only "http", "https" and "relative" are supported.

Default: null

Return

(string)

Home URL with the optional $path argument appended.

Example

```
$url = home_url();

echo esc_url( $url );
```

Output: http://www.example.com

(Note the lack of a trailing slash)

```
$url = home_url( '/' );

echo esc_url( $url );
```

Output: http://www.example.com/

4)get_home_url()

Description

Returns the 'home' option with the appropriate protocol. The protocol will be 'https' if is_ssl() evaluates to true; otherwise, it will be the same as the 'home' option. If `$scheme` is 'http' or 'https', is_ssl() is overridden.

Parameters

$blog_id

(int) (Optional) Site ID. Default null (current site).
Default value: null

$path

(string) (Optional) Path relative to the home URL.
Default value: ''
$scheme

(string|null) (Optional) Scheme to give the home URL context. Accepts 'http', 'https', 'relative', 'rest', or null.
Default value: null

Return

(string) Home URL link with optional path appended.

Basic Usage

```
1    <?php echo get_home_url(); ?>
```

Will output: `https://www.example.com` With the domain and the schema matching your settings.

5)site_url()

Description

The **site_url** template tag retrieves the site url for the current site (where the WordPress core files reside) with the appropriate protocol, 'https' if is_ssl() and 'http' otherwise. If scheme is 'http' or 'https', is_ssl() is overridden. Use this to get the "WordPress address" as defined in general settings.
Use home_url() to get the "site address" as defined in general settings.

In case of WordPress Network setup, use network_site_url() instead.

Usage

```
<?php site_url( $path, $scheme ); ?>
```

Default Usage

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

Parameters

$path

(*string*) (*optional*) Path to be appended to the site url.

Default: *None*

$scheme

(*string*) (*optional*) Context for the protocol for the url returned. Setting $scheme will override the default context. Allowed values are 'http', 'https', 'login', 'login_post', 'admin', or 'relative'.

Default: null

Return

(string)

Site url link with optional path appended.

Examples

```
$url = site_url();

echo $url;
```

Output: http://www.example.com or http://www.example.com/wordpress

(Note the lack of a trailing slash)

```
$url = site_url( '/secrets/', 'https' );

echo $url;
```

Output: https://www.example.com/secrets/ or https://www.example.com/wordpress/secrets/

6)get_site_url();

`get_site_url( int $blog_id = null, string $path = '', string $scheme = null )`

Retrieves the URL for a given site where WordPress application files (e.g. wp-blog-header.php or the wp-admin/ folder) are accessible.

Description

Returns the 'site_url' option with the appropriate protocol, 'https' if is_ssl() and 'http' otherwise. If `$scheme` is 'http' or 'https', `is_ssl()` is overridden.

Parameters

$blog_id

(int) (Optional) Site ID. Default null (current site).
Default value: null

$path

(string) (Optional) Path relative to the site URL.
Default value: ''

$scheme

(string) (Optional) Scheme to give the site URL context. Accepts 'http', 'https', 'login', 'login_post', 'admin', or 'relative'.
Default value: null

Return

(string) Site URL link with optional path appended.

Example:-

```
1    <?php echo get_site_url(); ?>
```

Results in the full site URL being displayed:

```
http://www.example.com
```

Author tags

Usage

1)The_author():=

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

Description

The author of a post can be **displayed** by using this Template Tag. This tag must be used within The Loop.

Displays the value in the user's **Display name publicly as** field.

```
<p>This post was written by <?phpthe_author(); ?></p>
```

Parameters

$post-id (integer).

2)get_the_author() :=

Description

Retrieve the post author's *display name*. This tag must be used within The Loop.

To get the post author's ID, use get_the_author_meta('ID').

To display a page for authors which have no posts, see this discussion.

Since WordPress 2.1 parameters are deprecated (not the function).

Usage

```
<?php $author = get_the_author(); ?>
```

Parameters

`$deprecated`

(*string*) (*optional*) Deprecated.

Default: ''

Returns

`(string)`

The author's display name.

Examples

Grab the Author's 'Public' Name

Grabs the value in the user's **Display name publicly as** field.

```
<?php $author = get_the_author(); ?>
```

3)the_author_link():=

Description

This tag displays a link to the Website for the author of a post. The **Website** field is set in the user's profile (Administration>Profile>Your Profile). The text for the link is the author's Profile **Display name publicly as** field. This tag must be used within The Loop.

Usage

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

Parameters

This function does not accept any parameters.

Example

Displays the author's Website URL as a link and the text for the link is the author's Profile **Display name publicly as** field. In this example, the author's Display Name is James Smith.

```
<p>Written by:
<?phpthe_author_link(); ?></p>
```

Which displays as:

Written by: James Smith

4)Get_the_author_link():=

Description

This tag returns a link to the Website for the author of a post. The **Website** field is set in the user's profile (Administration>Users>Your Profile). The text for the link is the author's Profile **Display name publicly as** field. This tag must be used within The Loop.

get_the_author_link() returns the link for use in PHP. To *display* the link instead, use the_author_link().

Usage

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

Parameters

This tag does not accept any parameters.

Example

The example echos (displays) the author's Website URL as a link and the text for the link is the author's Profile **Display name publicly as** field. In this example, the author's Display Name is James Smith.

```
<p>Written by:
<?php echo get_the_author_link(); ?></p>
```

Which displays as:

Written by: James Smith

5)the_author_meta():=

Description

The the_author_meta Template Tag displays a desired meta data field for a user. Only one field is returned at a time, you need to specify which you want.

If this tag is used within The Loop, the user ID value need not be specified, and the displayed data is that of the current post author. A user ID can be specified if this tag is used outside The Loop.

If the meta field does not exist, nothing is printed.

NOTE: Use `get_the_author_meta()` if you need to return (and do something with) the field, rather than just display it.

Usage

```
<?php the_author_meta( $field, $userID ); ?>
```

Parameters

$field

(*string*) Field name for the data item to be displayed. Valid values:

- user_login
- user_pass
- user_nicename
- user_email
- user_url
- user_registered
- user_activation_key
- user_status
- display_name
- nickname
- first_name
- last_name
- description
- jabber
- aim
- yim
- user_level
- user_firstname
- user_lastname
- user_description
- rich_editing
- comment_shortcuts
- admin_color
- plugins_per_page
- plugins_last_view
- ID
-

$userID

(*integer*) (*optional*) If the user ID fields is used, then this function display the specific field for this user ID.

Default: false

Examples

Display the Author's AIM screenname

Displays the value in the author's **AIM** (AOL Instant Messenger screenname) field.

```
<p>This author's AIM address is <?phpthe_author_meta('aim'); ?></p>
```

6)the_author_posts():=

Description

Displays the total number of posts an author has published. Drafts and private posts aren't counted. This tag must be used within The Loop.

Usage

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

Example

Displays the author's name and number of posts.

```
<p><?phpthe_author(); ?> has blogged <?phpthe_author_posts(); ?>
posts</p>
```

Harriett Smith has blogged 425 posts.

Post tages

1) the_content();

the_content(*string* $more_link_text = null, *bool* $strip_teaser = false)

Display the post content.

Parameters :-

$more_link_text

(string) (Optional) Content for when there is more text.
Default value: null

$strip_teaser

(bool) (Optional) Strip teaser content before the more text. Default is false.
Default value: false

Overriding Archive/Single Page Behavior

If the_content() isn't working as you desire (displaying the entire story when you only want the content above the <!--more--> Quicktag, for example) you can override the behavior with global $more.

```
1   // Declare global $more (before the loop).
2   global $more;
3
4   // Set (inside the loop) to display content above the more tag.
5   $more = 0;
6
7   the_content( 'More ...' );
8   ?>
```

If you need to display all of the content:

```
1   // Declare global $more (before the loop).
    global $more;
2
3   // Set (inside the loop) to display all content, including text below more.
4   $more = 1;
5
6   the_content();
```

7

```
<?php the_content( 'Continue reading ' .
get_the_title() ); ?>
```

2)the_excerpt();

Display the post excerpt.

Displays the excerpt of the current post after applying several filters to it including auto-p formatting which turns double line-breaks into HTML paragraphs. It uses get_the_excerpt() to first generate a **trimmed-down version** of the full post content should there not be an explicit excerpt for the post.
The trimmed-down version contains a 'more' tag at the end which by default is the [...] or "hellip" symbol. A user-supplied excerpt is NOT by default given such a symbol. To add it, you must either modify the raw `$post->post_excerpt` manually in your template before calling the_excerpt(), add a filter for `'get_the_excerpt'` with a priority lower than 10, or add a filter for `'wp_trim_excerpt'` (comparing the first and second parameter, because a user-supplied excerpt does not get altered in any way by this function).
See get_the_excerpt() for more details.
An auto-generated excerpt will also have all shortcodes and tags removed. It is trimmed down to a word-boundary and the default length is 55 words. For languages in which words are (or can be) described with single characters (ie. East-Asian languages) the word-boundary is actually the character.

Displays the post excerpt. Used in templates that provide a form of index, ie. for home, category, tag, archive pages. Not used for single post views. Not meaningful for static pages.

Default Usage

Used as a replacement for the_content() to force excerpts to show within The Loop.

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

3)the_ID()

Description

Displays the numeric ID of the current post. This tag must be within The Loop.

Note: This function displays the ID of the post, to return the ID use get_the_ID().

Usage

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

Parameters

This tag has no parameters.

Examples

Default Usage

```
<p>Post Number: <?php the_ID(); ?></p>
```

Post Anchor Identifier

Provides a unique anchor identifier to each post:

```
<h3 id="post-<?php the_ID(); ?>"><?php the_title(); ?></h3>
```

4)the_tags()

Description

This template tag displays a link to the tag or tags a post belongs to. If no tags are associated with the current entry, nothing is displayed. This tag should be used within The Loop.

Usage

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

Parameters

$before

(*string*) Text to display before the actual tags are displayed. Defaults to **Tags:**

$sep

(*string*) Text or character to display between each tag link. The default is a comma (,) between each tag.

$after

(*string*) Text to display after the last tag. The default is to display nothing.

Return Values

None.

Examples

Default Usage

The default usage lists tags with each tag (if more than one) separated by a comma (,) and preceded with the default text **Tags:** .

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

Separated by Commas

Displays a list of the tags with a line break after it.

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

Separated by Arrow

Displays links to tags with an arrow (>) separating the tags and preceded with the text **Social tagging:**

```
<?php the_tags( 'Social tagging: ',' > ' ); ?>
```

Separated by a Bullet

Displays links to tags with a bullet (•) separating the tags and preceded with the text **Tagged with:** and followed by a line break.

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

A List Example

Displays a list of the tags as an unordered list:

```
<?php the_tags( '<ul><li>', '</li><li>', '</li></ul>' ); ?>
```

5)the_title()

Description

Displays or returns the unescaped title of the current post. This tag may only be used within The Loop, to get the title of a post outside of the loop use get_the_title. If the post is protected or private, this will be noted by the words "Protected: " or "Private: " prepended to the title.

Usage

```
<?php the_title( $before, $after, $echo ); ?>
```

Parameters

$before

(*string*) (*optional*) Text to place before the title.

Default: *None*

$after

(*string*) (*optional*) Text to place after the title.

Default: *None*

$echo

(*Boolean*) (*optional*) Display the title (`TRUE`) or return it for use in PHP (`FALSE`).

Default: `TRUE`

Example

```
<?php the_title( '<h3>', '</h3>' ); ?>
```

This would print the title to the screen as an h3.

6) get_the_title();

get_the_title(*int|WP_Post* $post)

Retrieve post title.

Description

If the post is protected and the visitor is not an admin, then "Protected" will be displayed before the post title. If the post is private, then "Private" will be located before the post title.

Parameters

$post

(int|WP_Post) (Optional) Post ID or WP_Post object. Default is global $post.

Return

(string)

get_the_title intentionally allows for HTML

So get_the_title should not be escaped.

Use `the_title_attribute()` instead of `get_the_title()` if you're outputting the post title for html attributes.

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

7)the_date();

Description

Displays or returns the date of a post, or a set of posts if published on the same day.

Usage

```
<?php the_date( $format, $before, $after, $echo ); ?>
```

Parameters

$format

(*string*) (*optional*) The format for the date. Defaults to the date format configured in your WordPress options. See Formatting Date and Time.

Default: F j, Y

$before

(*string*) (*optional*) Text to place before the date.

Default: *None*

$after

(*string*) (*optional*) Text to place after the date

Default: *None*

$echo

(*boolean*) (*optional*) Display the date (`TRUE`), or return the date to be used in PHP (`FALSE`).

Default: `TRUE`

Return

(*string|null*) Null if displaying, string if retrieving.

Examples

Default Usage

Displays the date using defaults.

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

Date as Year, Month, Date in Heading

Displays the date using the `'2007-07-23'` format (ex: 2004-11-30), inside an <h2> tag.

8)get_the_date()

Description

The **get_the_date** template tag retrieves the date the current `$post` was written. Unlike the_date() this tag will **always return** the date. Modify output with 'get_the_date' filter.

Usage

```
<?php $pfx_date = get_the_date( $format, $post_id ); ?>
```

Parameters

`$format`

(*string*) (*optional*) PHP date format.

Default: the `date_format` option ('Date Format' on Settings > General panel)

`$post`

(*integer*) (*optional*) The ID of the post you'd like to fetch. By default the current post is fetched.

Default: null

Return

(*string*) The formatted date string

Filter

- apply_filters('get_the_date', $the_date, $format)

Examples

Default Usage

```
<span class="entry-date"><?php echo get_the_date(); ?></span>
```

8)the_time()

Description

Displays the time of the current post. To return the time of a post, use get_the_time(). This tag must be used within The Loop.

Usage

```
<?php the_time( $d ); ?>
```

Parameters

$d

(*string*) (*optional*) The format the time is to display in. Defaults to the time format configured in your WordPress options. See Formatting Date and Time.

Default: *None*

Examples

Default Usage

Displays the time using your WordPress defaults.

```
<p>Time posted: <?php the_time(); ?></p>
```

Time as AM/PM VS. 24H format

Displays the time using the format parameter string `'g:i a'` (ex: 10:36 pm).

```
<p>Time posted: <?php the_time('g:i a'); ?></p>
```

Displays the time using the 24 hours format parameter string `'G:i'` (ex: 17:52).

```
<p>Time posted: <?php the_time('G:i'); ?></p>
```

Date as Month Day, Year

Displays the time in the date format `'F j, Y'` (ex: December 2, 2004), which could be used to replace the tag the_date().

```
<div><?php the_time('F j, Y'); ?></div>
```

Date and Time

Displays the date and time.

```
<p>Posted: <?php the_date('F j, Y'); ?> at <?php the_time('g:i a'); ?></p>
```

Posted: July 17, 2007 at 7:19 am

9)next_post_link();

Description

Used on single post permalink pages, this template tag displays a link to the next post which exists in chronological order from the current post.

In standard usage (within the default, unaltered loop) `next_post_link` will generate a link to a post that is newer (more recent) than the current post. This is in contrary to the similarly-named `previous_posts_link`, which will typically link to a page of posts that is *older* than the current batch.
This tag must be used in The Loop.

Usage

```
<?php next_post_link( $format, $link, $in_same_term = false, $excluded_terms
= '', $taxonomy = 'category' ); ?>
```

Parameters

format

(string) (*Optional*) Format string for the link. This is where to control what comes before and after the link. `'%link'` in string will be replaced with whatever is declared as `'link'` (see next parameter). `'Go to %link'` will generate "Go to <a href=..." Put HTML tags here to style the final results.

Default: `'%link &raquo;'`

link

(string) (*Optional*) Link text to display.

Default: `'%title'` (next post's title)

in_same_term

Ex

D

al) Indicates whether next post must be within the same taxonomy te
post. If set to `'true'`, only posts from the current taxonomy term will be
he post is in both the parent and subcategory, or more than one term, the
nk will lead to the next post in any of those terms.

`se`
ult: false

_terms

(*string/array*) (*optional*) Array or a comma-separated list of numeric terms IDs from whi
the next post should not be listed. For example `array(1, 5)` or `'1,5'`. This argument
used to accept a list of IDs separated by `'and'`, this was deprecated in WordPress 3.3

Default: *None*

taxonomy

(*string*) (*Optional*) Taxonomy, if `$in_same_term` is true. Added in WordPress 3.8.

Default: `'category'`

Examples

Default Usage

Displays link with the post title of the next post (chronological post date order), followed
by a right angular quote (»).

Next Post Title »

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

9)previous_post_link();

Description

Used on single post permalink pages, this template tag displays a link to the previous post which exists in chronological order from the current post.

This tag must be used in The Loop.

Arguments

```
<?php previous_post_link( $format, $link, $in_same_term = false,
$excluded_terms = '', $taxonomy = 'category' ); ?>
```

Parameters

format

(string) (*Optional*) Format string for the link. This is where to control what comes before and after the link. `'%link'` in string will be replaced with whatever is declared as `'link'` (see next parameter). `'Go to %link'` will generate "Go to <a href=..." Put HTML tags here to style the final results.

Default: `'&laquo; %link'`

link

(string) (*Optional*) Link text to display.

Default: `'%title'` (previous post's title)

in_same_term

(boolean) (*optional*) Indicates whether previous post must be within the same taxonomy term as the current post. If set to `'true'`, only posts from the current taxonomy term will be displayed. If the post is in both the parent and subcategory, or more than one term, the previous post link will lead to the previous post in any of those terms.

- `true`
- `false`

Default: false

excluded_terms

(string/array) (*optional*) Array or a comma-separated list of numeric terms IDs from which the next post should not be listed. For example `array(1, 5)` or `'1,5'`. This argument used to accept a list of IDs separated by `'and'`, this was deprecated in WordPress 3.3

Default: *None*

taxonomy

(string) (*Optional*) Taxonomy, if `$in_same_term` is true. Added in WordPress 3.8.

Default: `'category'`

Examples

Default Usage

Displays link with left angular quote («) followed by the post title of the previous post (chronological post date order).

« Previous Post Title

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

10)posts_nav_link()

Description

Displays links for next and previous pages. Useful for providing "paged" navigation of index, category and archive pages.

For displaying next and previous pages of posts
see next_posts_link() and previous_posts_link().

For displaying next and previous post navigation on individual posts,
see next_post_link()and previous_post_link().

Usage

```
<?php posts_nav_link( $sep, $prelabel, $nextlabel ); ?>
```

Note: since weblog posts are traditionally listed in reverse chronological order (with most recent posts at the top), there is some ambiguity in the definition of "next page". WordPress defines "next page" as the "next page *toward the past*". In WordPress 1.5, the default Kubrick theme addresses this ambiguity by labeling the "next page" link as "previous entries". See Example: Kubrick Theme Format.

Parameters

$sep

(*string*) Text displayed between the links.

- Defaults to `' :: '` in 1.2.x.
- Defaults to `' – '` in 1.5.

$prelabel

(*string*) Link text for the previous page.

- Defaults to `'<< Previous Page'` in 1.2.x.
- Defaults to `'« Previous Page'` in 1.5.

$nxtlabel

(*string*) Link text for the next page.

- Defaults to `'Next Page >>'` in 1.2.x.
- Defaults to `'Next Page »'` in 1.5

Examples

Default Usage

By default, the `posts_nav_link()` look like this:

« Previous Page — Next Page »

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

11)post_class();

Description

WordPress theme authors who want to have finer css control options for their post styling, have the **post_class** function available. When the `post_class` function is added to a tag **within the loop**, for example `<div <?php post_class(); ?> >`, it will print out and add various post-related classes to the div tag. It can also be used **outside the loop** with the optional *post_id* parameter. This function is typically used in the index.php, single.php, and other template files that feature hierarchical post listings. If you would prefer to have the post classes returned instead of echoed, you would want to use get_post_class(). **Note:** get_post_class() does not return a string, but an array that must be processed to produce text similar to what is echoed by *post_class()*.

For css classes intended to help target entire pages, see body_class(), and for classes targeting comment listings, see comment_class().

Usage

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

The `post_class` may include one or more of the following values for the class attribute, dependent upon the pageview.

- .post-[id]
- .[post-type]
- .type-[post-type]
- .status-[post-status]
- .format-[post-format] (default to 'standard')
- .post-password-required
- .post-password-protected
- .has-post-thumbnail
- .sticky
- .hentry (hAtom microformat pages)
- .[taxonomy]-[taxonomy-slug] (includes category)
- .tag-[tag-name]

Default Values

The `post_class` CSS classes appear based upon the post pageview Conditional Tags as follows.

Front Page
If posts are found on the front page of the blog, be it static or not, the class selectors are: **post post-id hentry**

Single Post
Single post template files and pageviews feature the class selectors: **post post-id hentry**

Sticky Post
Posts designated as "sticky," sticking to the front page of the blog, feature the class selector: **sticky**

Author
Author template files and pageviews displaying posts feature the class selectors: **post post-id**

Category
Category template files and pageviews displaying posts feature the class selectors: **post post-id category-ID category-name**

Tags
Tag template files and pageviews with posts feature the class selectors: **tag-name post post-id**

Archives
Archive pageviews and pageviews with posts feature CSS classes: **post post-id**

Search
Search template files and pageviews with posts feature the class selectors: **post post-id**

Attachment
Attachment template files and pageviews feature the class selectors: **attachment post post-id**

Format
Posts using Post Formats feature the class selector: **format-name**

Parameters

How to pass parameters to tags with PHP function-style parameters

`class`

(*string or array*) (*optional*) One or more classes to add to the class attribute, separated by a single space.

Default: null

`$post_id`

(*int*) (*optional*) An optional post ID, used when calling this function from outside The Loop

Default: null

Examples

The following example shows how to implement the `post_class` template tag into a theme.

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

Post Thumbnail tags

1) `has_post_thumbnail();`

`has_post_thumbnail( int|WP_Post $post = null )`

Check if post has an image attached.

Parameters

$post

(int|WP_Post) (Optional) Post ID or WP_Post object. Default is global `$post`.
Default value: null

Return

(bool) Whether the post has an image attached.

This example first checks if there is a Post Thumbnail (aka Featured Image) set for the current queried item. If there is a Post Thumbnail set, it returns the Post Thumbnail. If not, it echoes out a default image which should be located in the current theme's image folder (assuming the folder is in the theme's root directory).

```
1   <?php
2   // Must be inside a loop.
3
    if ( has_post_thumbnail() ) {
```

```
4           the_post_thumbnail();
5       }
6       else {
7           echo '<img src="' . get_bloginfo( 'stylesheet_directory' )
8               . '/images/thumbnail-default.jpg" />';
9       }
10      ?>
11
```

You can use `set_post_thumbnail_size()` to set a default size for your thumbnail. Alternatively, you can add new image sizes to the defaults by use `add_image_size()`.

2)get_post_thumbnail_id()

Description

1. If a featured image (formerly known as post thumbnail) is set - Returns the ID of the featured image attached to the post
2. If no such attachment exists, the function returns an empty string
3. If the post does not exist, the function returns `false`

Note: To enable featured images, see post thumbnails, the current theme must include `add_theme_support( 'post-thumbnails' );` in its functions.php file. See also Post Thumbnails.

Usage

```
<?php $post_thumbnail_id = get_post_thumbnail_id( $post_id ); ?>
```

Parameters

$post

(*integer/WP_Post*) (*Optional*) Post ID or WP_Post object. If `null`, the current post will be used.

Default: `null`

Return Values

(*string*)

The ID of the post, or an empty string on failure.

Examples

Show all attachments for the current post except the Featured Image

To get all post attachments except the Featured Image, you can use this function with something like get_posts().

Do this inside The Loop (where `$post->ID` is available).

```
<?php

$args = array(

        'post_type'   => 'attachment',

        'numberposts' => -1,

        'post_status' => 'any',

        'post_parent' => $post->ID,

        'exclude'     => get_post_thumbnail_id(),

);

$attachments = get_posts( $args );

if ( $attachments ) {

        foreach ( $attachments as $attachment ) {

                echo apply_filters( 'the_title', $attachment->post_title );

                the_attachment_link( $attachment->ID, false );

        }

}?>
```

Notes

- "Post Thumbnail" is an outdated term for "Featured Image". This function returns the ID of the post's featured image. It does not return IDs of other images attached to posts that are thumbnail sized.

3)the_post_thumbnail();

the_post_thumbnail(*string|array* $size = 'post-thumbnail', *string|array* $attr = '')

Display the post thumbnail.

Description

When a theme adds 'post-thumbnail' support, a special 'post-thumbnail' image size is registered, which differs from the 'thumbnail' image size managed via the Settings > Media screen.

When using the_post_thumbnail() or related functions, the 'post-thumbnail' image size is used by default, though a different size can be specified instead as needed.

Parameters

$size

(string|array) (Optional) Image size to use. Accepts any valid image size, or an array of width and height values in pixels (in that order).
Default value: 'post-thumbnail'

$attr

(string|array) (Optional) Query string or array of attributes.
Default value: ''

Post thumbnail sizes:

```
1    //Default WordPress
2    the_post_thumbnail( 'thumbnail' );       // Thumbnail (150 x 150 hard cropped)
3    the_post_thumbnail( 'medium' );          // Medium resolution (300 x 300 max height
4    the_post_thumbnail( 'medium_large' );    // Medium Large (added in WP 4.4) resolutio
     the_post_thumbnail( 'large' );           // Large resolution (1024 x 1024 max height
```

```
5       the_post_thumbnail( 'full' );              // Full resolution (original size u
6
7       //With WooCommerce
8       the_post_thumbnail( 'shop_thumbnail' ); // Shop thumbnail (180 x 180 hard
9       the_post_thumbnail( 'shop_catalog' );    // Shop catalog (300 x 300 hard cr
        the_post_thumbnail( 'shop_single' );     // Shop single (600 x 600 hard cro
10
11
```

1)get_the_post_thumbnail();

get_the_post_thumbnail(*int|WP_Post* $post = null, *string|array* $size = 'post-thumbnail', *string|array* $attr = '')

Retrieve the post thumbnail.

Description

When a theme adds 'post-thumbnail' support, a special 'post-thumbnail' image size is registered, which differs from the 'thumbnail' image size managed via the Settings > Media screen.

When using the_post_thumbnail() or related functions, the 'post-thumbnail' image size is used by default, though a different size can be specified instead as needed.

Parameters

$post

(int|WP_Post) (Optional) Post ID or WP_Post object. Default is global $post.
Default value: null

$size

(string|array) (Optional) Image size to use. Accepts any valid image size, or an array of width and height values in pixels (in that order).
Default value: 'post-thumbnail'

$attr

(string|array) (Optional) Query string or array of attributes.
Default value: ''

Return

(string) The post thumbnail image tag.

1. **Thumbnail Sizes**
The default image sizes of WordPress are "thumbnail", "medium", "large" and "full" (the size of the image you uploaded).
These image sizes can be configured in the WordPress Administration Media panel under Settings > Media.
Themes may also add "post-thumbnail". This is how you can use these default sizes with `get_the_post_thumbnail()`:

```
1
2   // without parameter -> Post Thumbnail (as set by theme using set_post_thumbnail_si
3   get_the_post_thumbnail( $post_id );
4
5   get_the_post_thumbnail( $post_id, 'thumbnail' );       // Thumbnail (Note: different
6   get_the_post_thumbnail( $post_id, 'medium' );          // Medium resolution
7   get_the_post_thumbnail( $post_id, 'large' );           // Large resolution
8   get_the_post_thumbnail( $post_id, 'full' );            // Original resolution
9
10  get_the_post_thumbnail( $post_id, array( 100, 100) ); // Other resolutions
```

2. Register new image sizes for Post Thumbnails with: `add_image_size()`.
To set the default size for Post Thumbnails see: `set_post_thumbnail_size()`.

Navigation Menu tags

1) `wp_nav_menu( array $args = array() )`

Displays a navigation menu.

Parameters

$args

(array) (Optional) Array of nav menu arguments.

- **'menu'**
  (int|string|WP_Term) Desired menu. Accepts (matching in order) id, slug, name, menu object.
- **'menu_class'**
  (string) CSS class to use for the ul element which forms the menu. Default 'menu'.
- **'menu_id'**
  (string) The ID that is applied to the ul element which forms the menu. Default is the menu slug, incremented.
- **'container'**
  (string) Whether to wrap the ul, and what to wrap it with. Default 'div'.
- **'container_class'**
  (string) Class that is applied to the container. Default 'menu-{menu slug}-container'.
- **'container_id'**
  (string) The ID that is applied to the container.
- **'fallback_cb'**
  (callable|bool) If the menu doesn't exists, a callback function will fire. Default is 'wp_page_menu'. Set to false for no fallback.
- **'before'**
  (string) Text before the link markup.
- **'after'**
  (string) Text after the link markup.
- **'link_before'**
  (string) Text before the link text.
- **'link_after'**
  (string) Text after the link text.
- **'echo'**
  (bool) Whether to echo the menu or return it. Default true.
- **'depth'**
  (int) How many levels of the hierarchy are to be included. 0 means all. Default 0.
- **'walker'**
  (object) Instance of a custom walker class.

- **'theme_location'**
  (string) Theme location to be used. Must be registered with register_nav_menu() in order to be selectable by the user.
- **'items_wrap'**
  (string) How the list items should be wrapped. Default is a ul with an id and class. Uses printf() format with numbered placeholders.
- **'item_spacing'**
  (string) Whether to preserve whitespace within the menu's HTML. Accepts 'preserve' or 'discard'. Default 'preserve'.

Default value: array()

Return

(string|false|void) Menu output if $echo is false, false if there are no items or no menu was found.

Default example

Shows the first non-empty menu or `wp_page_menu()`.

```
1    <?php wp_nav_menu(); ?>
```

Targeting a specific menu

```
    wp_nav_menu( array(
1       'theme_location' => 'Primary'
2   ) );
3
```

Conditional tags

1) Is_archive()

Description

This Conditional Tag checks if any type of Archive page is being displayed. An Archive is a Category, Tag, Author, Date, Custom Post Type or Custom Taxonomy based pages. This is a boolean function, meaning it returns either TRUE or FALSE.

Usage

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

Return Values

(boolean)

True on success, false on failure.

Examples

```
<?php
if ( is_archive() ) {
    // write your code here ...
}
?>
```

Notes

Custom Post Types

is_archive() does not accept any parameters. If you want to check if this is the archive of a custom post type, use is_post_type_archive($post_type)

2) Is_category

`is_category( mixed $category = '' )`

Is the query for an existing category archive page?

Description

If the $category parameter is specified, this function will additionally check if the query is for one of the categories specified.

Parameters

$category

(mixed) (Optional) Category ID, name, slug, or array of Category IDs, names, and slugs.
Default value: ''

Return

(bool)

Examples

```
1   is_category();
2   // When any Category archive page is being displayed.
3
4   is_category( '9' );
5   // When the archive page for Category 9 is being displayed.
6
7   is_category( 'Stinky Cheeses' );
8   // When the archive page for the Category with Name "Stinky Cheeses" is being dis
9
10  is_category( 'blue-cheese' );
11  // When the archive page for the Category with Category Slug "blue-cheese" is bei
12
13  is_category( array( 9, 'blue-cheese', 'Stinky Cheeses' ) );
14  // Returns true when the category of posts being displayed is either term_ID 9,
15  // or slug "blue-cheese", or name "Stinky Cheeses".
16  // Note: the array ability was added in version 2.5.
```

3)is front page()

Description

This Conditional Tag checks if the main page is a posts or a Page. This is a boolean function, meaning it returns either TRUE or FALSE. It returns TRUE when the main blog page is being displayed and the **Settings->Reading->Front page displays** is set to "Your latest posts", **or** when is set to "A static page" and the "Front Page" value is the current Page being displayed.

Usage

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

Parameters

This tag does not accept any parameters.

Return Values

(boolean)

True on success, false on failure.

Examples

If you are using a static page as your front page, this is useful:

```
<title>

<?php bloginfo('name'); ?> » <?php is_front_page() ?
bloginfo('description') : wp_title(''); ?>
```

```
</title>
```

4)is_home()

Determines if the query is for the blog homepage.

Description

The blog homepage is the page that shows the time-based blog content of the site.

is_home() is dependent on the site's "Front page displays" Reading Settings 'show_on_front' and 'page_for_posts'.
If a static page is set for the front page of the site, this function will return true only on the page you set as the "Posts page".

Return

(bool) True if blog view homepage, otherwise false.

example

```
if ( is_home() ) {
    // This is the blog posts index
    get_sidebar( 'blog' );
} else {
    // This is not the blog posts index
    get_sidebar();
}
```

5) is_page()

is_page(*int|string|array* $page = '')

Is the query for an existing single page?

Description #Description

If the $page parameter is specified, this function will additionally check if the query is for one of the pages specified.

Parameters

$page

(int|string|array) (Optional) Page ID, title, slug, or array of such.
Default value: ''

Return

(bool) Whether the query is for an existing single page.

- Will return true if an empty value is passed
- Due to certain global variables being overwritten during The Loop, is_page() will not work. In order to call it after The Loop, you must call wp_reset_query() first.

```
// When any single Page is being displayed.
is_page();

// When Page 42 (ID) is being displayed.
is_page( 42 );

// When the Page with a post_title of "Contact" is
being displayed.
is_page( 'Contact' );

// When the Page with a post_name (slug) of "about-me"
is being displayed.
is_page( 'about-me' );

/*
```

```
 * Returns true when the Pages displayed is either post
ID 42,
 * or post_name "about-me", or post_title "Contact".
 * Note: the array ability was added in version 2.5.
 */
is_page( array( 42, 'about-me', 'Contact' ) );
```

6) is_single()

is_single(*int|string|array* $post = '')

Is the query for an existing single post?

Description

Works for any post type, except attachments and pages

If the $post parameter is specified, this function will additionally check if the query is for one of the Posts specified.

Parameters

$post

(int|string|array) (Optional) Post ID, title, slug, or array of such.
Default value: ''

Return

(bool) Whether the query is for an existing single post.

Example:-

```
is_single();
// When any single Post page is being displayed.

is_single('17');
```

```
// When Post 17 (ID) is being displayed.

is_single(17);
// When Post 17 (ID) is being displayed. Integer
parameter also works
is_single('Irish Stew');
// When the Post with post_title of "Irish Stew" is
being displayed.

is_single('beef-stew');
// When the Post with post_name (slug) of "beef-stew"
is being displayed.

is_single(array(17,'beef-stew','Irish Stew'));
// Returns true when the single post being displayed is
either post ID 17,
// or the post_name is "beef-stew", or the post_title
is "Irish Stew".
```

7)is search()

Description

This Conditional Tag checks if search result page archive is being displayed. This is a boolean function, meaning it returns either TRUE or FALSE.

Usage

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

Parameters

This tag does not accept any parameters.

Return Values