CS-16: Content Management System using WordPress

B.C.A. Semester - 3

Topic-4

Theme Development

- Anatomy of a Theme: header.php, footer.php and sidebar.php
- Template Files (style.css, index.php, page.php, home.php, archive.php, single.php, comments.php, search.php, attachment.php, 404.php, category.php, tag.php, author.php, date.php)
- The Loop (have_posts (), the_post())
- Template Tags
  1. General tags (wp_head(), get_footer(), get_header(), get_sidebar(), get_search_form(), bloginfo(), wp_title(), single_post_title(), wp_footer(), comments_template(), add_theme_support(), get_template_directory_uri(), body_class())
  2. Author tags (the_author(), get_the_author(), the_author_link(), get_the_author_link(), the_author_meta(), the_author_posts())
  3. Category tags (category_description(), single_cat_title(), the_category())
  4. Link tags (the_permalink(), get_permalink(), home_url(), get_home_url(), site_url(), get_site_url())
  5. Post tags (the_content(), the_excerpt(), the_ID(), the_tags(), the_title(), get_the_title(), the_date(), get_the_date(), the_time(), next_post_link(), previous_post_link(), posts_nav_link(), post_class())
  6. Post Thumbnail tags (has_post_thumbnail(), get_post_thumbnail_id(), the_post_thumbnail(), get_the_post_thumbnail())
  7. Navigation Menu tags (wp_nav_menu())
  8. Conditional Tags (is_archive(), is_category(), is_front_page(), is_home(), is_page(), is_single(), is_search(), is_attachment(), is_active_sidebar())
- functions.php file

MR. Gaurav.k.sardhara

9067351366

Why WordPress Themes?

WordPress Themes are files that work together to create the design and functionality of a WordPress site. Each Theme may be different, offering many choices for site owners to instantly change their website look.

You may wish to develop WordPress Themes for your own use, for a client project or to submit to the WordPress Theme Directory. Why else should you build a WordPress Theme?

- To create a unique look for your WordPress site.
- To take advantage of templates, template tags, and the WordPress Loop to generate different website results and looks.
- To provide alternative templates for specific site features, such as category pages and search result pages.
- To quickly switch between two site layouts, or to take advantage of a Theme or style switcher to allow site owners to change the look of your site.

A WordPress Theme has many benefits, too.

- It separates the presentation styles and template files from the system files so the site will upgrade without extreme changes to the visual presentation of the site.
- It allows for customization of the site functionality unique to that Theme.
- It allows for quick changes of the visual design and layout of a WordPress site.
- It removes the need for a typical WordPress site owner to have to learn CSS, HTML, and PHP in order to have a great-looking website.

Why should you build your own WordPress Theme? That's the real question.

- It's an opportunity to learn more about CSS, HTML, and PHP.
- It's an opportunity to put your expertise with CSS, HTML, and PHP to work.
- It's creative.
- It’s fun (most of the time).
- If you release it to the public, you can feel good that you shared and gave something back to the WordPress Community

Theme Development Standards

WordPress Themes should be coded using the following standards:

- Use well-structured, error-free PHP and valid HTML. See WordPress Coding Standards.
- Use clean, valid CSS. See CSS Coding Standards.
- Follow design guidelines in Site Design and Layout.

The Basic Elements

All WordPress themes are made up of a few essential elements. They are:

- HTML: The basic building block of all websites, HTML stands for HyperText Markup Language and uses elements enclosed in tags (<tag>like this<tag>), most of the time to help web browsers identify how a web page should look.
- PHP: A scripting language that runs server-side, PHP is used more often to generate elements of a web page. It is typically embedded within an HTML file.
- CSS: This stands for Cascading Style Sheets and is a quick way to change the look and formatting of an entire website just by modifying one file: style.css. So long as each of the files within a site call up the CSS file, those pages will adopt the style elements identified there, from fonts, colors, tables, and so forth.
- Images: Usually JPEG or PNG files, images bring a WordPress site wonderful.

The Home Page

Since WordPress runs primarily on PHP, so the home page is actually made up of two files:

- index.php
- home.php

When you visit the home page of a WordPress site in your web browser, the HTML will call up these two PHP files. Then, these files will call up additional files to generate the website on the spot. It's pretty cool how dynamic all of this is.

To get more specific, the index.php file calls up the following files:

- header.php
- sidebar.php
- footer.php

If you've gone around at the internal workings (the PHP files) in themes at all, you probably understand what the above files are referring to.

- The header.php file brings up the content you want to appear at the top of the homepage. This usually includes the name of the site and top navigation elements.
- The sidebar.php file calls up the sidebar portion of the theme, which is where most people place their social media stuff, site categories, recent posts, and recent comments and so on.
- Finally, there's the footer.php file, which calls up the footer portion of the site.

The index.php files tells the header, sidebar, and footer where to appear on the page and each of these subsequent files tells the browser what content to display in those areas.

Individual Posts and Pages

On most WordPress themes, the attributes for individual posts are carried in the single.php file. When you click on the "Read more," link on the teaser (hypertext) for a post on the homepage of a blog, you'll be taken to that individual post. While you can have many posts on a site, how they look and function are determined by the single.php file. It's a template, basically.

The same can be said for individual pages. You know, your "About Me," "Contact," and other static pieces of content? Page attributes are carried by the page.php file.

Categories, Tags, Search, 404 and Comments

Similar things happen server-side for the categories and tags pages. Individual category pages are generated using category.php and archive.php and tags are handled by tag.php.

So, when you create new categories or new tags, they'll automatically be included on their respective pages, with all of their associated content included. Again, it's pretty cool to think how all of this happens when you click on a link. It's all disassembled content, and with one click, it's cobbled together.

If the theme you're using has the option of adding a search box (which they all do), the search results will appear thanks to search.php. And if someone makes a boo-boo when typing in an address or a link is broken, that'll be handled by 404.php. Can you guess what comments.php controls? Yup. The comments section on each post is handled by this file.

It's a pretty simple concept to understand when you get right down to it and every WordPress theme uses these bones and muscles to hold the structure of the site together. Without these elements, the site wouldn't exist in any usable sort of way. It'd just be a bunch of disparate components. A header floating here. A footer floating there. It'd be a headless, footless, beast of a thing and we definitely wouldn't want that!

Anatomy of a Theme

WordPress Themes live in subdirectories of the WordPress themes directory (wp-content/themes/ by default) which cannot be directly moved using the wp-config.php file. The Theme's subdirectory holds all of the Theme's stylesheet files, template files, and optional functions file (functions.php), JavaScript files, and images. For example, a Theme named "test" would reside in the directory wp-content/themes/test/. Avoid using numbers for the theme name, as this prevents it from being displayed in the available themes list.

WordPress includes a default theme in each new installation. Examine the files in the default theme carefully to get a better idea of how to build your own Theme files.

WordPress Themes typically consist of three main types of files, in addition to images and JavaScript files.

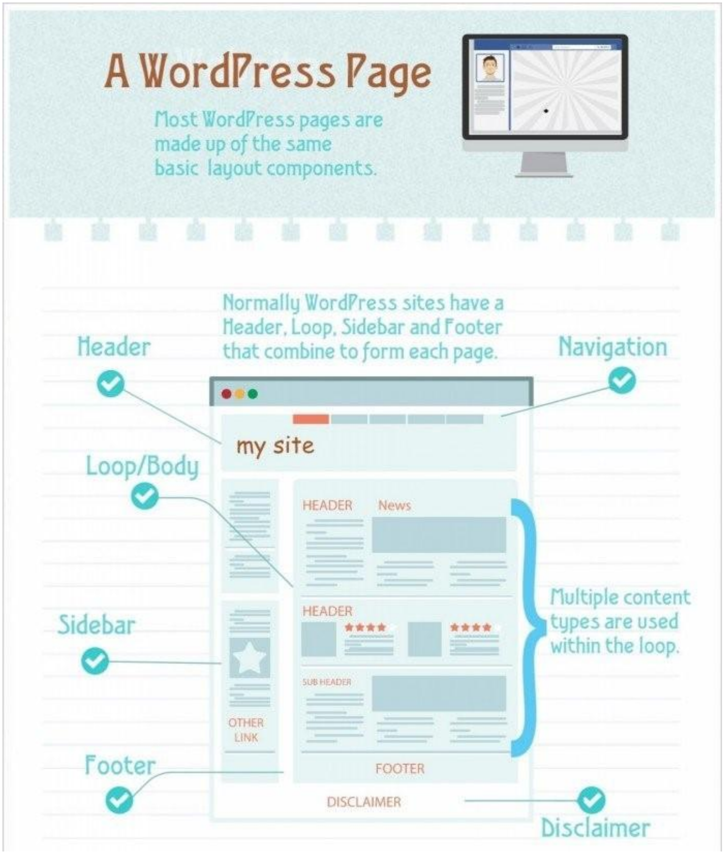
1. The stylesheet called style.css, which controls the presentation (visual design and layout) of the website pages.
2. WordPress template files which control the way the site pages generate the information from your WordPress database to be displayed on the site.
3. The optional functions file (functions.php) as part of the WordPress Theme files.

Building a WordPress Theme from Scratch

Building your own WordPress theme is a brilliant idea to showcase your site’s originality and authority. In this part of the theme development we’re going to standardize some things and finish up without theme’s ‘base’ ready to style it and make it look beautiful. This time we’ll be adding 3 new files and modifying index.php.

The Basic Anatomy of a WordPress Theme

Thanks to one of my favorite WordPress Developers site, Yoast, there’s a sweet infographic showing the anatomy of a WordPress theme for you to check out. I’ll let you have a look at it if you want but the basic anatomy uses templates and we use the WordPress Template Tags to include it in out pages. A normal layout is as follows.


A WordPress Page
Most WordPress pages are made up of the same basic layout components.
Normally WordPress sites have a Header, Loop, Sidebar and Footer that combine to form each page.
Header
Navigation
my site
Loop/Body
HEADER
News
HEADER
SUB HEADER
Sidebar
Multiple content types are used within the loop.
OTHER LINK
Footer
FOOTER
DISCLAIMER
Disclaimer

- Header: We create the template file header.php and use a template tag to include it in our pages. This is often where the site title and tag line are coded as well as the main site navigation.
- Loop: The loop is perhaps the most powerful part of WordPress. We use it to pull our posts from the database into the html and display it. It’s normally included in the page template your using but you can easily selectively use custom loop templates.
- Sidebar: Most sites make use of a sidebar for some reason or another. It’s a common feature online and WordPress has a template tag to pull it into our pages.
- Footer: This is there in most themes, even if they don’t have a visible footer. It’s the place where you close all your open div tags and place your ending html tag.

WordPress Theme Template Tags

There is a list of template tags available that allow you to include a file within other templates. The most common ones are used for the header, sidebar and footer. The index.php file is the one that gets called when any of the pages on our site are requested. Within that file we use the template tags to pull in our other files.

What to Include in Header.php

WordPress themes should always include valid markup and that includes defining your doctype. Header.php is the first file that will be called to build every page of your site so you should start with a valid doctype and luckily WordPress.org provides a valid HTML5 head section for us to use and we will be using html5 markup as we improve the framework.

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>
<head>
	<meta charset="<?php bloginfo( 'charset' ); ?>" />
	<title><?php wp_title(); ?></title>
	<link rel="profile" href="http://gmpg.org/xfn/11" />
	<link rel="stylesheet" href="<?php echo get_stylesheet_uri(); ?>"
type="text/css" media="screen" />
	<link rel="pingback" href="<?php bloginfo( 'pingback_url' ); ?>" />
	<?php if ( is_singular() && get_option( 'thread_comments' ) )
wp_enqueue_script( 'comment-reply' ); ?>
	<?php wp_head(); ?>
</head>
```

After that is the first point where you’ll actually be showing your content so the first thing we need to do is wrap everything in a wrapper so we can style widths and borders if we want them. Then we’ll be building the visible header with a site title and tagline the same as we did in the first part of the build. In future we’ll be including site navigation in the header as well. Get the full code in the source files.

What to do With Sidebar.php

The sidebar is pretty self-explanatory, it’s the bar that goes down one side of your page, or sometimes both. A few years ago themes would be refered to as ‘widgetized’ or ‘widget ready’. Nowadays it’s expected that all themes are widgetized so that’s what will be done in the sidebar.php file. For now it just has static html. The full code for sidebar.php is in the source files.

Closing up With Footer.php

We will simply be closing up our tags inside footer.php but later on we’ll be adding a few flourishes and widget areas to make our theme more functional. The full code is in the source files.

Template Files

Template files are the building blocks of your WordPress site. They fit together like the pieces of a puzzle to generate the web pages on your site. Some templates (the header and footer template files for example) are used on all the web pages, while others are used only under specific conditions.
A traditional web page consists of two files:

- The XHTML page to hold the structure and content of the page and
- the CSS Style Sheet which holds the presentation styles of the page.

In WordPress, the (X)HTML structure and the CSS style sheet are present but the content is generated "behind the scenes" by various template files. The template files and the style sheet are stored together as a WordPress Theme.

<u>Concept</u>

Templates are PHP source files used to generate the pages requested by visitors, and are output as HTML. Template files are made up of HTML, PHP, and WordPress Template Tags.

Let's look at the various templates that can be defined as part of a Theme.

WordPress allows you to define separate templates for the various aspects of your site. It is not essential, however, to have all these different template files for your site to fully function. Templates are chosen and generated based upon the Template Hierarchy, depending upon what templates are available in a particular Theme.

As a Theme developer, you can choose the amount of customization you want to implement using templates. For example, as an extreme case, you can use only one template file, called index.php as the template for all pages generated and displayed by the site. A more common use is to have different template files generate different results, to allow maximum customization.

Using a WordPress theme is one of the fastest and easiest ways to get a high quality website. A good theme can allow you to implement all the latest design trends and get a site up the same day (5 minutes to install). Knowing the anatomy of a WordPress theme can help you get a better understanding of how a WordPress theme works.

The best thing about themes for many people is how simple and easy it makes your work. All you need is some basic knowledge to carry out complex tasks. There are many things a WordPress user can do without having any technical background at all.

Below is an example of an extremely simple index.php file for a blog using pseudo code.

```
start
        load header
        load sidebar
        if there are posts:
                load posts until there are no more
        load footer
end
```

So when you type in a URL in the browser, the code will run and generate a WordPress webpage. You will notice functions like <u>get_header()</u> and <u>get_footer()</u>. These will render the different blocks on your page. In the example above you can see that the blocks are loaded in a particular order. First the header, then the sidebar, then posts, then the content() adds a filter and finally the footer is loaded.

This is a very simplistic example. In most themes you will find much more code in the index.php file. Most of the time the index.php file will reference many other files located on your web server. This includes the following file types:

CSS – for styles such as mobile themes.
JS – Javascript files which handles dynamic behavior such as animation.
Img – images for icons, buttons, logos etc.
Language – for translation.

Template Files List

Files Needed to Make a Theme

All that is necessary to make a WordPress theme are two files.

- index.php – creates the blog.
- style.css – styles your website.

While the files mentioned above are all that is needed, there are almost always more files involved. Below is a list of some of the most common files found in a WordPress theme.

- header.php
  generates the header, which is usually made up of an image and a nav menu.
- footer.php
  generates the footer which typically contains widgets, links and contact information.
- sidebar.php
  sidebars which can contain widgets. The one on this page has category links, recent posts with thumbnails and a social sharing widget.
- functions.php
  adds functions to your site related to the theme/framework. It works like a WordPress plugin.

Here is the list of the Theme files recognized by WordPress. Of course, your Theme can contain any other stylesheets, images, or files. Just keep in mind that the following have special meaning to WordPress -- see Template Hierarchy for more information.

- style.css
  The main stylesheet. This must be included with your Theme, and it must contain the information header for your Theme.
- index.php
  The main template. If your Theme provides its own templates, index.php must be present.
- page.php
  The page template. Used when an individual Page is queried.
- home.php
  The home page template, which is the front page by default. If you use a static front page this is the template for the page with the latest posts.
- archive.php
  The archive template. Used when a category, author, or date is queried. Note that this template will be overridden by category.php, author.php, and date.php for their respective query types.
- single.php
  The single post template. Used when a single post is queried. For this and all other query templates, index.php is used if the query template is not present.
- comments.php
  The comments template.

- search.php
  The search results template. Used when a search is performed.

- attachment.php
  Attachment template. Used when viewing a single attachment.

- 404.php
  The 404 Not Found template. Used when WordPress cannot find a post or page that matches the query.

- category.php
  The category template. Used when a category is queried.

- tag.php
  The tag template. Used when a tag is queried.

- author.php
  The author template. Used when an author is queried.

- date.php
  The date/time template. Used when a date or time is queried. Year, month, day, hour, minute, second.

<u>Few More Files based on themes</u>

- front-page.php
  The front page template.

- single-{post-type}.php
  The single post template used when a single post from a custom post type is queried. For example, single-book.php would be used for displaying single posts from the custom post type named "book". index.php is used if the query template for the custom post type is not present.

- taxonomy.php
  The term template. Used when a term in a custom taxonomy is queried.

- image.php
  Image attachment template. Used when viewing a single image attachment. If not present, attachment.php will be used.

- rtl.css
  The rtl stylesheet. This will be included automatically if the website's text direction is right-to-left. This can be generated using the RTLer plugin.

These files have a special meaning with regard to WordPress because they are used as a replacement for index.php, when available, according to the Template Hierarchy, and when the corresponding Conditional Tag returns true. For example, if only a single post is being displayed, the is_single() function returns 'true', and, if there is a single.php file in the active Theme, that template is used to generate the page.

Main Stylesheet (style.css)

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 addition to CSS style information for your theme, style.css provides details about the Theme in the form of comments. The stylesheet must provide details about the Theme in the form of comments. No two Themes are allowed to have the same details listed in their comment headers, as this will lead to problems in the Theme selection dialog. If you make your own Theme by copying an existing one, make sure you change this information first.

In order for WordPress to recognize the set of theme template files as a valid theme, the style.css file needs to be located in the root directory of your theme, not a subdirectory.

Basic Structure

WordPress uses the header comment section of a style.css to display information about the theme in the Appearance (Themes) dashboard panel.

Example

Here is an example of the header part of style.css.

```
/*
Theme Name: Twenty Seventeen
Theme URI: https://wordpress.org/themes/twentyseventeen/
Author: the WordPress team
Author URI: https://wordpress.org/
Description: Twenty Seventeen brings your site to life with immersive featured
images and subtle animations. With a focus on business sites, it features multiple
sections on the front page as well as widgets, navigation and social menus, a logo,
and more.
Version: 1.0
License: GNU General Public License v2 or later
License URI: http://www.gnu.org/licenses/gpl-2.0.html
Text Domain: twentyseventeen
Tags: one-column, two-columns, right-sidebar, flexible-header, accessibility-ready,
custom-colors, custom-header, custom-menu, custom-logo, editor-style, featured-
images, footer-widgets, post-formats, rtl-language-support, sticky-post, theme-
options, threaded-comments, translation-ready
This theme, like WordPress, is licensed under the GPL.
Use it to make something cool, have fun, and share what you've learned with others.
*/
```

Items indicated with (*) are required for a theme in the WordPress Theme Repository.

- Theme Name (*): Name of the theme.
- Theme URI: The URL of a public web page where users can find more information about the theme.
- Author (*): The name of the individual or organization who developed the theme. Using the Theme Author’s wordpress.org username is recommended.
- Author URI: The URL of the authoring individual or organization.
- Description (*): A short description of the theme.
- Version (*): The version, written in X.X or X.X.X format.
- License (*): The license of the theme.
- License URI (*): The URL of the theme license.
- Text Domain (*): The string used for textdomain for translation.
- Tags: Words or phrases that allow users to find the theme using the tag filter.

After the required header section, style.css can contain anything a regular CSS file has.

The Loop

The most exciting part is being able to dynamically insert content, and in WordPress we do that with The Loop. It's the most important function of WordPress. All of your content is generated through a loop.

In the dashboard, if you click on Posts, you will see a "Hello, world!" post in there by default. Our goal is to display that post in the blog.

The Loop itself is quite simple.

```
<?php if ( have_posts() ) : while ( have_posts() ) : the_post(); ?>
      <!-- contents of the loop -->
<?php endwhile; endif; ?>
```

It explains itself – IF there are posts, WHILE there are posts, DISPLAY the post. Anything inside the loop will be repeated. For a blog, this will be the post title, the date, the content, and comments. Where each individual post should end is where the loop will end. We're going to add the loop to index.php.

Here's your new index file.

```
<?php get_header(); ?>
      <div class="row">
            <div class="col-sm-8 blog-main">
            <?php
            if ( have_posts() ) :
                  while ( have_posts() ) : the_post();
                        get_template_part( 'content', get_post_format() );
                  endwhile;
            endif;
            ?>
            </div> <!-- /.blog-main -->
      <?php get_sidebar(); ?>
      </div> <!-- /.row -->
<?php get_footer(); ?>
```

The only thing inside your loop is content.php, which will contain the contents of one single post. So open content.php and change the contents to this:

```
<div class="blog-post">
      <h2 class="blog-post-title"><?php the_title(); ?></h2>
      <p class="blog-post-meta"><?php the_date(); ?> by <a href="#"> <?php
the_author(); ?></a></p>
      <?php the_content(); ?>
</div><!-- /.blog-post -->
```

It's amazingly simple! the_title(); is the title of the blog post, the_date(); shows the date, the_author(); the author, and the_content(); is your post content. I added another post to prove at the loop is working.

Anatomy of a Template Tag

It provides a brief examination of the animal known as the WordPress template tag, to help those who may be new to WordPress and PHP understand what template tags are and how they're used.

A WordPress template tag is made up of three components:

- A PHP code tag

  WordPress is built with the PHP scripting language

  ```
  <?php ?>
  ```

  The above shows the opening (<?php) and closing (?>) tag elements used to embed PHP functions and code in a HTML document, i.e. web page. There are a number of ways to embed PHP within a page, but this is the most "portable," in that it works on nearly every web server - as long as the server supports PHP (typically a document's filename also needs to end with the extension .php, so the server recognizes it as a PHP document).

  Anything within this tag is parsed and handled by the PHP interpreter, which runs on the web server (the interpreter is the PHP engine that figures out what the various functions and code do, and returns their output). For our purposes, the PHP tag lets you place WordPress functions in your page template, and through these generate the dynamic portions of your blog.

- A WordPress function

  A WordPress or template function is a PHP function that performs an action or displays information specific to your blog. And like a PHP function, a WordPress function is defined by a line of text (of one or more words, no spaces), open and close brackets (parentheses), and typically a semi-colon, used to end a code statement in PHP. An example of a WordPress function is:

  ```
  the_ID();
  ```

  the_ID() displays the ID number for a blog entry or post. To use it in a page template, you slip it into the PHP tag shown above:

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

  This is now officially a WordPress template tag, as it uses the PHP tag with a WordPress function.

- Optional parameters

  The final item making up a template tag is one you won't necessarily make use of unless you want to customize the tag's functionality. This, or rather these, are the parameters or arguments for a function. Here is the template function bloginfo(), with the show parameter being passed the 'name' value:

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

  If your blog's name is Super Weblog, the bloginfo() template tag, when using 'name' as the show parameter value, will display that name where it's embedded in your page template.

  Not all template tags accept parameters (the_ID() is one), and those which do accept different ones based on their intended use, so that the_content() accepts parameters separate from those which get_calendar() can be passed

Template Tags

General Tags

- wp_head()

  Put this template tag immediately before </head> tag in a theme template (ex. header.php, index.php).

Usage: <?php wp_head(); ?>

Parameters: This function does not accept any parameters.

Return values: None.

Source File: wp_head() is located in wp-includes/general-template.php.

Examples: In twentyseventeen theme [wp-content/themes/twentyseventeen/header.php]

```
<?php
...
/* 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. */
    wp_head();
?>
</head>
```

- wp_footer()

  Put this template tag immediately before </body> tag in a theme template (ex. footer.php, index.php).

Usage: <?php wp_footer(); ?>

Parameters: This function does not accept any parameters.

Source File: wp_footer() is located in wp-includes/general-template.php.

Examples: In twentyseventeen theme [wp-content/themes/twentyseventeen/footer.php]

```
...
<?php
/* 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. */
      wp_footer();
?>
</body>
</html>
```

- get_header()

  Includes the header.php template file from your current theme's directory. If a name is specified then a specialized 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.

Usage: <?php get_header($name); ?>

Parameters: $name (string) (optional) Calls for header-name.php. Default: None

Source File: get_header() is located in wp-includes/general-template.php.

Examples: Simple 404 page

The following code is a simple example of a template for an "HTTP 404: Not Found" error (which you could include in your Theme as 404.php).

```
<?php get_header(); ?>
      <h2>Error 404 - Not Found</h2>
<?php get_sidebar(); ?>
<?php get_footer(); ?>
```

Multiple Headers: Different header for different pages.

```
<?php
      if ( is_home() ) :
            get_header( 'home' );
      elseif ( is_404() ) :
            get_header( '404' );
```

```
        else :
                get_header();
        endif;
?>
```

The file names for the home and 404 headers should be header-home.php and header-404.php respectively.

- get_footer()

  Includes the footer.php template file from your current theme's directory. if a name is specified then a specialized 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.

Usage: <?php get_footer($name); ?>
Parameters: $name (string) (optional) Calls for footer-name.php. Default: None
Return Values: None.
Source File: get_footer() is located in wp-includes/general-template.php.
Examples: [Simple 404 page]
The following code is a simple example of a template for an "HTTP 404: Not Found" error (which you could include in your Theme as 404.php).

```
<?php get_header(); ?>
        <h2>Error 404 - Not Found</h2>
<?php get_sidebar(); ?>
<?php get_footer(); ?>
```

Multiple Footers: Different footer for different pages.

```
<?php
if ( is_home() ) :
        get_footer( 'home' );
elseif ( is_404() ) :
        get_footer( '404' );
else :
        get_footer();
endif;
?>
```

The file names for the home and 404 footers should be footer-home.php and footer-404.php respectively.

- get_sidebar()

  Loads sidebar template. Includes the sidebar template for a theme or if a name is specified then a specialized 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 specialized sidebar. Default value: null
Source File: wp-includes/general-template.php
Example:

```
function get_sidebar( $name = null )
{       do_action( 'get_sidebar', $name );
        $templates = array();
        $name = (string) $name;
        if ( '' !== $name )
                $templates[] = "sidebar-{$name}.php";

        $templates[] = 'sidebar.php';
        locate_template( $templates, true );
}
```

- get_search_form()

  Displays the search form. 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
Return: (string|void) String when $echo is false.
Usage: `get_search_form( $echo );`
Source File: wp-includes/general-template.php

- bloginfo()

  Displays information about the current site.

Parameters: $show (string) (Optional) Site information to display. Default value: ""
<u>Possible values for$show</u>

- 'name' – Displays the "Site Title" set in Settings > General. This data is retrieved from the "blogname" record in the wp_options table.
- 'description' – Displays the "Tagline" set in Settings > General. This data is retrieved from the "blogdescription" record in the wp_options table.
- 'wpurl' – Displays the "WordPress address (URL)" set in Settings > General. This data is retrieved from the "siteurl" record in the wp_options table. Consider echoing site_url() instead, especially for multi-site configurations using paths instead of subdomains (it will return the root site not the current sub-site).
- 'url' – Displays the "Site address (URL)" set in Settings > General. This data is retrieved from the "home" record in the wp_options table. Consider echoing home_url() instead.
- 'admin_email' – Displays the "E-mail address" set in Settings > General. This data is retrieved from the "admin_email" record in the wp_options table.
- 'charset' – Displays the "Encoding for pages and feeds" set in Settings > Reading. This data is retrieved from the "blog_charset" record in the wp_options table. Note: this parameter always echoes "UTF-8", which is the default encoding of WordPress.
- 'version' – Displays the WordPress Version you use. This data is retrieved from the $wp_version variable set in wp-includes/version.php.
- 'html_type' – Displays the Content-Type of WordPress HTML pages (default: "text/html"). This data is retrieved from the "html_type" record in the wp_options table. Themes and plugins can override the default value using the pre_option_html_type filter.
- 'text_direction' – Displays the Text Direction of WordPress HTML pages. Consider using is_rtl() instead.
- 'language' – Displays the language of WordPress.
- 'stylesheet_url' – Displays the primary CSS (usually style.css) file URL of the active theme. Consider echoing get_stylesheet_uri() instead.
- 'stylesheet_directory' – Displays the stylesheet directory URL of the active theme. (Was a local path in earlier WordPress versions.) Consider echoing get_stylesheet_directory_uri() instead.
- 'template_url' / 'template_directory' – URL of the active theme's directory. Within child themes, both get_bloginfo('template_url') and get_template() will return the parent theme directory. Consider echoing get_template_directory_uri() instead (for the parent template directory) or get_stylesheet_directory_uri() (for the child template directory).
- 'pingback_url' – Displays the Pingback XML-RPC file URL (xmlrpc.php).
- 'atom_url' – Displays the Atom feed URL (/feed/atom).
- 'rdf_url' – Displays the RDF/RSS 1.0 feed URL (/feed/rfd).

- 'rss_url' – Displays the RSS 0.92 feed URL (/feed/rss).
- 'rss2_url' – Displays the RSS 2.0 feed URL (/feed).
- 'comments_atom_url' – Displays the comments Atom feed URL (/comments/feed).
- 'comments_rss2_url' – Displays the comments RSS 2.0 feed URL (/comments/feed).
- 'siteurl' – Deprecated since version 2.2. Echo home_url(), or use bloginfo('url').
- 'home' – Deprecated since version 2.2. Echo home_url(), or use bloginfo('url').

Source File: wp-includes/general-template.php

Example (1):
```
function bloginfo( $show = '' )
{      echo get_bloginfo( $show, 'display' );     }
```
Example (2):
```
<a href="<?php bloginfo('url'); ?>" title="<?php bloginfo('name'); ?>">
<?php bloginfo('name'); ?></a>
```
Example (3):
```
<?php if ( get_bloginfo( 'description' )  !== '' ) { ?>
<a class="site-description"><?php bloginfo( 'description' ); ?></a>
<?php } ?>
```

- wp_title()

  Display or retrieve page title for all areas of blog.

```
wp_title( string $sep = '&raquo;', bool $display = true, string $seplocation = '' )
```

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'.

Parameters:
- $sep (string) (Optional) default is '»'. How to separate the various items within the page title. Default value: '&raquo;'
- $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.

Example: The sep string may be zero characters, which will remove » from the returned value. To do this, set the sep parameter to zero characters, for example:
```
<title><?php wp_title(''); ?></title>
```

If the title of the post is "Hello world!", then the function will return: Hello world!

- single_post_title()

  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

Source File: single_post_title() is located in wp-includes/general-template.php.

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

 comments_template()

Load the comment template specified in $file.

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

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

Source File: wp-includes/comment-template.php

Default Usage: <?php comments_template(); ?>

Example: Alternative Comment Template

On some occasions you may want display your comments differently within your theme. For this you would build an alternate file (ex. short-comments.php) and call it as follows:

```
<?php comments_template( '/short-comments.php' ); ?>
```

The path to the file used for an alternative comments template should be relative to the current theme root directory, and include any subfolders. So if the custom comments template is in a folder inside the theme, it may look like this when called:

```
<?php comments_template( '/custom-templates/alternative-comments.php' ); ?>
```

 add_theme_support()

Registers theme support for a given feature.

```
add_theme_support( string $feature )
```

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.

Parameters:

- $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.

Source File: wp-includes/theme.php

Features

Post Formats

This feature enables Post Formats support for a theme. When using child themes, be aware that

```
add_theme_support( 'post-formats' )
```

will override the formats as defined by the parent theme, not add to it.

To enable the specific formats (see supported formats at Post Formats), use:

```
add_theme_support( 'post-formats', array( 'aside', 'gallery' ) );
```

To check if there is a 'quote' post format assigned to the post, use has_post_format():

```
// In your theme single.php, page.php or custom post type
if ( has_post_format( 'quote' ) )
{     echo 'This is a quote.';      }
```

Post Thumbnails

This feature enables Post Thumbnails support for a theme. Note that you can optionally pass a second argument, $args, with an array of the Post Types for which you want to enable this feature.

```
add_theme_support( 'post-thumbnails' );
add_theme_support( 'post-thumbnails', array( 'post' ) );          // Posts only
add_theme_support( 'post-thumbnails', array( 'page' ) );          // Pages only
add_theme_support( 'post-thumbnails', array( 'post', 'movie' ) ); // Posts and Movies
```

This feature must be called before the 'init' hook is fired. That means it needs to be placed directly into functions.php or within a function attached to the 'after_setup_theme' hook. For custom post types, you can also add post thumbnails using the register_post_type() function as well.

To display thumbnails in themes index.php or single.php or custom templates, use:

```
the_post_thumbnail();
```

To check if there is a post thumbnail assigned to the post before displaying it, use:

```
if ( has_post_thumbnail() )
{      the_post_thumbnail();    }
```

Custom Background

This feature enables Custom_Backgrounds support for a theme.

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

Note that you can add default arguments using:

```
$defaults = array (
      'default-image' => '',
      'default-preset' => 'default',
      'default-position-x' => 'left',
      'default-position-y' => 'top',
      'default-size' => 'auto',
      'default-repeat' => 'repeat',
      'default-attachment' => 'scroll',
      'default-color' => '',
      'wp-head-callback' => '_custom_background_cb',
      'admin-head-callback' => '',
      'admin-preview-callback' => '',
);
add_theme_support( 'custom-background', $defaults );
```

Custom Header

This feature enables Custom_Headers support for a theme.

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

Note that you can add default arguments using:

```
$defaults = array (
      'default-image' => '',
      'random-default' => false,
      'width' => 0,
      'height' => 0,
      'flex-height' => false,
      'flex-width' => false,
      'default-text-color' => '',
      'header-text' => true,
      'uploads' => true,
      'wp-head-callback' => '',
      'admin-head-callback' => '',
      'admin-preview-callback' => '',
      'video' => false,
      'video-active-callback' => 'is_front_page',
);
add_theme_support( 'custom-header', $defaults );
```

Custom Logo

This feature, first introduced in Version_4.5, enables Theme_Logo support for a theme.

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

Note that you can add default arguments using:

```
add_theme_support( 'custom-logo', array (
      'height'       => 100,
      'width'        => 400,
      'flex-height' => true,
      'flex-width'  => true,
      'header-text' => array( 'site-title', 'site-description' ),
) );
```

- get_template_directory_uri()

  Retrieve theme directory URI.

  ```
  get_template_directory_uri()
  ```

Return: (string) Template directory URI.

Source File: wp-includes/theme.php

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

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

- body_class()

  Display the classes for the body element.

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

Parameters:

- $class (string|array) (Optional) One or more classes to add to the class list. Default value: ''

Source File: wp-includes/post-template.php

Example:

```
function body_class( $class = ''  )
{      // Separates classes with a single space, collates classes for body element
       echo 'class="' . join( ' ', get_body_class( $class ) ) . '"';
}
```

Author Tags

Source File: located in wp-includes/author-template.php

- the_author()

  The author of a post can be displayed by using this Template Tag. This tag must be used within The Loop. To return to PHP rather than displaying, use get_the_author().

Usage: <?php the_author(); ?>

Examples: Display Author's 'Public' Name

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

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

- get_the_author()

  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(); ?>
```

- the_author_link()

  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(); ?>

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: <?php the_author_link(); ?></p>
```

- get_the_author_link()

  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(); ?>

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>
```

- the_author_meta()

  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

Example (1): 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 <?php the_author_meta('aim'); ?></p>
```

Example (2): Display a User Email Address

Displays the email address for user ID 25.

```
<p>The email address for user id 25 is <?php
the_author_meta('user_email',25); ?></p>
```

Example (3): Advanced Uses

A plugin may add an additional field in the registration or manage users, which adds a new value in the wp_usermeta table (where wp_ is your data base prefix). For this example we will use a Twitter ID. For a meta_key value of "twitter" and meta_value of "WordPress" then

```
<p>This author's Twitter name is <?php the_author_meta('twitter'); ?></p>
```

- the_author_posts()

  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><?php the_author(); ?> has blogged <?php the_author_posts(); ?> posts</p>
```

Category Tags

- category_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>
```

- single_cat_title()

  Display or retrieve page title for category archive.

  `single_cat_title( string $prefix = '', bool $display = true )`

  Useful for category template files for displaying the category page title. The prefix does not automatically place a space between the prefix, so if there should be a space, the parameter value will need to have it at the end.

Parameters:

- $prefix (string) (Optional) What to display before the title. Default value: ''
- $display (bool) (Optional) Whether to display or retrieve title. Default value: true

Return: (string|void) Title when retrieving.

Example:

```
function single_cat_title( $prefix = '', $display = true )
{      return single_term_title( $prefix, $display );   }
```

- the_category()

  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 of the current post. Default: false

Examples:

<u>Separated by Space</u>

List categories with a space as the separator.

```
<?php the_category( ' ' ); ?>
```

<u>Separated by Comma</u>

Displays links to categories, each category separated by a comma (if more than one).

```
<?php the_category( ', ' ); ?>
```

<u>Separated by Arrow</u>

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( '&gt; ' ); ?>
```

Separated by a Bullet

Displays links to categories with a bullet (•) separating the categories.

```
<?php the_category( '&bull;' ); ?>
```

Link Tags

- the_permalink()

  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(); ?>`

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>
```

- get_permalink()

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

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

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.

Source File: wp-includes/link-template.php

Example: Get current post/page url outside loop. $post is a global variable.

```
<?php echo get_permalink( $post->ID ); ?>
```

- home_url()

  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/`
```
        $url  = home_url( '/', 'https' );
        echo esc_url( $url );
```
Output: `https://www.example.com/`
```
        $url  = home_url( 'example', 'relative' );
        echo esc_url( $url );
```
Output: `/example`

 get_home_url()

Retrieves the URL for a given site where the front end is accessible.
```
        get_home_url( int $blog_id = null, string $path = '', string|null $scheme = null )
```
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.

Example: Basic Usage

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

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

 site_url()

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/
```

 get_site_url()

Retrieves the URL for a given site where WordPress application files (e.g. wp-blog-header.php or the wp-admin/ folder) are accessible.
```
        get_site_url( int $blog_id = null, string $path = '', string $scheme = null )
```

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:

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

Results in the full site URL being displayed: `http://www.example.com`

Post Tags

- the_content()

  Display the post content.

Usage: `the_content( string $more_link_text = null, bool $strip_teaser = false )`

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

Source File: wp-includes/post-template.php

Example: 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.

```
// Declare global $more (before the loop).
global $more;
// Set (inside the loop) to display content above the more tag.
$more = 0;
the_content( 'More ...' );
?>
```

If you need to display all of the content:

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

- the_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).

Note: If the current post is an attachment, such as in the attachment.php and image.php template loops, then the attachment caption is displayed. Captions do not include the "[…]" text.

Comparison with the <!–more–> quicktag

Excerpts provide an alternative to the use of the <!--more--> quicktag. Whereas this more tag requires a post author to manually create a 'split' in the post contents, which is then used to generate a "read more" link on index pages, the excerpts require, but do not necessarily demand, a post author to supply a 'teaser' for the full post contents.

The <!--more--> quicktag requires templates to use the_content() whereas using excerpts requires, and allows, template writers to explicitly choose whether to display full posts (using the_content()) or excerpts (using the_excerpt()).

Example: Use with Conditional Tags
Replaces the_content() tag with the_excerpt() when on archive or category pages.

```
<?php
        if ( is_category() || is_archive() )       {       the_excerpt();     }
        else  {       the_content();     }
?>
```

Example: Control Excerpt Length using Filters
By default, excerpt length is set to 55 words. To change excerpt length to 20 words using the excerpt_length filter,
add the following code to the functions.php file in your theme:

```
/*      * Filter the except length to 20 words.
        * @param int $length Excerpt length.
        * @return int (Maybe) modified excerpt length.   */
function wpdocs_custom_excerpt_length( $length )
{      return 20;   }
add_filter( 'excerpt_length', 'wpdocs_custom_excerpt_length', 999 );
```

- the_ID()

  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().
Example: Default Usage

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

Example: Post Anchor Identifier. Provides a unique anchor identifier to each post:

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

- the_tags()

  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.
Example: 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>
```

Example: Separated by Commas. Displays a list of the tags with a line break after it.

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

Example: 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: ',' > ' ); ?>
```

Example: 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 />' ); ?>
```

Example: A List Example. Displays a list of the tags as an unordered list:

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

- the_title()

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

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

 get_the_title()

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)

Example:

```
echo '<div class="breadcrumb">';
// If there is a parent, display the link.
$parent_title = get_the_title( $post->post_parent );
if ( $parent_title != the_title( ' ', ' ', false ) )
{
      echo '<a href="' . esc_url( get_permalink( $post->post_parent ) ) . '" alt="'
. esc_attr( $parent_title ) . '">' . $parent_title . '</a> » ';
}
// Then go on to the current page link.
echo '<a href="' . esc_url( get_permalink() ) . '" rel="bookmark" alt="' .
esc_attr( the_title_attribute() ) . '">' . the_title() . '</a>';
echo '</div>';
```

 the_date()

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. 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.

Example: Default Usage. Displays the date using defaults.

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

Example: Date as Year, Month, Date in Heading. Displays the date using the '2007-07-23' format (ex: 2004-11-30), inside an <h2> tag.

```
      <?php the_date('Y-m-d', '<h2>', '</h2>'); ?>
```

Example: Date in Heading Using $my_date Variable. Returns the date in the default format inside an <h2> tag and assigns it to the $my_date variable. The variable's value is then displayed with the PHP echo command.

```
<?php $my_date = the_date('', '<h2>', '</h2>', FALSE); echo $my_date; ?>
```

 get_the_date()

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)

Example: Default Usage

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

- the_time()

  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 ); ?>`

Parameter:

- $d (string) (optional) The format the time is to display in. Defaults to the time format configured in your WordPress options. Default: None

Example: Default Usage. Displays the time using your WordPress defaults.

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

Example: 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>
```

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

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

Example: 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>
```

Example: Displays the date and time.

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

- next_post_link()

  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 (boolean) (optional) Indicates whether next 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 next post link will lead to the next post in any of those terms. 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'

Example: 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(); ?>
```

Example: Bold Post Title As Link. Displays link with next chronological post's title wrapped in 'strong' tags (typically sets text to bold).

Next Post Title

```
<?php next_post_link( '<strong>%link</strong>' ); ?>
```

Example: Text As Link, Without Post Title, Within Same Category. Displays custom text as link to the next post within the same category as the current post. Post title is not included here. "Next post in category" is the custom text, which can be changed to fit your requirements.

Next post in category

```
<?php next_post_link( '%link', 'Next post in category', TRUE ); ?>
```

Example: Within Same Category, Excluding One. Displays link to next post in the same category, as long as it is not in category 13 (the category ID #). You can change the number to any category you wish to exclude. Exclude multiple categories by using " and " as a delimiter.

Next post in category

```
<?php next_post_link( '%link', 'Next post in category', TRUE, '13' ); ?>
```

Example: Within Same Taxonomy. Displays link to next post in the same taxonomy term. Post Formats are a taxonomy so the following would link to the next post with the same post format.

Next post in taxonomy

```
<?php next_post_link( '%link', 'Next post in taxonomy', TRUE, ' ', 'post_format' ); ?>
```

- previous_post_link()

  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. 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'

Example: 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(); ?>
```

Example: Bold Post Title As Link. Displays link with previous chronological post's title wrapped in 'strong' tags (typically sets text to bold).

Previous Post Title

```
<?php previous_post_link('<strong>%link</strong>'); ?>
```

Example: Text As Link, Without Post Title, Within Same Category. Displays custom text as link to the previous post within the same category as the current post. Post title is not included here. "Previous in category" is the custom text, which can be changed to fit your requirements.

Previous in category

```
<?php previous_post_link('%link', 'Previous in category', TRUE); ?>
```

Example: Within Same Category, Excluding One. Displays link to previous post in the same category, as long as it is not in category 13 (the category ID #). You can change the number to any category you wish to exclude. Array or comma-separated list of category ID(s) from which the previous post should not be listed. For example array(1, 5) or '1,5'.

Previous in category

```
<?php previous_post_link('%link', 'Previous in category', TRUE, '13'); ?>
```

Example: Within Same Taxonomy. Displays link to previous post in the same taxonomy term. Post Formats are a taxonomy so the following would link to the previous post with the same post format.

Previous post in taxonomy

```
<?php previous_post_link( '%link', 'Previous post in category', TRUE, ' ', 'post_fo
rmat' ); ?>
```

Example: Post Title As Link, Within Same Custom Taxonomy. Displays link to previous post in the same custom taxonomy term. You have a Custom Post Type called Buildings, and a custom taxonomy called Neighborhood. Here you don't need to use the Custom Post Type(as Custom Post Type already in the WP query). Just mention the taxonomy name(which is neighborhood as per this example) as last parameter of previous_post_link function.

Previous (Custom) Post Title in Neighborhood Taxonomy

```
<?php previous_post_link( '%link', '%title', TRUE, ' ', 'neighborhood' ); ?>
```

- posts_nav_link()

  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".

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

Example: Default Usage. By default, the posts_nav_link() look like this:

« Previous Page — Next Page »

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

Example: In Centered DIV. Displays previous and next page links ("previous page · next page") centered on the page.

```
<div style="text-align:center;">
<?php posts_nav_link( ' &#183; ', 'previous page', 'next page' ); ?>
</div>
```

Example: Using Images

```
<?php posts_nav_link( ' ', '<img src="' . get_bloginfo( 'stylesheet_directory' ) .
'/images/prev.jpg" />', '<img src="' . get_bloginfo( 'stylesheet_directory' ) .
'/images/next.jpg" />' ); ?>
```

- post_class()

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

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

The actual HTML output might resemble something like this for a post in the "dancing" category:

```
<div id="post-4564" class="post post-4564 category-48 category-dancing logged-in">
```

In the WordPress Theme stylesheet, add the appropriate styles, such as:

```
.post {     /* styles for all  posts */     }
.post-4564  {     /* styles for only post ID number 4564 */ }
.category-dancing { /* styles for all posts within the category of dancing */ }
```

<u>Adding More Classes</u>

The classes are restricted to the defaults listed. To add more classes to the post content area, the template tag's parameter can be added. For example, to add a unique class to any template file:

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

The results would be:

```
<div id="post-4564" class="post post-4564 category-48 category-dancing logged-in
class-name">
```

Or you can use an array with classes:

```
<?php $classes = array('class1', 'class2', 'class3', );      ?>
<div id="post-<?php the_ID(); ?>" <?php post_class( $classes ); ?>>
```

<u>Add Classes By Filters</u>

To add a category class to single post pageviews and template files in the post content area for the post class and the entire page in the body HTML class, add the following to the functions.php.

```
// add category nicenames in body and post class
function category_id_class( $classes )
{
       global $post;
       foreach ( ( get_the_category( $post->ID ) ) as $category )
       {      $classes[] = $category->category_nicename;       }
       return $classes;
}
add_filter( 'post_class', 'category_id_class' );
add_filter( 'body_class', 'category_id_class' );
```

<u>Display Posts Outside of the Loop</u>

For displaying posts outside the Loop or in an alternate Loop, the second parameter to the post_class function can be the post ID. Classes will then be determined from that post.

```
<?php post_class( '', $post_id ); ?>
```

Post Thumbnail Tags

- has_post_thumbnail()

  Check if post has an image attached.

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

Parameter: $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.

Example:

```
<?php
// Must be inside a loop.
if ( has_post_thumbnail() )
{       the_post_thumbnail();     }
else
{     echo '<img src="' . get_bloginfo( 'stylesheet_directory' ) .
'/images/thumbnail-default.jpg" />';     }
?>
```

- get_post_thumbnail_id()

  If a featured image (formerly known as post thumbnail) is set - Returns the ID of the featured image attached to the post.

  If no such attachment exists, the function returns an empty string.

  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 );
        }
}
?>
```

- the_post_thumbnail()

  Display the post thumbnail

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

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.

Usage: `the_post_thumbnail( $size, $attr );`

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: ''

Example: Post thumbnail sizes:

```
//Default WordPress
the_post_thumbnail( 'thumbnail' );     // Thumbnail (150 x 150 hard cropped)
the_post_thumbnail( 'medium' );        // Medium resolution (300 x 300 max height 300px)
the_post_thumbnail( 'medium_large' );  // Medium Large (added in WP 4.4) resolution (768 x 0
infinite height)
the_post_thumbnail( 'large' );         // Large resolution (1024 x 1024 max height 1024px)
the_post_thumbnail( 'full' );          // Full resolution (original size uploaded)

//With WooCommerce
the_post_thumbnail( 'shop_thumbnail' ); // Shop thumbnail (180 x 180 hard cropped)
the_post_thumbnail( 'shop_catalog' );   // Shop catalog (300 x 300 hard cropped)
the_post_thumbnail( 'shop_single' );    // Shop single (600 x 600 hard cropped)
```

Example (1): To link Post Thumbnails to the Post Permalink in a specific loop, use the following within your Theme's template files:

```
<?php if ( has_post_thumbnail() ) : ?>
	<a href="<?php the_permalink(); ?>" title="<?php the_title_attribute(); ?>">
	<?php the_post_thumbnail(); ?>
	</a>
<?php endif; ?>
```

Example (2): To link all Post Thumbnails on your website to the Post Permalink, put this in the current Theme's functions.php file:

```
function wpdocs_post_image_html( $html, $post_id, $post_image_id )
{
	$html = '<a href="' . get_permalink( $post_id ) . '" alt="' . esc_attr(
get_the_title( $post_id ) ) . '">' . $html . '</a>';
	return $html;
}
add_filter( 'post_thumbnail_html', 'wpdocs_post_image_html', 10, 3 );
```

- get_the_post_thumbnail()

  Retrieve the post thumbnail.

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

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.

Example:

```
<?php $pages = get_pages( array( 'child_of' => 1 ) ); ?>
<ul>
	<?php foreach ( $pages as $page ) : ?>
	<li>
	<?php echo get_the_post_thumbnail( $page->ID, 'thumbnail' ); ?>
	<h1><?php echo apply_filters( 'the_title', $page->post_title, $page->ID );
?></h1>
	<?php echo apply_filters( 'the_content', $page->post_content ); ?>
	</li>
	<?php endforeach; ?>
</ul>
```

Navigation Menu Tags

- wp_nav_menu()

  Displays a navigation menu.

Usage: `wp_nav_menu( array $args = array() )`

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.

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

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

Targeting a specific menu

```
wp_nav_menu( array( 'menu' => 'Project Nav' ) );
```

Conditional Tags

Source File: wp-includes/query.php.

- is_archive()

  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.

Example:

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

- is_category()

  Is the query for an existing category archive page?

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

Usage: `is_category( mixed $category = '' )`

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

Return: (bool)

Examples:

// When any Category archive page is being displayed.
```
        is_category();
```
// When the archive page for Category 9 is being displayed.
```
        is_category( '9' );
```
// When the archive page for the Category with Name "Stinky Cheeses" is being displayed.
```
        is_category( 'Stinky Cheeses' );
```
// When the archive page for the Category with Category Slug "blue-cheese" is being displayed.
```
        is_category( 'blue-cheese' );
```
// Returns true when the category of posts being displayed is either term_ID 9, or slug "blue-cheese", or name "Stinky Cheeses". Note: the array ability was added in version 2.5.
```
        is_category( array( 9, 'blue-cheese', 'Stinky Cheeses' ) );
```

- is_front_page()

  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(); ?>

Return Values: (boolean) True on success, false on failure.

Example: 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>
```

Usage in a Custom Function

Added to your themes functions file, this code includes the is_front_page() conditional tag after the function name so the content only displays on the front page.

```
add_action( 'loop_start', 'using_front_page_conditional_tag' );
function using_front_page_conditional_tag()
{     if ( is_front_page() )
      {     echo'<h2>Only Displays On The Front Page</h2>'; }
}
```

- is_home()

  Determines if the query is for the blog homepage. 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();    }
```

- is_page()

  Is the query for an existing single page?

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

  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.

Example: supports array too:

```
if( is_page( array( 'about-us', 'contact', 'management' ) )
        //either in about us, or contact, or management page is in view
else
        //none of the page about us, contact or management is in view
```

Example:

```
if ( is_page( 'about' ) || '2' == $post->post_parent )
{       // the page is "About", or the parent of the page is "About"
        $bannerimg = 'about.jpg';      }
elseif ( is_page( 'learning' ) || '56' == $post->post_parent )
{       $bannerimg = 'teaching.jpg';   }
elseif ( is_page( 'admissions' ) || '15' == $post->post_parent )
{       $bannerimg = 'admissions.jpg';       }
else
{       $bannerimg = 'home.jpg';
// just in case we are at an unclassified page, perhaps the home page    }
```

- is_single()

  Is the query for an existing single post?

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

  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:

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

- is_search()

  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(); ?>

Return Values: (boolean) True on success, false on failure.

Example:

```
if ( is_search() )
{      // add external search form (Google, Yahoo, Bing...)   }
```

- is_attachment()

  This Conditional Tag checks if an attachment is being displayed. An attachment is an image or other file uploaded through the post editor's upload utility. Attachments can be displayed on their own 'page' or template. This is a boolean function, meaning it returns either TRUE or FALSE.

Usage: <?php is_attachment(); ?>

Return Values: (boolean) True on success, false on failure.

Example:

```
if ( is_attachment() )
{      // show adv. #1    }
else
{      // show adv. #2    }
```

- is_active_sidebar()

  This Conditional Tag checks if a given sidebar is active (in use). This is a boolean function, meaning it returns either TRUE or FALSE.

  Any sidebar that contains widgets will return TRUE, whereas any sidebar that does not contain any widgets will return FALSE.

Usage: <?php is_active_sidebar($index); ?>

Parameters: $index (mixed) (required) Sidebar name or id. Default: None

Return Values: (boolean) True if the sidebar is in use, otherwise the function returns false

Example: Display different output depending on whether the sidebar is active or not.

```
<?php if ( is_active_sidebar( 'left-sidebar' ) ) {     ?>
<ul id="sidebar">
       <?php dynamic_sidebar( 'left-sidebar' ); ?>
</ul>
<?php } ?>
```

Functions File

One way to change the default behaviors of WordPress is using a file named functions.php. It goes in your Theme's folder.

The functions file behaves like a WordPress Plugin, adding features and functionality to a WordPress site. You can use it to call functions, both PHP and built-in WordPress, and to define your own functions. You can produce the same results by adding code to a WordPress Plugin or through the WordPress Theme functions file.

There are differences between the two.

A WordPress Plugin:

- Requires specific, unique Header text.
- Is stored in wp-content/plugins, usually in a subdirectory.
- Executes only when individually activated, via the Plugins panel.
- Applies to all themes.
- Should have a single purpose, e.g., convert posts to Pages, offer search engine optimization features, or help with backups.

A functions file:

- Requires no unique Header text.
- Is stored with each Theme in the Theme's subdirectory in wp-content/themes.
- Executes only when in the currently activated theme's directory.
- Applies only to that theme. If the Theme is changed, the functionality is lost.
- Can have numerous blocks of code used for many different purposes.

Each theme has its own functions file, but only the functions.php in the active Theme affects how your site publicly displays. If your theme already has a functions file, you can add code to it. If not, you can create a plain-text file named functions.php to add to your theme's directory.

A Child Theme can have its own functions.php. This child functions file can be used to augment or replace the parent theme's functions.

With a functions file you can:

- Use WordPress Hooks, that vast collection of WordPress actions and filters that can alter almost everything WordPress does. For example, with the excerpt_length filter you can change your Post Excerpt length (from default of 55 words).

- Enable WordPress features such as add_theme_support() to turn on Post Thumbnails, Post Formats, and Navigation Menus.

- Define functions you wish to re-use in multiple theme template files.

Beware: if a WordPress Plugin calls the same function, or filter, as you do in your functions file, the results can be unexpected -- even site-disabling.

Search the web for "WordPress functions.php" to find suggestions to enhance the functionality of your WordPress site.