ROCKABLE

HOW TO BE A ROCKSTAR WordPress Designer 2

Rohan Mehta

ROCKABLE

Rockablepress.com
Envato.com

Contents

INTRODUCTION

Introduction

WordPress today is no longer regarded as just a blogging tool, though quite possibly that was the reason it was created. Its flexibility and ease of customization has led WordPress to be one of the most popular Content Management Systems (CMS) used in the web design and development industry.

As a web designer or developer, WordPress is one of the most potent tools available at your disposal. WordPress 3.0 brings with it a multitude of new features — the long-awaited Menu Manager and Custom Post Types which expand WordPress' CMS capabilities. In this book, I am going to teach you to take advantage of WordPress' more advanced features, from shortcodes to custom post types, from creating options pages to custom widgets. Once you're done with this book, you will have a thorough grasp of exactly what is needed to make a WordPress theme usable, flexible and useful.

In this book, I'm not going to be following the conventional methods of writing that most books on WordPress development follow. The approach being taken here assumes that you already know intermediate WordPress coding (for example, you've read the Rockstar WordPress Designer book) and have an understanding of PHP. I've already converted the theme from HTML to WordPress. You can find the complete theme in the folder *WordPress*, entitled *Rockable*. What you are going to be working with is a stripped down version of this theme. You will find a folder in the *WordPress* directory called *BasicFramework*. Take a quick glance at it. You will notice that there are no files in the functions, language, and widgets folders. This is the main difference between the framework and the complete theme. As we go along, you will be filling up these folders with the necessary code — if you have any problems, you can just take a look at the completed theme to figure things out. Note that the *BasicFramework* is **not** a theme. It lacks many critical files to work as a theme, so WordPress will just show you errors if you try to install it as a theme. You have to start with this *BasicFramework* and make it into a complete theme.

So, jump in and get ready to rumble!

Files Included With The Book

Packaged with this book, you will find several folders with files relating to this theme. They are:

- **Photoshop Files:** The PSD files for the design used in this book.
- **HTML Theme:** This is the HTML version of the PSD theme. It consists of the various pages used in the theme — home page, static page, blog page, single blog post, etc.
- **HTML Admin:** This is the HTML design of the administration panel used in the theme. I find it useful first to create the admin theme in HTML, and then logically break it up into smaller parts which are generated dynamically by PHP. Sounds like a mouthful, but it's quite interesting — I will demonstrate how to do this in the book.
- **BasicFramework:** This is the folder with the basic framework for the theme. Rather than repeat everything you probably already know about WordPress, I decided to convert and create the basic structure for the theme. This consists of the regular WordPress theme files, including the `index.php` file, `style.css` stylesheet and various other regular theme files such as `page.php`, `single.php`, etc. You can read more about WordPress template files here and here.
- **Completed Theme:** This is the final completed theme which is ready for use. It is basically the *BasicFramework* plus all the advanced features you will learn about in this book.

These files and templates can be used by you for both personal and commercial purposes, but they may not be resold or redistributed.

In addition, I recommend you use the *BasicFramework* as a base to work upon while proceeding through the book. Note a few things about this though:

- The *BasicFramework* is NOT a working theme
- It regularly uses the **`get_option()`** call. This is a WordPress function that basically fetches an option from the database. Example usage: **`get_option('theme_style')`**. You can view the WordPress Codex article on this function here. We are going to be creating a WordPress theme options page which adds options to the database, so this function is immensely helpful.

A Code Convention

Sometimes the code will be longer than can fit on the width of this book's page. If so, there will be a continuation marker (▸ or ▹) at the end of the line. This means the text on the following line is intended to be typed together with the previous line as one. For example, the following is all typed as one line, with no break:

```
define('ROCKABLE_THEME_DIR',                                    ▹
        get_bloginfo('template_directory'));
```

A hollow arrow indicates that a space is permissible between the character at the end of the starting line and the first character of the second; a solid arrow indicates that there should be no space. The marker itself is not typed in.

1

Setup & Managing Files

Managing Files

While coding a WordPress theme, you should keep something in mind: keep things as easy to edit as possible. This means that you should reuse code that occurs multiple times, separate files that are used many times, and write functions rather than put, in large pieces of code all over the place.

Open up your *BasicFramework* and take a look at the **`search.php`**, **`archive.php`**, and **`taxonomy-portfolio_cat.php`** files. These files are respectively used for search results, archives (category, tag, date archives), and archives for the custom taxonomy we define later. You will notice that in all three files an **`include`** statement is used:

```
<?php include (ROCKABLE_INCLUDES . 'post-excerpt.php'); ?>
```

This file, **`post-excerpt.php`**, displays the excerpt for a post along with some metadata and a thumbnail. Rather than repeating this code in three different files, I decided instead to include one file that is reusable in many places. That way, if I wanted to make a change, I could make it in one place and the change would be reflected in every page template that uses this file. In addition to this file, there are five other files in the **`includes`** folder:

- **`homepage-slider.php`:** This file is for the homepage slider. I have separated this into a file of its own for modularity and so that editing it is easier. It makes sense to separate large code blocks into their own files.
- **`pagination.php`:** This file includes the plugin required for pagination and displays pagination links.

- **`post-meta.php`:** This displays the metadata (date of publishing, comments, author, categories, etc.) for the post.
- **`post-thumbnail.php`:** This displays (optionally) a post-thumbnail for the post.
- **`widget-columns.php`:** This displays the three widget columns in the footer.

The functions.php file

The **`functions.php`** file is a vital part of your theme. From the Codex:

> *A theme can optionally use a functions file, which resides in the theme subdirectory and is named functions.php. This file basically acts like a plugin, and if it is present in the theme you are using, it is automatically loaded during WordPress initialization (both for admin pages and external pages).*

We're going to use the functions file a lot during this book. Open it up, and add in this code:

```
<?php
$curr_theme = get_theme_data(TEMPLATEPATH . '/style.css');
$theme_version = trim($curr_theme['Version']);
if (!$theme_version) $theme_version = "1.0";

//Define constants:
define('ROCKABLE_FUNCTIONS', TEMPLATEPATH . '/functions/');
define('ROCKABLE_WIDGETS', TEMPLATEPATH . '/widgets/');
define('ROCKABLE_INCLUDES', TEMPLATEPATH . '/includes/');
define('ROCKABLE_THEME', 'WordThemer');
define('ROCKABLE_THEME_DIR',                                    ▷
        get_bloginfo('template_directory'));
```

```
define('ROCKABLE_THEME_DOCS',                                    ▷
       ROCKABLE_THEME_DIR.'/functions/docs/docs.pdf');
define('ROCKABLE_THEME_LOGO',                                    ▷
       ROCKABLE_THEME_DIR.'/functions/img/logo.png');
define('ROCKABLE_MAINMENU_NAME', 'general-options');
define('ROCKABLE_THEME_VERSION', $theme_version);
```

In this section of code, we are defining some constants which will be used in our theme. These constants can be used in any file in the theme, and they help in making code more readable and reducing its length. For example, if I wanted to include the file `Rockable/includes/post-excerpt.php`, I would have to use this code:

```
include(TEMPLATEPATH . 'includes/post-excerpt.php');
```

Using my constants, it becomes shorter, and you immediately understand what's happening when you read it:

```
include(ROCKABLE_INCLUDES . 'post-excerpt.php');
```

In addition, there is another advantage: suppose I decided to change my includes folder to modules — if using constants, all I would have to do is change the definition for `ROCKABLE_INCLUDES` — otherwise, I would have to search through every single file, replacing "/includes" with "/modules" — a rather annoying task.

The first three lines use built-in WordPress functions to get the current theme version from `style.css`.

Basic Setup

Next step: put the **`BasicFramework`** folder into your **`wp-content`** folder, and activate the theme. Now, go to your WordPress site. You should see an error:

```
Fatal error: Call to undefined function rockable_titles()
```

This means that we have called a function **`rockable_titles()`** in **`header.php`**, but haven't defined this function. Let's do that now.

First, create an empty file named **`custom-functions.php`** in the **`functions`** folder. This will hold all of our custom functions used in various places in the theme. The function in question dynamically outputs the title for the current page. Here's the code I used:

```
//TITLES
function rockable_titles(){
  $separator=stripslashes(get_option('rockable_separator'));

  if (!$separator)
    $separator="|";
  if (is_front_page())
    bloginfo('name');

  else if (is_single() or is_page() or is_home()){
    bloginfo('name');
    wp_title($separator,true,'');
  }
  else if (is_404()){
    bloginfo('name');
    echo " $separator ";
    _e('404 error - page not found', 'rockable');
  }
  else{
    bloginfo('name');
```

```
        wp_title($separator,true,'');
      }
    }
```

Quite straightforward. First, get a separator. If none is defined, default to "|". Then, output a title:

- The blog name on the home page.
- The blog name, separator, and page title on pages and posts — e.g.: **`Sitename | Page name`**.
- The blog name and "404 error" on 404 error pages.
- A default fallback on other pages.

Note:

You will notice, instead of just using

```
    echo '404 error - page not found';
```

I use

```
   _e('404 error - page not found', 'rockable');
```

The reason for this is theme localization (easy translation). We'll come to that later, but for now know this: wherever you have 'hard coded' text, use **`_e`** instead of **`echo`**, and use **`__`** (double underscore) instead of a direct assignation. Examples:

- **`_e('some text', 'rockable')`** instead of **`echo 'some text'`**
- **`$str = __('some text', 'rockable')`** instead of **`$str = 'some text'`**

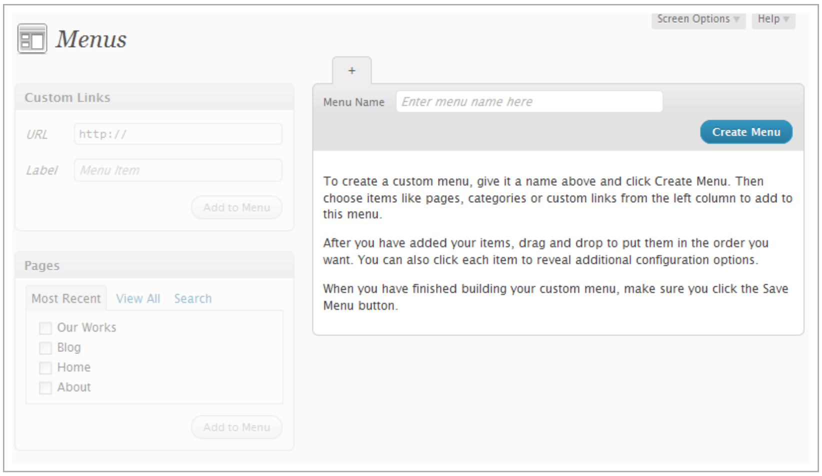
Now that our function is defined, we need to include the file so that this function is available to WordPress. Add this to **`functions.php`**:

```
  //Load all-purpose functions:
    require_once (ROCKABLE_FUNCTIONS .                          ▷
                   'custom-functions.php');
```

Next, we'll get some basic stuff set up — let's start with menus. WordPress 3.0 created a new type of menu management — you can add any links you want, and are no longer restricted to page-only or category-only menus as before. You can see the new menu interface by going to *Appearance* *Menus* in your WordPress Dashboard. This will only show up if you have support for menus in your theme. To add support, include this line of code in **`functions.php`**:

```
    add_theme_support( 'menus' );
```

Now you will be able to see the menu interface:



To create a custom menu, give it a name above and click Create Menu. Then choose items like pages, categories or custom links from the left column to add to this menu.

After you have added your items, drag and drop to put them in the order you want. You can also click each item to reveal additional configuration options.

When you have finished building your custom menu, make sure you click the Save Menu button.

Fig. 1-1. Menus interface.

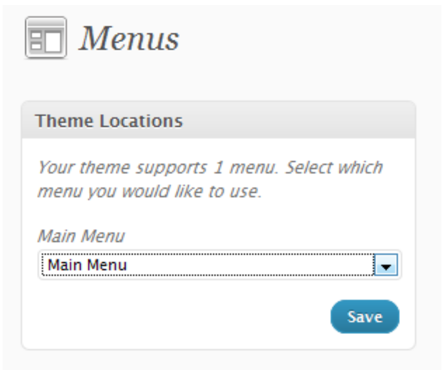
Next, you have to "register" a menu area. This means that when a user creates a menu, they will see a place in the theme to assign the menu. For example, I could assign a location for a header menu — when a user creates a menu and adds it to the header menu location, the menu will show up there. Create a new file named **`register-wp3.php`** in the **`functions`** folder and add this code to it:

```
  if (function_exists('register_nav_menu'))
    register_nav_menu('main_menu', __( 'Main Menu',           ▹
                                  'rockable' ));
  endif;
```

This registers a new nav menu area with the ID "main_menu" and the (translatable) name "Main Menu". Include the new file from **`functions.php`** with this code:

```
    require_once (ROCKABLE_FUNCTIONS . 'register-wp3.php');
```

Now, create a new menu. You will see this box when it's created:



Your theme supports 1 menu. Select which menu you would like to use.

Main Menu: Main Menu

Fig. 1-2. Registered menu locations.

If you look at the HTML version of the theme, you will notice that there is a small description under the menu title. To do this, we need some custom code, as it is not immediately supported in WordPress. In your **`custom-functions.php`** file, create a new function **`rockable_menu()`** and add in this code:

```
function rockable_menu(){
  if (function_exists('wp_nav_menu') &&                    ▹
     has_nav_menu('main_menu')):
    wp_nav_menu(
      array(
        'theme_location' => 'main_menu',
        'container' => '',
        'menu_class' => 'sf-menu',
        'depth' => 2,
        'walker' => new rockable_menu_walker())
    );

  else
    echo '<ul class="sf-menu">';
      wp_list_pages('depth=1&title_li=');
    echo '</ul>';

  endif;
}
```

We're using a few new functions here:

- First, we check if the function **`wp_nav_menu()`** exists. If it does not, the version is less than WP 3.0, so we can't use custom nav menus.
- Then, we use **`has_nav_menu()`** — This function checks if a menu has been assigned to a theme location.
- If either of these are false, then we can't display a nav menu. So, we default to the **`wp_list_pages()`** function.

The wp_nav_menu() function

The `wp_nav_menu()` function displays a custom nav menu. Its parameters are:

- **menu:** The ID, slug or name of a created menu
- **container:** The wrapping element, e.g. `<div>`
- **container_class:** The class applied to the container
- **container_id:** The ID applied to the container
- **menu_class:** The class assigned to the menu `<ul>`
- **menu_id:** The ID applied to the menu `<ul>`
- **echo:** Whether to actually display the menu, or just return it; defaults to `true`
- **fallback_cb:** The fallback function if the menu cannot be displayed, e.g. if no menu is found
- **before, after:** The text to display before and after the link text
- **link_before, link_ after:** The text to display before and after the link
- **depth:** How many levels of sub-menus to display. Default is `0` (unlimited levels)
- **walker:** a "walker" object to use for the menu (more on this below)
- **theme_location:** The ID of a registered nav menu (`main_menu`) in our case

For our menu, I have added arguments as per the HTML — no container element, a class of `sf-menu` assigned to the `<ul>` and a depth of 2 (parents and one sublevel). The interesting part here is the argument for a custom Walker. In 99% of cases, this isn't required — we need it only to display the description. On its own, WordPress uses a *Walker* class to display the navigation. We can simply *extend* this Walker class to output things our way. Read more about classes here: http://php.net/manual/en/language.oop5.php.

Rather than extend our Walker without any base, I decided to just modify the regular Walker code. If you see here, in the core code of WordPress: http://core.trac.wordpress.org/browser/trunk/wp-includes/nav-menu-template.php, you can see the Walker code at the top of the file. We only need to deal with the **`start_el()`** function. Add this code to **`custom-functions.php`**:

```
class rockable_menu_walker extends Walker_Nav_Menu
{
  function start_el(&$output, $item, $depth, $args){
```

The keyword **`extends`** tells PHP we are writing a class which just builds upon the **`Walker_Nav_Menu`** class. Basically, we are just overwriting the **`start_el()`** function.

Now, just copy the entire code of the **`start_el()`** function from the source code in that link and paste it into your **`start_el()`** function. Add this line below the **`$attributes`** assignment:

```
$description  =! empty( $item->description ) ? '<span>'. ▶
esc_attr( $item->description ).'</span>' : '';
```

That fetches the description of the menu item and assigns it to **`$description`**.

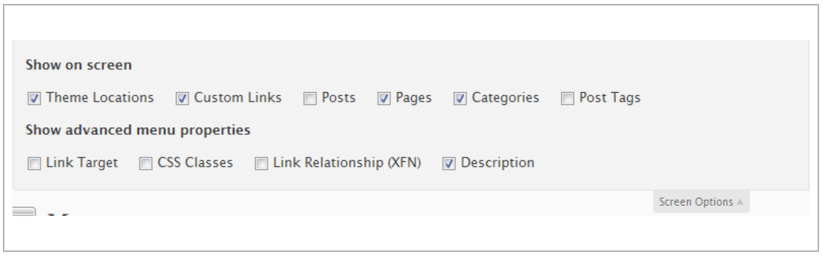
Now, we need to display this description somewhere. Look for this line in your code:

```
    $item_output .= $args->link_before . apply_filters(             ▷
        'the_title', $item->title, $item->ID ) .                    ▷
        $args->link_after;
```

Change that to:

```
    $item_output .= $args->link_before . apply_filters(             ▷
        'the_title', $item->title, $item->ID );
    $item_output .= $description . $args->link_after;
```

The difference: we've added in our **`$description`** variable to show in the output. That's it. With those two edits we're done with the backend code. You need to do one more thing to add descriptions though. In the nav menu interface, click on *Screen Options* and check the *Description* checkbox. This will allow you to enter descriptions.


Show on screen

[✓] Theme Locations [✓] Custom Links [ ] Posts [✓] Pages [✓] Categories [ ] Post Tags

Show advanced menu properties

[ ] Link Target [ ] CSS Classes [ ] Link Relationship (XFN) [✓] Description

Fig. 1-3. Menus interface — screen options.

And that's it for the nav menus. Before we go further, we should quickly write up a few generic functions which will be used throughout the theme and that can be used in other themes as well.

JavaScript Files

You might notice, there are no JavaScript files included in **`header.php`**. Instead, we are going to use a WordPress function called `wp_enqueue_script()`. This function **safely** adds JS files to the WordPress page. It prevents conflicts: for example, if you added jQuery to the header and a plugin also wanted to add jQuery, there would be two jQuery files included in the page. If you used **`wp_enqueue_script()`** then it would only be added once — WordPress would take care of that.

So, we are going to enqueue all the JS files used in the theme. Add this to **`functions.php`**:

```
if ( !is_admin() ) {
  wp_enqueue_script('jquery');
  wp_enqueue_script('superfish', ROCKABLE_THEME_DIR.         ▶
    '/assets/js/superfish.js');
  wp_enqueue_script('jcarousel', ROCKABLE_THEME_DIR.         ▶
    '/assets/js/jquery.jcarousel.min.js');
  wp_enqueue_script('jqueryui_custom', ROCKABLE_THEME_DIR.   ▶
    '/assets/js/jquery-ui-1.8.1.custom.min.js');
  wp_enqueue_script('cufon', ROCKABLE_THEME_DIR.             ▶
    '/assets/js/cufon.js');
  wp_enqueue_script('font', ROCKABLE_THEME_DIR.              ▶
    '/assets/js/myriad_pro.font.js');
  wp_enqueue_script('rockable', ROCKABLE_THEME_DIR.          ▶
    '/assets/js/rockable.js');
  wp_enqueue_script('rockable_contact',  ▷
    ROCKABLE_THEME_DIR.'/assets/js/rockable_contact.js');
  wp_enqueue_script( 'comment-reply' );
}
```

The usage of the function is:

```
wp_enqueue_script($handle, $src, $deps, $ver, $in_footer);
```

`handle` is the unique name you give to the script, **`src`** is the source URL, **`deps`** is the dependency (e.g. if it is reliant on jQuery), **`ver`** is the version, used to prevent caching when versions are changed and **in_footer** specifies whether to add the footer to the <**head**> or to the footer.

Image Functions

Images are a pretty important part of premium WordPress themes. They are used in most themes as post thumbnails, in galleries, etc. So rather than perform the same image-fetching techniques many times, we should automate the process. Here are two functions (to be added to **custom-functions.php**) which will automate this for us.

```
function rockable_post_image(){
  global $post;
  $image = '';

  //Get the image from the post meta box
  $image = get_post_meta($post->ID, 'rockable_post_image', ▷
    true);
  if ($image) return $image;

  //If the above doesn't exist, get the post thumbnail
  $image_id = get_post_thumbnail_id($post->ID);
  $image = wp_get_attachment_image_src($image_id,          ▷
    'rockable_thumb');
  $image = $image[0];
  if ($image) return $image;

  //If there is still no image, get the first image from   ▷
    the post
  return rockable_get_first_image();
}
function rockable_get_first_image(){
  global $post, $posts;
  $first_img = '';
```

```
    ob_start();
    ob_end_clean();
    $output = preg_match_all('/<img.+src=[\'"]([^\'"]+)    ▸
      [\'"].*>/i', $post->post_content, $matches);

    $first_img="";

    if (isset($matches[1][0]))
      $first_img = $matches[1][0];

    return $first_img;
  }
```

Explanations: **`rockable_post_image()`** fetches the image for the current post. It follows an order: first, it looks for a custom field with the key "**`rockable_post_image`**". If not found, it looks for the post thumbnail (a built in WordPress feature). If there is still no image found, it scans the content of the post to search for the first image, using regular expressions (the function **`rockable_get_first_image()`**). Therefore, if you wanted to display the post image, you could use code like this:

```
  <img src="<?php echo rockable_post_image(); ?>" alt="" />
```

Simple, don't you agree?

Note: *You might notice that I am prefixing every function with* **`rockable_`** *— e.g.* **`rockable_post_image()`***. This is so that there are no conflicts with other plugins or scripts. If I used a generic function name like* **`get_image()`** *there could be conflicts with the core WordPress code or with plugins. Always try to use a unique prefix in your code.*

If you take a look at the **`php`** folder of the theme, you'll notice a file **`timthumb.php`** — this is an automatic resizing script — read more about it here: http://www.darrenhoyt.com/2008/04/02/timthumb-php-script-released/

TimThumb is very useful in WordPress themes. Since it dynamically resizes images, by uploading just one image you can have different image sizes throughout the theme without uploading many differently cropped images. It's used like this:

```
<img src="/timthumb.php?src=image.jpg&w=100&h=100" alt=""/>
```

That will give you a resized image of **`image.jpg`**, with a width (**`w`**) and height (**`h`**) of 100 pixels.

We're going to be using TimThumb in this theme. Now, rather than writing the same code over and over, I've written a function in **`custom-functions.php`** to generate the TimThumb code for us:

```
function rockable_build_image($img='', $w=false, $h=false, ▷
                                        $zc=1 ){
  if ($h)
    $h = "&amp;h=$h";
  else
    $h = '';

  if ($w)
    $w = "&amp;w=$w";
  else
    $w = '';

  $image_url = ROCKABLE_THEME_DIR .                          ▷
    "/php/timthumb.php?src=" . $img . $h . $w;
  return $image_url;
}
```

The parameters for this function are the image source URL, width, height, and the zoom-crop. One thing to note: if you specify only a height or width, TimThumb will maintain the aspect ratio. Therefore, you are able to specify only height or width here, if wanted. Sample usage:

- **`rockable_build_image('image.jpg', 100)`** will give you the URL to an image with a width of 100, and a corresponding height. So if your original image was 200 × 70 pixels, the new one will be 100 × 35.
- Similarly, **`rockable_build_image('image.jpg', false, 100)`** will give you the URL to an image with a height of 100, and a corresponding width. So if your original image was 120 × 200 pixels, the new one will be 60 × 100.

Finally, there is one more generic function to be written. WordPress has a built in function called **`the_excerpt()`** which displays a 55 word excerpt of the page. But what if we want a different number of words in the excerpt? Here's the code:

```
function rockable_excerpt($len=20, $trim="&hellip;"){
  $limit = $len+1;
  $excerpt = explode(' ', get_the_excerpt(), $limit);
  $num_words = count($excerpt);
  if ($num_words >= $len){
    $last_item = array_pop($excerpt);
  }
  else{
    $trim = "";
  }
  $excerpt = implode(" ",$excerpt) . "$trim";
  echo $excerpt;
}
```

Whoa, you say, what's that? Let me break it down for you. The parameters are the excerpt length (default 20) and the trim character displayed after the excerpt (defaults to the HTML entity **`&hellip;`** which is an ellipsis: `...`). What it does is this:

- First, it gets the excerpt using `get_the_excerpt()`.
- Then, it explodes (breaks up) the excerpt into words by separating them by blanks. For example, "Rockable is awesome" would be broken into `array('Rockable', 'is', 'awesome')`. The exploded array is limited to the length of the new excerpt, given by `$len`.
- Then, if the number of words in `$excerpt` is more than the length, it removes the rest of the words.
- Finally, it joins up the array `$excerpt` using the `implode()` function (opposite of `explode()`) and appends the `$trim` to the end. It prints this new excerpt.

Sample usage:

```
    rockable_excerpt(5); //prints a 5 word excerpt
```

With this, our basic setup is over and Chapter 1 comes to an end. In the next chapter, you will learn how to create and style theme options pages.

About The Author

Rohan Mehta is a Wordpress and jQuery artist, with significant experience in Java and web technologies. He is passionate about racquet sports, reading, and of course, code. In his opinion, code is a form of art, and he expresses his creativity in Java, PHP and JavaScript. Rohan is currently in pursuit of a Bachelor's degree in Computer Science, and resides in the United States and India; he can be reached at his email, rohan@getrohan.com.

As a web designer or developer, WordPress is one of the most potent tools at your disposal. In Rockstar WordPress Designer 2, Rohan Mehta picks up where the original Rockstar WordPress Designer left off and teaches you to take advantage of WordPress 3.0's more advanced features. Included in the book:

- Using Metaboxes
- Advanced image functions
- Custom widgets and option pages
- Using shortcodes and custom post types
- Internationalization and translation issues
- Crafting threaded comments, breadcrumb
- navigation, and more!

Rockstar WordPress Designer 2 also includes EVERY project file referenced in the book, helping you get off to a running start. Grab a copy of Rockstar WordPress Designer 2 and improve your WordPress mastery.

ROCKABLE