Hello everyone. Today we will be learning about two main WordPress features that may come in handy while making your plugins how to create option pages, and the function**register_activation_hook**.

So, I am basing this video off of Mark Jaquith's tutorial on creating a simple WordPress plugin. Just for a quick recap on his video: he created a simple plugin that would replace the word *foo* with the word *bar* in post titles and post content.

Today, we are going to bring that plugin into the world of customization. We are going to permit the user to choose what word should be replaced, and what it should be replaced with.

All consumers love to customize, because they all have different needs. How do you accommodate for these needs in a WordPress plugin, you ask? Create some option pages!

Option pages are custom pages that you can create in the WordPress dashboard. They can usually be found under the Options menu, hence the name. It is possible to create custom pages under other menus, but I won't be showing you that in this tutorial. If you want to put your page under a different menu, try following this loosely with help from the WordPress Codex.

A quick note in this tutorial, options and functions will use a`replace` prefix, to avoid conflict with other options and functions. I highly recommend you use a unique prefix as I do so that your plugin has the best results, and lower chance for failure.

Let's get started! Here I have the code for the plugin that Mark Jaquith created in his tutorial. We're going to be adding a few more functions to this.

Here is the part where we learn about **register_activation_hook**. This function tells WordPress that when the plugin is activated, it should execute the function that we tell **register_activation_hook** about. This is the syntax of **register_activation_hook**:

```
register_activation_hook( $file, $function);
```

There are two parameters here: `$file` and `$function`. Let's move backwards, starting with the latter. The second parameter is a function that will be run when the plugin is activated. The first parameter is the file that contains said function. So, let's tell WordPress to run a function when it installs our plugin:

```
register_activation_hook( basename( __FILE__ ), "replace_add_options" );
```

This is telling WordPress to run the function `replace_add_options` inside the current file. Usually you'll want to use what I did for the first parameter; it means that the function is inside the current file. Only use something else if, well, the function *isn't* inside the current file.

Now we need to add the function `replace_add_options` into our plugin:

```
function replace_add_options () {
     add_option( "replace_from" );
     add_option( "replace_with" );
}
```

You may have been wondering the purpose of this first part. We are preparing options so that they can be easily set later. You may be able to tell from `replace_add_options` that we are simply adding options into WordPress. What this actually does is add two rows to the options table of the database. By doing this, we won't get an error later when we try to modify these that the options don't exist.

Now we need to tell WordPress to run a function when it is loading the menu of the WordPress dashboard. In this

function that we'll be creating, we will create a link to our page in the Options menu.

First, let's add a hook so that WordPress knows we're doing something:

```
add_action( "admin_menu", "replace_add_pages" );
```

Now we need to create the `replace_add_pages` function. All this function will do is add a small link to the end of the Options menu:

```
function replace_add_pages () {
     add_options_page( "Replace", "Replace", 8, __FILE__, "replace_options");
}
```

Don't worry if all those parameters overwhelm you! I'll explain them each.

First, the syntax of the function `add_options_page`:

```
add_options_page( $page_title, $menu_title, $access_level, $file, [$function]);
```

And here is an explanation for the parameters:

- `$page_title`
  The title of the page. This will be displayed in the title-bar of a web browser.
- `$menu_title`
  The title of the page. This will be displayed in the Options menu.
- `$access_level`
  The minimum user level or capability required to display and use the page.
- `$file`
  A **unique** identifier for the menu page. Use a filename, or else.
- `$function`
  The function that displays the page content. If this parameter is not declared, WordPress assumes that the entire contents of `$file` will display the page content.

So, we've added a link to our yet-to-be-made page. Now we must create that page!

One of the things that makes WordPress option-pages so easy is that you don't need to manually tell WordPress to update the options when the user sends new ones. You just tell WordPress in a form which we'll create that two options need to be updated, and give WordPress the new option values.

Let's create our function:

```
function replace_options () {
     ?><div class="wrap">
          <form method="post" action="options.php">
               <?php wp_nonce_field( "update-options" ); ?>
               Replace from: <input type="text" name="replace_from" value="<?php
                    get_option( 'replace_from' ); ?>" /><br/>
               Replace with: <input type="text" name="replace_with" value="<?php
                    get_option( 'replace_with' ); ?>" /><br/>
               <input type="hidden" name="action" value="update" />
               <input type="hidden" name="page_options"
                    value="replace_from,replace_with" />
               <p class="submit">
                    <input type="submit" name="Submit" value="<?php _e('Update
                         Options »'); ?>" />
```

```
            </p>
        </form>
    </div><?php
}
```

Now that is one overwhelming piece of code, don't you think? But do not worry. I'll walk you through it.

First, we declare our function and create a **div** with a class of *wrap*. This is an element that WordPress just uses for styling. This is only there so that our page may look normal.

Next, we create a form. The *method* of option forms will always be *post*. We also declare that it should return to **options.php** when the form is submitted.

The next line inserts two hidden fields—one checks that the user is able to update these options (has the right capabilities), and the other changes the *action* of the form to a special value. This is almost absolutely necessary for a good plugin to use this line.

Now we output some visible text on the page. The **input** elements' name is the name of the option that they will be updating. We grab the current value of these options and make it visible.

Next, we tell WordPress through a hidden field that it should update options once the form is submitted. The next tells which options shall be updated (notice how they correspond to the text fields!).

Lastly, we make a Submit button inside a styled paragraph. I recommend keeping *_e* inside the *value* field. This will translate the string automatically if the user has specified a different language.

That's actually all there is to it! WordPress will automatically update the options, because we told it to.

Now, this is all fine and dandy, but these options don't affect anything yet. We need to modify Mark's original function. Change it from the original:

```
function my_function ($text) {
    $text = str_replace("foo", "bar");
    return $text;
}
```

To this:

```
function my_function ($text) {
    $text = str_replace( get_option("replace_from"),
        get_option("replace_with") );
    return $text;
}
```

As you can see, instead of using two fixed values, the new function gets the user-defined values.

Well, we're done here! To end with a cheesy statement, a good plugin is a happy plugin!