WordPress Plugin Development

WordPress for Developers

Requirements

Tools/Software

1. Web server solution stack (preferably XAMPP or Laragon)
2. A clean WordPress installation
3. Text Editor or IDE (Notepad++, Sublime Text, VS Code, etc.)

Technical

Learners must have at least basic knowledge about the following:

- HTML
- CSS
- JavaScript
- PHP
- An overview of how WordPress works

Before we start

After every end of the topics. You will be given time for question/concerns. Raise your hands if you have questions/concerns about the topic/s.

Course Outline

1. Introduction
2. Plugin Basics
3. Shortcodes
4. Custom Post Types
5. Plugin Security
6. Metadata
7. Hooks
8. Javascript, JQuery & AJAX
9. Administration Menus
10. Settings
11. Cron

About Me

Name: Darell Duma

- WordPress Developer
- Specializes in Platform Development using WordPress & Laravel

Youtube (just started/no contents yet):

- Web Shifu
- WordPress Docs Explained

Facebook Page: Darell Duma::Web Developer

Introduction

PHP

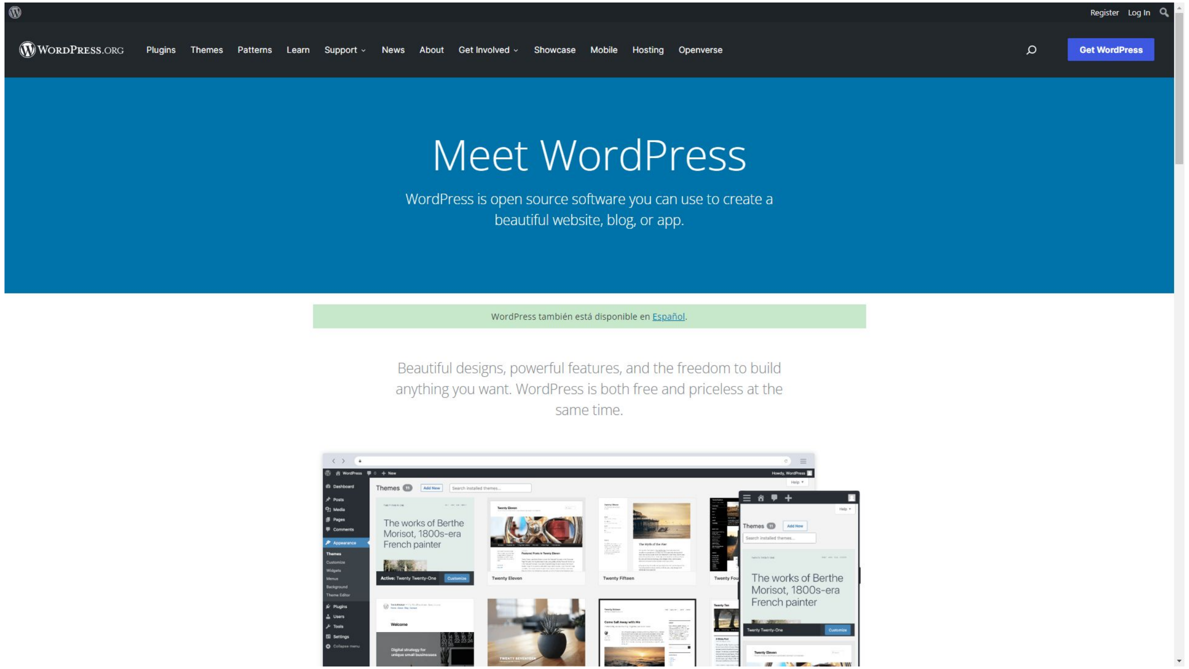
A **popular general-purpose scripting language** that is especially suited to web development.
Fast, flexible and pragmatic, PHP powers everything from your blog to the most popular websites in the world.

php
What's new in 8.1
Download
8.1.8 · Changelog · Upgrading
8.0.21 · Changelog · Upgrading
7.4.30 · Changelog · Upgrading

php.net

WordPress

- a free and open-source content management system written in PHP and paired with a MySQL or MariaDB database (Wikipedia)



Meet WordPress

WordPress is open source software you can use to create a beautiful website, blog, or app.

WordPress también está disponible en Español.

Beautiful designs, powerful features, and the freedom to build anything you want. WordPress is both free and priceless at the same time.

Themes
Add New
The works of Berthe Morisot, 1800s-era French painter
Active: Twenty Twenty-One
Customize
Twenty Eleven
Twenty Fifteen
Twenty Twenty-One

WordPress Plugins

- allow you to greatly extend the functionality of WordPress without touching WordPress core itself.
- Examples:

Wordfence
WORDPRESS SECURITY

Woo

Plugin Basics

How to Create Your Plugin Files

1. Go to wp-content/plugins
2. Create Folder
3. Create your files

Header Requirements (contd.)

1. **Plugin Name:** (*required*) The name of your plugin, which will be displayed in the Plugins list in the WordPress Admin.
2. **Plugin URI:** The home page of the plugin, which should be a unique URL, preferably on your own website. This *must be unique* to your plugin. You cannot use a WordPress.org URL here.
3. **Description:** A short description of the plugin, as displayed in the Plugins section in the WordPress Admin. Keep this description to fewer than 140 characters.
4. **Version:** The current version number of the plugin, such as 1.0 or 1.0.3.
5. **Requires at least:** The lowest WordPress version that the plugin will work on.
6. **Requires PHP:** The minimum required PHP version.
7. **Author:** The name of the plugin author. Multiple authors may be listed using commas.
8. **Author URI:** The author's website or profile on another website, such as WordPress.org.
9. **License:** The short name (slug) of the plugin's license (e.g. GPLv2). More information about licensing can be found in the WordPress.org guidelines.
10. **License URI:** A link to the full text of the license (e.g. https://www.gnu.org/licenses/gpl-2.0.html).
11. **Text Domain:** The gettext text domain of the plugin. More information can be found in the Text Domain section of the How to Internationalize your Plugin page.
12. **Domain Path:** The domain path lets WordPress know where to find the translations. More information can be found in the Domain Path section of the How to Internationalize your Plugin page.
13. **Network:** Whether the plugin can only be activated network-wide. Can only be set to *true*, and should be left out when not needed.
14. **Update URI:** Allows third-party plugins to avoid accidentally being overwritten with an update of a plugin of a similar name from the WordPress.org Plugin Directory. For more info read related dev note.

Including a Software License

Most WordPress plugins are released under the GPL, which is the same license that WordPress itself uses. However, there are other compatible options available. It is always best to clearly indicate the license your plugin uses.

In the Header Requirements section, we briefly mentioned how you can indicate your plugin's license within the plugin header comment. Another common, and encouraged, practice is to place a license block comment near the top of your main plugin file (the same one that has the plugin header comment).

Software License Block Comment Example

```
/*
{Plugin Name} is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 2 of the License, or
any later version.

{Plugin Name} is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with {Plugin Name}. If not, see {URI to Plugin License}.
*/
```

Activation/Deactivation Hooks

- On *activation*, plugins can run a routine to add rewrite rules, add custom database tables, or set default option values.
- On *deactivation*, plugins can run a routine to remove temporary data such as cache and temp files and directories.

```
register_activation_hook( __FILE__, 'pluginprefix_function_to_run' );
```

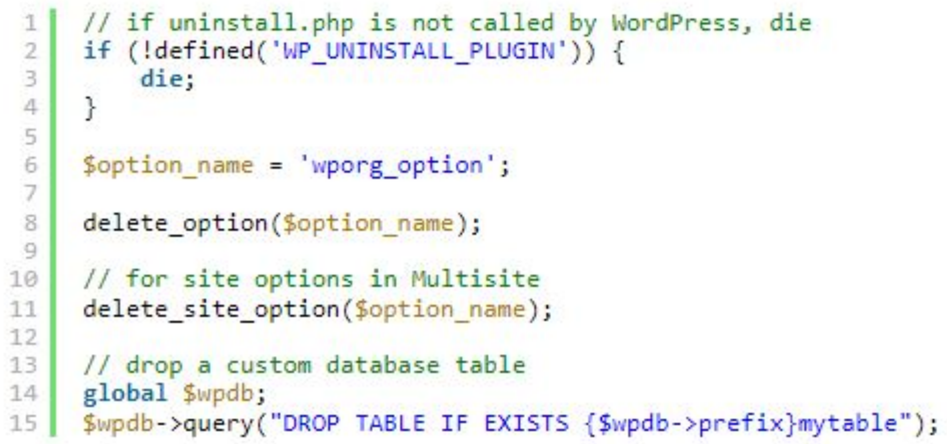
```
register_deactivation_hook( __FILE__, 'pluginprefix_function_to_run' );
```

Uninstall Methods

Method 1: register_uninstall_hook

```
register_uninstall_hook(__FILE__, 'pluginprefix_function_to_run');
```

Method 2: uninstall.php



```php
// if uninstall.php is not called by WordPress, die
if (!defined('WP_UNINSTALL_PLUGIN')) {
    die;
}

$option_name = 'wporg_option';

delete_option($option_name);

// for site options in Multisite
delete_site_option($option_name);

// drop a custom database table
global $wpdb;
$wpdb->query("DROP TABLE IF EXISTS {$wpdb->prefix}mytable");
```

Best Practices

- Avoid naming collisions
  - Procedural Coding
    - Prefix Everything
    - Check for Existing Implementations
  - Object Oriented Method
    - File Organization
    - Plugin Architecture
    - Conditional Loading

Determining Plugin and Content Directories

When coding WordPress plugins you often need to reference various files and folders throughout the WordPress installation and within your plugin or theme.

WordPress provides several functions for easily determining where a given file or directory lives. Always use these functions in your plugins instead of hard-coding references to the wp-content directory or using the WordPress internal constants.

Determining Plugin and Content Directories

Plugins

```
plugins_url( 'images' )  //http://www.example.com/wp-content/plugins/images

plugin_dir_url( __FILE__ )
//http://example.com/wp-content/plugins/my-plugin/assets/foo-styles.css

plugin_dir_path( __FILE__ ) ///home/user/var/www/wordpress/wp-content/plugins/

plugin_basename( __FILE__ ) //wpdocs-plugin/wpdocs-plugin.php
```

Determining Plugin and Content Directories

Home

`home_url()` https://example.com

`get_home_path()` /var/www/htdocs/

Determining Plugin and Content Directories

WordPress

`admin_url()`	https://example.com/wp-admin
`site_url()`	https://example.com/
`content_url()`	https://example.com/wp-content
`includes_url()`	https://example.com/wp-includes
`wp_upload_dir()`	https://example.com/wp-content/uploads

Shortcodes

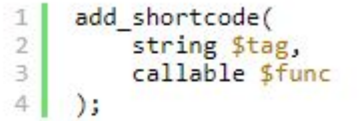
Shortcodes

As a security precaution, running PHP inside WordPress content is forbidden; to allow dynamic interactions with the content, Shortcodes were presented in WordPress version 2.5.

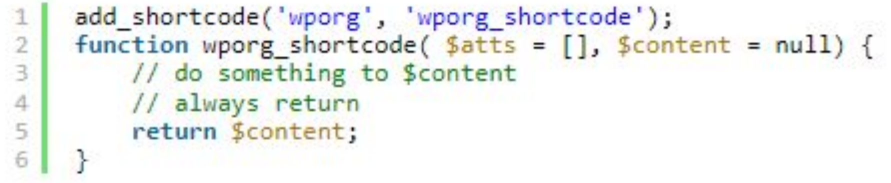
Shortcodes are macros that can be used to perform dynamic interactions with the content. i.e creating a gallery from images attached to the post or rendering a video.

Shortcodes

Basic Shortcode



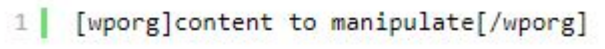
```
add_shortcode(
    string $tag,
    callable $func
);
```



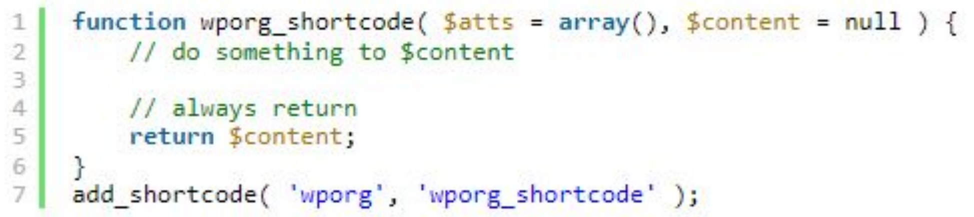
```
add_shortcode('wporg', 'wporg_shortcode');
function wporg_shortcode( $atts = [], $content = null) {
    // do something to $content
    // always return
    return $content;
}
```

Shortcodes

Enclosing Shortcodes



```
[wporg]content to manipulate[/wporg]
```

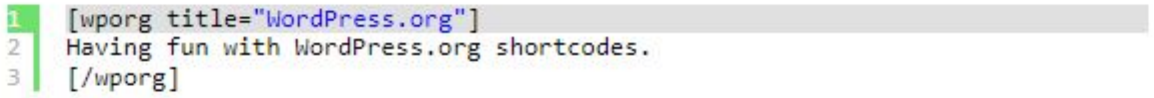


```
function wporg_shortcode( $atts = array(), $content = null ) {
    // do something to $content

    // always return
    return $content;
}
add_shortcode( 'wporg', 'wporg_shortcode' );
```

Shortcodes

Shortcodes with Parameters



```
[wporg title="WordPress.org"]
Having fun with WordPress.org shortcodes.
[/wporg]
```

Hooks

Hooks

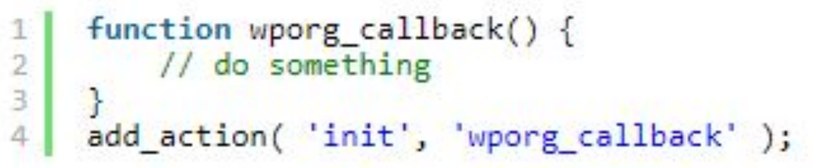
Hooks are a way for one piece of code to interact/modify another piece of code at specific, pre-defined spots. They make up the foundation for how plugins and themes interact with WordPress Core, but they're also used extensively by Core itself.

There are two types of hooks: Actions and Filters. To use either, you need to write a custom function known as a `Callback`, and then register it with a WordPress hook for a specific action or filter.

Actions allow you to add data or change how WordPress operates. Actions will run at a specific point in the execution of WordPress Core, plugins, and themes. Callback functions for Actions can perform some kind of a task, like echoing output to the user or inserting something into the database. Callback functions for an Action do not return anything back to the calling Action hook.

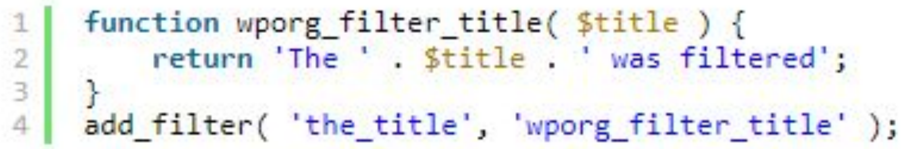
Filters give you the ability to change data during the execution of WordPress Core, plugins, and themes. Callback functions for Filters will accept a variable, modify it, and return it. They are meant to work in an isolated manner, and should never have side effects such as affecting global variables and output. Filters expect to have something returned back to them.

Action Hooks



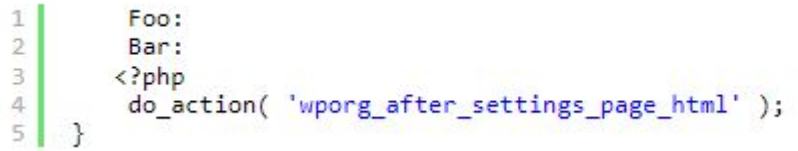
```php
function wporg_callback() {
    // do something
}
add_action( 'init', 'wporg_callback' );
```

Filter Hooks

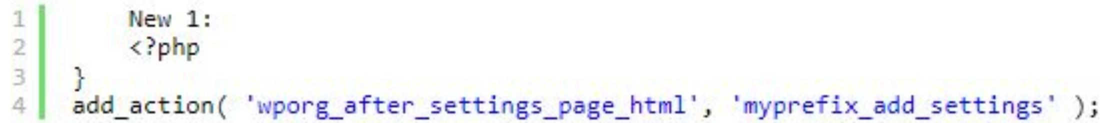


```php
function wporg_filter_title( $title ) {
    return 'The ' . $title . ' was filtered';
}
add_filter( 'the_title', 'wporg_filter_title' );
```

Custom Hooks



```
      Foo:
      Bar:
     <?php
      do_action( 'wporg_after_settings_page_html' );
}
```



```
      New 1:
      <?php
}
add_action( 'wporg_after_settings_page_html', 'myprefix_add_settings' );
```

WordPress Hooks Reference

https://developer.wordpress.org/reference/hooks/

Custom Post Types

Custom Post Types

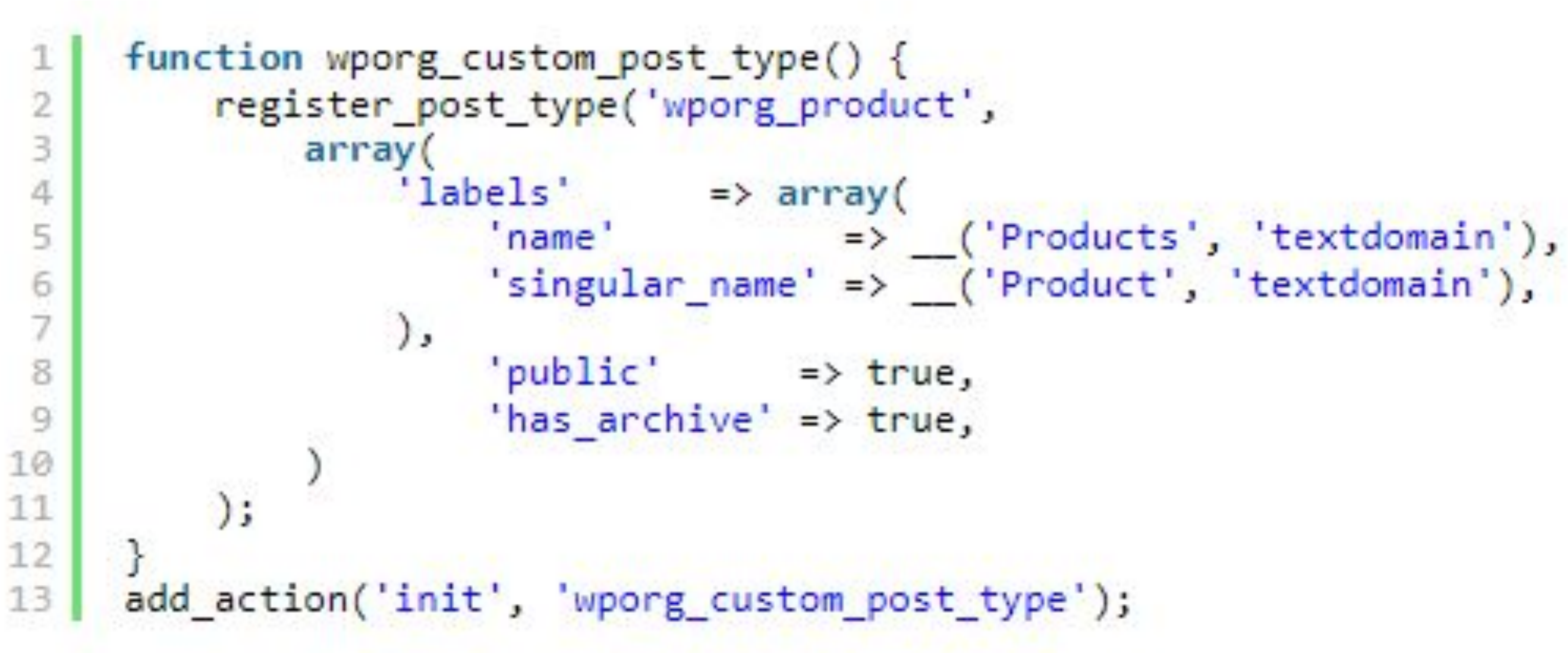
WordPress comes with five default post types: post, page, attachment, revision, and menu.

While developing your plugin, you may need to create your own specific content type: for example, products for an e-commerce website, assignments for an e-learning website, or movies for a review website.

Using Custom Post Types, you can register your own post type. Once a custom post type is registered, it gets a new top-level administrative screen that can be used to manage and create posts of that type.

Custom Post Types

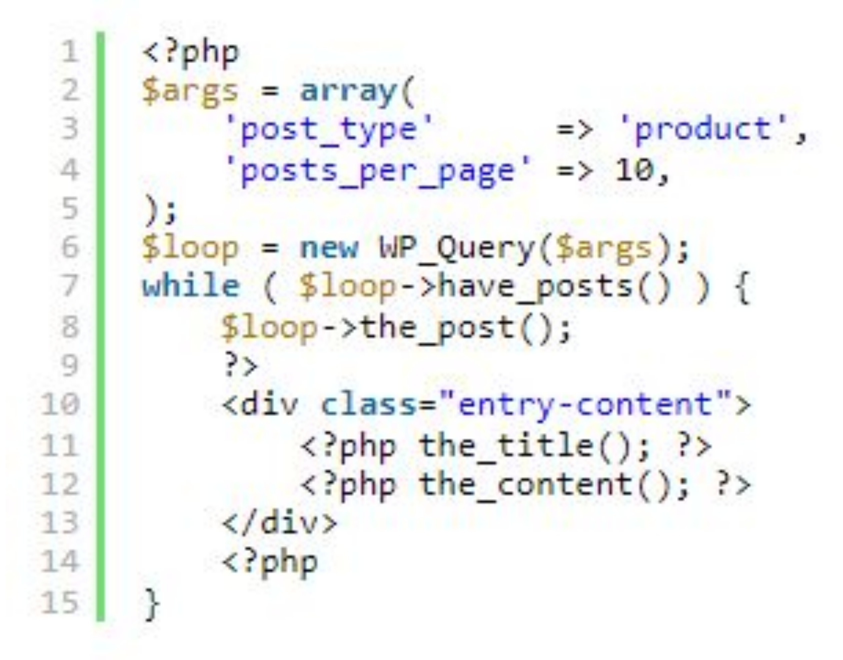
Registering Custom Post Types



```php
function wporg_custom_post_type() {
    register_post_type('wporg_product',
        array(
            'labels'      => array(
                'name'          => __('Products', 'textdomain'),
                'singular_name' => __('Product', 'textdomain'),
            ),
                'public'      => true,
                'has_archive' => true,
        )
    );
}
add_action('init', 'wporg_custom_post_type');
```

Working with Custom Post Types

Querying by Post Types



```php
<?php
$args = array(
     'post_type'      => 'product',
     'posts_per_page' => 10,
);
$loop = new WP_Query($args);
while ( $loop->have_posts() ) {
    $loop->the_post();
    ?>
    <div class="entry-content">
        <?php the_title(); ?>
        <?php the_content(); ?>
    </div>
    <?php
}
```

Metadata

Managing Metadata

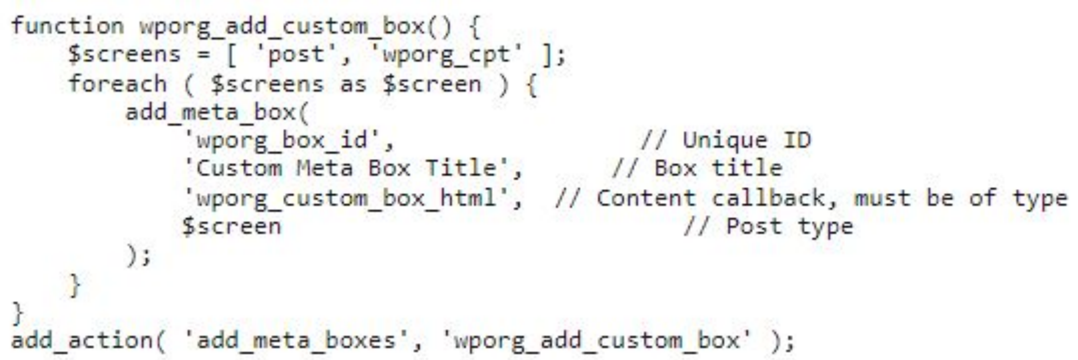
Adding Metadata add_post_meta(int $post_id, string key, string value);

Getting Metadata get_post_meta(int $post_id, string key, boolean single);

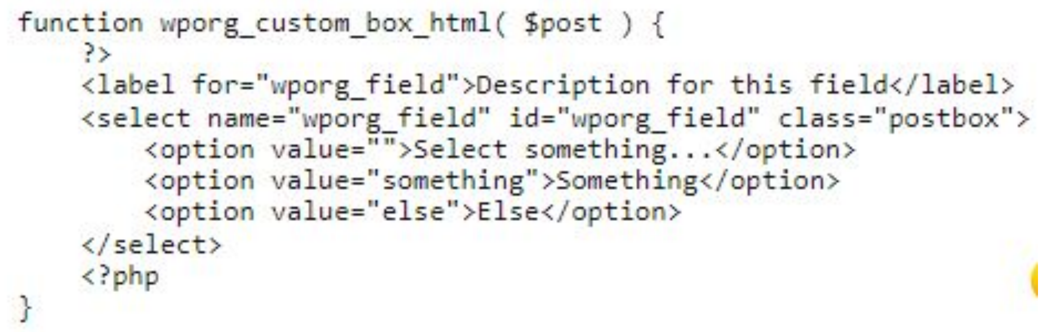
Updating Metadata update_post_meta(int $post_id, string key, string value);

Deleting Metadata delete_post_meta(int $post_id, string key);

Adding Meta Boxes



```
function wporg_add_custom_box() {
    $screens = [ 'post', 'wporg_cpt' ];
    foreach ( $screens as $screen ) {
        add_meta_box(
            'wporg_box_id',                 // Unique ID
            'Custom Meta Box Title',      // Box title
            'wporg_custom_box_html',  // Content callback, must be of type
            $screen                            // Post type
        );
    }
}
add_action( 'add_meta_boxes', 'wporg_add_custom_box' );
```



```
function wporg_custom_box_html( $post ) {
    ?>
    <label for="wporg_field">Description for this field</label>
    <select name="wporg_field" id="wporg_field" class="postbox">
        <option value="">Select something...</option>
        <option value="something">Something</option>
        <option value="else">Else</option>
    </select>
    <?php
}
```

Getting Values and Pass to Meta Box

```
function wporg_custom_box_html( $post ) {
    $value = get_post_meta( $post->ID, '_wporg_meta_key', true );
    ?>
    <label for="wporg_field">Description for this field</label>
    <select name="wporg_field" id="wporg_field" class="postbox">
        <option value="">Select something...</option>
        <option value="something" <?php selected( $value, 'something' ); ?>>Something</option>
        <option value="else" <?php selected( $value, 'else' ); ?>>Else</option>
    </select>
    <?php
}
```

Saving Values

```
function wporg_save_postdata( $post_id ) {
    if ( array_key_exists( 'wporg_field', $_POST ) ) {
        update_post_meta(
            $post_id,
            '_wporg_meta_key',
            $_POST['wporg_field']
        );
    }
}
add_action( 'save_post', 'wporg_save_postdata' );
```

Plugin Security

Checking User Capabilities

```
if ( current_user_can( 'edit_others_posts' ) ) {
        /**
         * Add the delete link to the end of the post content.
         */
        add_filter( 'the_content', 'wporg_generate_delete_link' );

        /**
         * Register our request handler with the init hook.
         */
        add_action( 'init', 'wporg_delete_post' );
}
```

Data Validation

- Check that required fields have not been left blank
- Check that an entered phone number only contains numbers and punctuation
- Check that an entered postal code is a valid postal code
- Check that a quantity field is greater than 0

Data Validation

- `is_email()` will validate whether an email address is valid.
- `term_exists()` checks whether a tag, category, or other taxonomy term exists.
- `username_exists()` checks if username exists.
- `validate_file()` will validate that an entered file path is a real path (but not whether the file exists).

Securing (sanitizing) Input

- sanitize_email()
- sanitize_file_name()
- sanitize_hex_color()
- sanitize_hex_color_no_hash()
- sanitize_html_class()
- sanitize_key()
- sanitize_meta()
- sanitize_mime_type()
- sanitize_option()
- sanitize_sql_orderby()
- sanitize_text_field()
- sanitize_textarea_field()
- sanitize_title()
- sanitize_title_for_query()
- sanitize_title_with_dashes()
- sanitize_user()
- sanitize_url()
- wp_kses()
- wp_kses_post()

Securing (sanitizing) Output

- esc_attr() – Use on everything else that's printed into an HTML element's attribute.
- esc_html() – Use anytime an HTML element encloses a section of data being displayed. This **WILL NOT** display HTML as HTML (so `<strong>` would be output as is, instead of making something bold), it is meant for being used inside HTML and will remove your HTML.
- esc_js() – Use for inline Javascript, it escapes javascript for use in `<script>` tags.
- esc_textarea() – Use this to encode text for use inside a textarea element.
- esc_url() – Use on all URLs, including those in the `src` and `href` attributes of an HTML element.
- esc_url_raw() – Use when storing a URL in the database or in other cases where non-encoded URLs are needed.
- esc_xml() – Use to escape XML block.
- wp_kses() – Use to safely escape for all non-trusted HTML (post text, comment text, etc.). This **WILL** display HTML as HTML (so `<em>` will show as emphasized text)
- wp_kses_post() – Alternative version of `wp_kses()` that automatically allows all HTML that is permitted in post content.
- wp_kses_data() – Alternative version of `wp_kses()` that allows only the HTML permitted in post comments.

Nonces

“Nonce” is a portmanteau of “*N*umber used *ONCE*”. It is essentially a unique hexadecimal serial number used to verify the origin and intent of requests for security purposes. As the name suggests, each nonce can only be used once.

Nonces

```
function wporg_generate_delete_link( $content ) {
    // Run only for single post page.
    if ( is_single() && in_the_loop() && is_main_query() ) {
        // Add query arguments: action, post, nonce
        $url = add_query_arg(
            [
                'action' => 'wporg_frontend_delete',
                'post'   => get_the_ID(),
                'nonce'  => wp_create_nonce( 'wporg_frontend_delete' ),
            ], home_url()
        );
        return $content . ' <a href="' . esc_url( $url ) . '">' . esc_html__( 'Delete Post', 'wporg' ) .
'</a>';
    }
    return null;
}
```

Nonces

```
function wporg_delete_post() {
    if ( isset( $_GET['action'] )
         && isset( $_GET['nonce'] )
         && 'wporg_frontend_delete' === $_GET['action']
         && wp_verify_nonce( $_GET['nonce'], 'wporg_frontend_delete' ) ) {
        wp_trash_post( $post_id );
        // Redirect to admin page.
        $redirect = admin_url( 'edit.php' );
        wp_safe_redirect( $redirect );

        // We are done.
        die;
    }
}
```

Javascript, AJAX, & JQuery

Javascript and JQuery

- Javascript is an important component in many WordPress plugins.
- WordPress comes with a variety of JavaScript libraries bundled with core.
- One of the most commonly-used libraries in WordPress is jQuery because it is lightweight and easy to use.
- jQuery can be used in your plugin to manipulate the DOM object or to perform Ajax actions.

Server Side PHP and Enqueueing

```
wp_enqueue_script(
    'Ajax-script', //handle
    plugins_url( '/js/myjquery.js', __FILE__ ), //location
    array( 'jquery' ), //dependencies
    '1.0.,0', //version
    true //in footer
);
```

Localizing

```
wp_localize_script(
    'ajax-script',
    'my_ajax_obj',
    array(
        'ajax_url' => admin_url( 'admin-ajax.php' ),
        'nonce'   => $title_nonce,
    )
);
```

JQuery AJAX

```
$(".pref").change(function() {                //event
    var this2 = this;                         //use in callback
    $.post(my_ajax_obj.ajax_url, {            //POST request
        _ajax_nonce: my_ajax_obj.nonce,        //nonce
        action: "my_tag_count",               //action
        title: this.value                     //data
        }, function(data) {                       //callback
            this2.nextSibling.remove();           //remove current title
            $(this2).after(data);                 //insert server response
        }
    );
} );
```

Administration Menus

Top Level Menus

```
<?php
add_menu_page(
    string $page_title,
    string $menu_title,
    string $capability,
    string $menu_slug,
    callable $function = '',
    string $icon_url = '',
    int $position = null
);
```

Sub-Level Menus

```
add_submenu_page(
    string $parent_slug,
    string $page_title,
    string $menu_title,
    string $capability,
    string $menu_slug,
    callable $function = ''
);
```

Settings

Settings

WordPress provides two core APIs to make the administrative interfaces easy to build, secure, and consistent with the design of WordPress Administration.

The Settings API focuses on providing a way for developers to create forms and manage form data.

The Options API focuses on managing data using a simple key/value system.

Settings

WordPress provides two core APIs to make the administrative interfaces easy to build, secure, and consistent with the design of WordPress Administration.

The Settings API focuses on providing a way for developers to create forms and manage form data.

The Options API focuses on managing data using a simple key/value system.

Using the Settings API

```
register_setting( 'general', 'misc-test-field', [ 'type' => 'boolean', 'This is a test field' ] );

add_settings_section( 'misc-test-settings', 'Misc', function(){ echo "This is a test field group"; },
'general' );

add_settings_field( 'misc-test-field','Misc. Field', [ $this, 'misc_field' ], 'general',
'misc-test-settings', [
                    'type'      =>  'checkbox',
                    'name'      =>  'misc-test-field',
                    'checked'   =>  get_option( 'misc-test-field' ),
                ] );
```

Using the Options API

```
add_option( string key, mixed value, boolean autoload );

get_option( string key, mixed default );

update_option( string key, mixed value, boolean autoload );

delete_option( string key );
```

Custom Settings Page

1. Register settings, add setting sections and settings field in one hook.
2. Add the Custom Page

Cron

What is WP-Cron?

- how WordPress handles scheduling time-based tasks
- Several WordPress core features, such as checking for updates and publishing scheduled post, utilize WP-Cron.
- The “Cron” part of the name comes from the cron time-based task scheduling system that is available on UNIX systems.
- WP-Cron works by checking, on every page load, a list of scheduled tasks to see what needs to be run. Any tasks due to run will be called during that page load.

Default Intervals

- hourly
- twicedaily
- daily
- weekly (since WP 5.4)

Custom Interval

```
add_filter( 'cron_schedules', 'example_add_cron_interval' );
function example_add_cron_interval( $schedules ) {
    $schedules['five_seconds'] = array(
        'interval' => 5,
        'display'  => 'Every Five Seconds' ;
    return $schedules;
}
```

Scheduling WP-Cron Events

```
wp_schedule_event( time(), 'five_seconds', 'bl_cron_hook' );

if ( ! wp_next_scheduled( 'bl_cron_hook' ) ) {
    wp_schedule_event( time(), 'five_seconds', 'bl_cron_hook' );
}

add_action( 'bl_cron_hook', 'bl_cron_exec' );
```

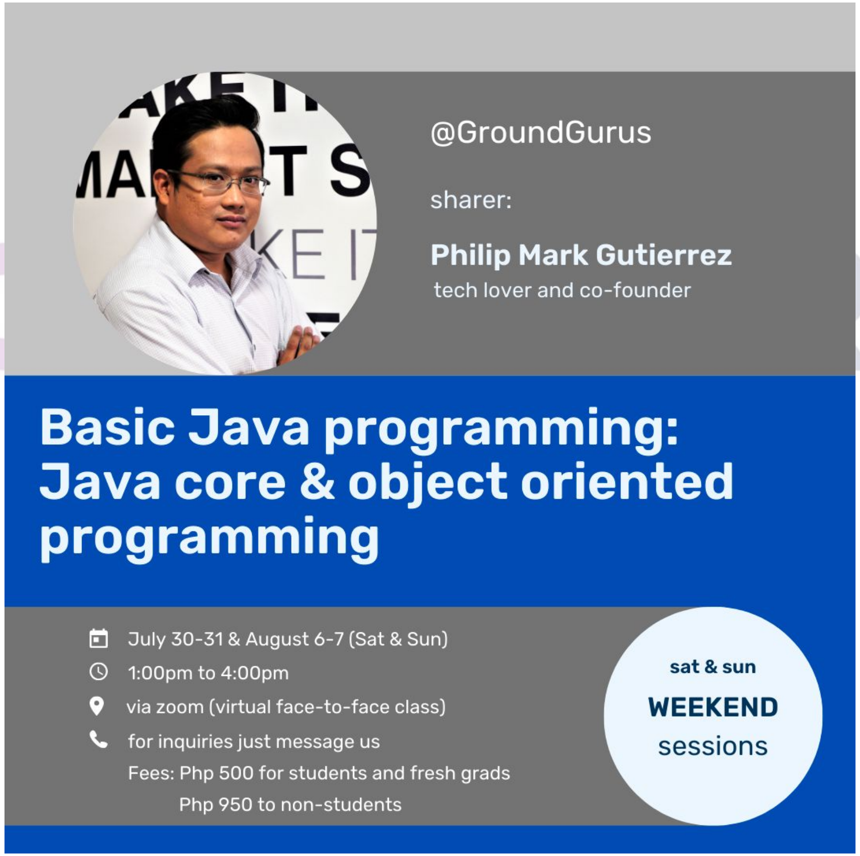
Unscheduling WP-Cron Events

```
register_deactivation_hook( __FILE__, 'bl_deactivate' );

function bl_deactivate() {
    $timestamp = wp_next_scheduled( 'bl_cron_hook' );
    wp_unschedule_event( $timestamp, 'bl_cron_hook' );
}
```

Questions?

Training Schedules



@GroundGurus

sharer:

Philip Mark Gutierrez
tech lover and co-founder

Basic Java programming: Java core & object oriented programming

July 30-31 & August 6-7 (Sat & Sun)
1:00pm to 4:00pm
via zoom (virtual face-to-face class)
for inquiries just message us
Fees: Php 500 for students and fresh grads
Php 950 to non-students

sat & sun
WEEKEND
sessions

Thank You