Moin!

Stephan
aka st3phan

/ WordPress-Fan und -Enthusiast

Kurz mal eben gefragt ...

„Wer von euch hat Code in der functions.php?“

Kurz mal eben gefragt ...

„Weißt Du nach 6 Monaten noch, wo du den Code eingefügt hast und was er macht?“

Ein inhaltlicher Überblick über die heutigen Themen

- **Ein kurzes “Moin”** → Begrüßung und Einstieg
- **Der lange Weg** → Vom Customizer bis hin zum MU-Plugin
- **Ein Blick über den Tellerrand** → Eigene Snippets verwalten
- **Code-Dokumentation** → kein “nice to have”!
- **Tor 1, 2 oder 3** → Deine Entscheidung
- **Ende gut alles gut** → Ein Fazit

</> Brauchen wir überhaupt eigenen Code?

Brauchen wir überhaupt eigenen Code?

ⓘ Nö, ... nicht unbedingt.

- Reichen die "Bordmittel" des Themes und der Plugins aus, ist eigener Code nicht notwendig.
- Für komplexe (Code-)Lösungen gibt es vielleicht schon etablierte Plugins oder Core-Funktionen. Dies vermeidet unnötige Risiken bei Wartungs-, Sicherheits- oder Update-Maßnahmen.

ⓘ Mit eigenem Code können wir aber ...

- Individuelle Anforderungen verwirklichen, die mit Standard-Plugins nicht oder nur unzureichend lösbar sind.
- Unter Umständen können Plugins eingespart und Ressourcen geschont werden.

Welche Code-Sprachen / Syntax können zum Einsatz kommen?

PHP	JS	HTML	CSS
Hauptsprache. serverseitige Logik, Themes, Plugins und Core-Funktionen.	für interaktive Elemente im Frontend (z. B. Gutenberg-Editor, dynamische Formulare).	„Markup-Sprache" für Struktur und Inhalte der Seiten.	„Stylesheet-Sprache" für die erweiterte Formatierung der HTML-Ausgabe.
+ SQL Datenbankabfragen mit SQL in PHP	+ Ajax asynchrone Serveranfragen in Kombination mit XML	+ XML Individuelles Markup zum Lesen, Schreiben und Durchsuchen von Daten	+ SASS (SCSS) Dynamische Erstellung von Stylesheets
		+ Sprachdateien .po-Datei als menschenlesbare Datei (gettext). .mo-Datei als binäre Version der .po-Datei.	

Der Lange Weg ...

Vom Customizer bis hin zum MU-Plugin

Der Theme-Customizer

CSS-Anpassungen

Der Theme-Customizer: CSS-Anpassungen

Vorteile

- Niedrige Einstiegshürde
- Kein PHP-Wissen notwendig
- Sichtbare Ergebnisse (ja/nein)
- Ideal für kleine, schnelle, sichbare Veränderungen

Nachteile

- Nur CSS-Code möglich
- Nicht update- oder wechselsicher
- Umfangreicher CSS-Code wird unübersichtlich
- Keine Strukturierung möglich
- CSS-Code wird auf allen Seiten geladen
- Nicht fürs Backend geeignet

Fazit und Wissenswertes

- Themeabhängigkeit
- Inline CSS-Einbindung im <head>
- Keine Revisionen

Mögliche Einsatzbereiche

- Kleine Layout-Fixes
- Farb- / Schriftanpassungen
- "Quick & Dirty"-Anpassungen für einzelne Fälle

„Nice-to-know" falls es mal hakt: Die CSS-Kaskade

Die Syntax deines CSS-Codes ist richtig, hat aber keine Auswirkung? Tipps:

- C in CSS = cascading
- Problem/Umstand: Dokumente können von einer Vielzahl von Style-Sheets formatiert werden.
- Kaskade wurde nicht berücksichtigt: Regelung der Gewichtung/Spezifität für CSS-Regeln und ihre Eigenschaften

Browser	Default-Style des User-Agents (z.B. Firefox, Chrome)
Browser-Benutzer	Benutzerdefinierter Style (z.B. Reader-Mode, Accessibility)
Autor	Theme-CSS, Plugin-CSS, Block-Editor, Customizer
@keyframe-Anmiationen	Animationen aus Theme oder Pagebuildern
!important (Autor)	!important-Regeln des Autors
!important (Benutzer)	!important-Regeln des Benutzers (z.B. Accessibility-Tools)
!important (Browser)	!important-Regeln des Browser (z.B. Barrierefreiheit)
CSS-Transitionen	Z.B. :hover oder :focus bei Zustandswechsel

[ ] Shortcodes

Einfache Content-Implementierung für „Einsteiger“

Shortcodes

Vorteile

- Einfache Platzhalter Logik / Wiederverwendbarkeit
- Block-Unterstützung / große Kompabilität
- Erweiterbar / eigene Shortcodes
- Dynamische Ausgaben möglich

Nachteile

- Ggf. Theme-abhängig / Namenskonflikte möglich
- Oft fehlendes visuelles Feedback
- Shortcode im Klartext bei Plugin-Deaktivierung / Theme-Wechsel
- Nicht überall zulässig

Fazit und Wissenswertes

- Nur wenig native Shortcodes (gallery, audio, video, etc.)
- Können Attribute/Parameter enthalten
- Funktioniert in Widgets, Blöcken, Menus

Mögliche Einsatzbereiche

- Dynamische Inhalte (Datum, Benutzernmame ...)
- Einbindung von Formularen, Downloads, Maps, Kalender, etc.
- Layout-Templates
- Copyright, Jahreszahlen, etc.

Funktionsweise von Shortcodes

Info

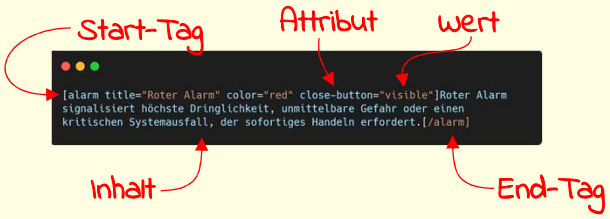
- Selbstschließender, minimaler Shortcode

```
[alarm-red]
```

- ... mit Attribut und Wert

```
[alarm-red info="Die Daten konnten nicht gespeichert werden!"]
```

- Umschließender, maximaler Shortcode



- Mindestens ein eindeutiger Name oder Start-Tag
- Optional: Attribute mit Werten
- Optional: Endtag zum Umschließen von Inhalt

Roter Alarm
Roter Alarm signalisiert höchste Dringlichkeit, unmittelbare Gefahr oder einen kritischen Systemausfall, der sofortiges Handeln erfordert.

Die functions.php

Für individuellen PHP-Code

(Child) Theme: functions.php

Hinweis

- Die functions.php des Themes sollte nicht zum von Speichern eigenem PHP-Code verwendet werden. Sie wird mit jedem Theme-Update neu geschrieben. Für diesen Zweck wird ggf. ein Child-Theme empfohlen.

Vorteile mit Child-Theme

- Code-Trennung vom Haupt-Theme
- Updatesicher

Nachteile mit Child-Theme

- Child-Theme notwendig
- Code-Verlust bei Theme-Wechsel (Bindung des Child-Theme an Haupt-Theme)

Fazit und Wissenswertes

- Themeabhängigkeit
- Reihenfolge: Haupttheme → Childtheme
- Zugriff per WP-Editor oder FTP (Entwicklung im eigenen Editor.

Mögliche Einsatzbereiche

- Theme-Anpassung
- Template-Overrides
- Hook-Nutzung

PHP-Prioritäten bei Hooks und Filtern

Hinweis

- Je kleiner die Zahl, desto früher wird ausgeführt
- Standard-Priorität ist 10
- Gleiche Priorität → Ausführung in Reihenfolge der Registrierung

```php
function meineFooterFunktion(){
    //meine neue funktion
}
add_action( 'wp_footer', 'meineFooterFunktion', 10 );
```

Priorität

Inoffizielle „Faustregel“ der Prioritäten

	gängige Verwendung
1 – 5	Extrem früh, selten nötig
10	Standard, (Core, Plugins, Themes)
20 – 30	Erweiterungen, Anpassungen
50	Bewusst „nachgelagerte“ Logik
99	Sehr spät, Overrides, Fixes
100+	„Letzter Rettungsanker“

Code-“Schnipsel“ mit Snippet-Plugin verwalten

Vorteile

- Unabhängig / Kein Theme-Bezug
- Gute Übersicht einzelner Skripte
- Updatesicher

Nachteile

- Plugin notwendig (Abhängigkeit)
- Meist keine echte Dateistruktur abbildbar.
- Geringe Skalierbarkeit
- Fehlende Namespaces

Fazit und Wissenswertes

- Das Snippet-Plugin auf gewünschte Funktionen testen und prüfen.
- Plugin sollte Code-Validierung und Highlighting enthalten.

Mögliche Einsatzbereiche

- Verwendung von Hooks und Filtern
- Temporäre (Test-)Lösungen
- Ersatz von einzelnen Plugin-Funktionen

Code-Plugins: Was gibt es da so am Markt?

Einige Beispiele

- WP Code
- Code Snippets
- Code Kit
- Simple Custom CSS and JS
- Fluent Snippets
- Woody Code Snippets
- WP Coder
- Simple CSS
- Shortcodes Plugin
- Shortcoder
- Snippet Shortcodes
- Header Footer Code Manager
- Insert HTML Snippet

Eigenes Plugin

Für individuellen Code

Eigenes Plugin für eigene Funktionen

Vorteile

- Eigene Dateistruktur und Architektur möglich
- Trennung von Logik und Darstellung
- Professioneller Ansatz
- Ideal für Team- und Kundenprojekte

Nachteile

- Höhere Einstiegshürde
- "Übertrieben" für kleinere Änderungen
- Pflegebedürftig

Fazit und Wissenswertes

- Vollständig Theme-unabhängig
- Kann versioniert werden.
- Muss gewartet werden.
- Sollte dokumentiert werden

Mögliche Einsatzbereiche

- Individuelle Business-Logik (angepasste Funktionen im Hintergrund)
- Wiederverwendbare Funktionen
- Kundenspezifische Funktionen

In 6 Schritten zum eigenen Plugin (Textausgabe im Footer)

Ordner und Datei anlegen

- 1. Ordner 'my-first-plugin' anlegen.
- 2. Im Ordner 'myfirst-plugin' die Datei 'my-first-plugin.php' anlegen.
- 3. PHP-Datei: Header und Funktion einfügen

```php
<?php
/**
 * Plugin Name: Mein erstes Plugin
 * Plugin URI:  https://example.com
 * Description: Ein kleines Beispiel-Plugin für ein Meetup.
 * Version:     1.0.0
 * Author:      Dein Name
 * Author URI:  https://example.com
 * Text Domain: my-first-plugin
 */

defined( 'ABSPATH' ) || exit;

add_action( 'wp_footer', function () {
    echo '<p style="text-align:center;">Hallo aus meinem ersten Plugin 🔥</p>';
});
```

Plugin installieren

- 4. Verzeichnis und Datei zu einem ZIP (mein-plugin.1.0.zip) packen
- 5. Im Wordpress Dashboard → Plugins → Plugin hinzufügen das Zip-File hochladen
- 6. Nach erfolgreicher Installation kann das Plugin im Wordpress Dashboard → Plugins → installierte Plugins aktiviert werden.

Wichtiger Hinweis

- Stelle sicher, dass Du Dateizugriff Zugriff per FTP oder über dein Webhosting hast, um im Falle eines Fehlers das Plugin durch Umbennenn zu deaktivieren!

Infos zum Plugin-Header

Minimaler Plugin-Header (Pflichtfeld)

```
/*
 * Plugin Name: YOUR PLUGIN NAME
 */
```

Voller Plugin-Header

```
/*
 * Plugin Name:       My Basics Plugin
 * Plugin URI:        https://example.com/plugins/the-basics/
 * Description:       Handle the basics with this plugin.
 * Version:           1.10.3
 * Requires at least: 5.2
 * Requires PHP:      7.2
 * Author:            John Smith
 * Author URI:        https://author.example.com/
 * License:           GPL v2 or later
 * License URI:       https://www.gnu.org/licenses/gpl-2.0.html
 * Update URI:        https://example.com/my-plugin/
 * Text Domain:       my-basics-plugin
 * Domain Path:       /languages
 * Requires Plugins:  my-plugin, yet-another-plugin
 */
```

Hinweis

- **WordPress vergleicht die Versionen**
- **Kann mit DocBlock kombiniert werden**

Must-Use Plugins (MU-Plugin)

Individueller Code im Hintergrund

MU-Plugin („must use“) für eigene Funktionen / Konfigurationen

Vorteile

- Kein versehentliches Deaktivieren / Löschen
- Ideal für eigene Plattformstandards
- Ideal für Agenturen

Nachteile

- Wenig bekannt
- Fehlen im Plugin-Screen (Kein über Plugins)
- Höhere Verantwortung (bei fatalen Fehlern)
- Erfordert ggf. FTP-Zugang

Fazit und Wissenswertes

- Verzeichnis /wp-content/mu-plugins
- Werden immer (automatisch) geladen
- Automatische Aktualisierung
- Können nicht deaktiviert werden.
- Normale Plugins können als MU-Plugin fehlerhaft sein
- Ggf. MU Manager installieren
- Nicht mit WP-Editor zu bearbeiten

Mögliche Einsatzbereiche

- Sicherheitsregeln
- Zentrale Standards
- Kunden- / Agentur-Grundlagen
- Multisite-Konfigurationen

Code-Dokumentation

Kein „nice to have“!

Code-Dokumentation: Lohnt sich das?

Vorteile

- Nachvollziehbarkeit zu einem späteren Zeitpunkt
- Nachvollziehbarkeit durch andere
- Bessere Auffindbarkeit von Anpassungen
- Bessere Übersicht
- Spart langfristig Entwicklungs- / Arbeitszeit

Nachteile

- Erfordert Zeit und Disziplin

Fazit und Wissenswertes

- "Ausklammern" in PHP, JS, HTML und CSS
- Script-Header: Plugin-Header, DocBlock
- Readme.txt

Mögliche Einsatzbereiche

- "Ausklammern" in PHP, JS, HTML und CSS
- Script-Header: Plugin-Header, DocBlock
- Readme.txt

Code-Dokumentation: Lohnt sich das?

PHP

```php
// Dies ist ein einzeiliger Kommentar
# Auch ein Kommentar
/*
   Mehrzeiliger
   Kommentar
*/
echo "Hallo Welt"; // Ein Kommentar nach Code
```

PHP-Kommentare werden serverseitig entfernt und sind nicht im Quellcode sichtbar.

HTML / XML

```html
<!-- <p>Dieser Absatz wird nicht angezeigt</p> -->
<div>Sichtbarer Inhalt</div>
```

Kommentare dürfen nicht in anderen Tags (z.B. in <p>) verschachtelt werden.

JavaScript

```javascript
// alert('Test');
/*
console.log("Nicht ausgeführt");
*/
```

CSS

```css
/* body { background-color: blue; } */
p { color: red; } /* Text ist rot */
```

Code-Dokumentation: DocBlock

PHP und ggf. andere

```php
<?php
/**
 * Kurze Zusammenfassung der Datei (Summary).
 *
 * Eine ausführlichere Beschreibung der Datei oder des Moduls,
 * die auch über mehrere Zeilen gehen kann. Hier können Details
 * zur Funktionsweise oder Abhängigkeiten stehen.
 *
 * @category   IhrProjekt
 * @package    ModulName
 * @author     Max Mustermann <m.mustermann@beispiel.de>
 * @copyright  2026 Mustermann Digital GmbH
 * @license    https://opensource.org MIT License
 * @version    GIT: 1.2.0
 * @link       https://projekt-website.de
 * @since      Datei verfügbar seit Release 1.0.0
 */

// Hier beginnt der eigentliche Programmcode
namespace App\Controller;

class BeispielController {
    // ...
}
```

Regeln

- Ein DocBlock muss mit /** beginnen, um erkannt zu werden.
- Jede Zeile beginnt mit einem *
- Informationen wie @author, @package, oder @license sind Standard-Tags, die übergreifend genutzt werden können
- Der Dateikopf muss der erste Block in der Datei sein.
- Einige Editoren (z.B. PhpStorm) erzeugen den DocBlock automatisch.

Ein Blick über den Tellerrand

Eigene Snippets verwalten

Wann lohnt sich eine Code-Hosting (GitHub, GitLab o.a.)?

Vorteile

- Versionierung und Rückverfolgbarkeit von Änderungen
- Ideal für Teams oder Agenturen
- Vermeidet "copy & paste" aus letztem Projekt

Nachteile

- Lohnt sich nur bei umfangreichen Codesammlungen oder Plugins
- Pflegeaufwand
- Datenschutz berücksichtigen (ggf. Eigenes Git betreiben)

Fazit und Wissenswertes

- Code kann öffentlich / privat sein
- Erfordert eine Lizenz beim Veröffentlichen
- Keine Plugin-UI (nicht sichtbar)

Mögliche Einsatzbereiche

- Kundenspezifischer Code
- Zentrale Standards
- Kunden- / Agentur-Grundlagen
- Multisite-Konfigurationen

Coding mit KI?

Nicht nur ein Trend ...

Eigener Code mit AI / KI? Macht das Sinn?

Einige Beispiele

MS Copilot

Claude AI

Open AI / Chat GPT

Telex by Automattic

Vorteile

- Snippets / kleine Plugins auch von „Nicht-Entwicklern“ umsetzbar.
- „Learning by doing“ für kleine Anpassungen.
- Vermeidung unnötigen Plugins / Erweiterungen

Nachteile

- Nicht fehlerfrei
- Nicht immer ist alles möglich
- Updates / Weiterentwicklungen bei fehlendem Know-how schwierig / aufwendig

Fazit und Wissenswertes

- Verschiedene KI-Modell mit unterschiedlichen Ergebnissen
- KI erfährt immer mehr Implementierung in WP
- Ein Testen / eine Fehlersuche kann aufwendig sein

Mögliche Einsatzbereiche

- Kundenspezifischer Code
- Zentrale Standards
- Kunden- / Agentur-Grundlagen
- Multisite-Konfigurationen

Tor 1, 2 oder 3

Eine Entscheidungshilfe

Worauf läuft es hinaus? Deine Entscheidung ...

Customizer ✗	• Nur CSS, themeabhängig
Theme: functions.php ✗	• Nicht updatesicher
Child-Theme: functions.php ✓	• Globales PHP (ideal für wenige Anpassungen) • Indirekt auch Content als JS, HTML, CSS
Snippets-Plugin ✓	• Ideal für individuelle Anpassungen (PHP, JS, HTML, CSS) • Plugin auf benötigte Funktionen prüfen (z.B. Logik)
Eigens Plugin ?	• Globales PHP → theme- und updatesicher • Plugin-Architektur möglich • Erfordert Wartung
Eigens MU- Plugin ?	• Wie "Eigenes Plugin" (s.o.) • Ohne UI (unsichtbar / kann nicht deaktiviert werden) • Automatische Updates

Ende gut — alles gut!

Ein Fazit

Fazit: Ende gut — alles gut ;-)

Gedanken und zu stellende Fragen

- Möchtest du WP erweitern? → Fehlen Anpassungen / Funktionen?
- Kleine Anpassungen / Erweiterungen erfordern nicht gleich ein umfangreiches Plugin!
- Man muss das Rad nicht neu erfinden: Oft gibt es schon passende Code-Schnipsel!
- Möchtest du eigenen Code einsetzen?
  - Kenne die Möglichkeiten deiner WP-Installation! (Wo und wie kann ich eigenen Code einfügen)
  - Ggf. bei der Projektplanung den Einsatz eines Snippet-Plugins oder eigener (MU-)Plugins mit einbeziehen.
  - Testen, testen, testen ...
  - Codepflege und Dokumentation bedenken

Danke und Tschüß!

Stephan
aka st3phan

wordpress@st3phan.de
st3phan76

credits:

- carbon.now.sh (Codeboxen)
- lucide.org (Icons)

https://st3phan.de/wp/hhmeetup_code.pdf