How to override parent theme functions in a child theme

Customizations · https://mhthemes.com/support/knb/overriding-parent-theme-functions-child-theme-tutorial/

The best practice to customize your WordPress (parent) theme and override template files or functions is by using a child theme. We've already published a basic tutorial on how to create and use child themes. However, while overriding template files in a child theme is easy, overriding parent theme functions in a child theme takes some additional effort if you want to do it the right way.



Image Source: TBIT – Pixabay.com / License: CC0 Public Domain

When it comes to overriding parent theme functions in your child theme, you have to keep track of a couple of things. Beforehand, it is important to know that all functions in your parent theme are running as well when using a child theme. However, functions in your child theme will be loaded first. With this in mind, there basically are three common ways to override parent theme functions in WordPress while using a child theme based on the given scenario:

- Taking advantage of pluggable functions - an easy way for overriding parent theme functions but only possible if your parent theme is making use of pluggable functions.
- Specify the priority for function execution - you can determine in which order functions are running in WordPress - but sometimes setting the priority isn't enough to override functions.
- Making use of hooks, actions, and filters - applies when you want to *prevent* the outcome of a parent theme function taking effect on your website. After you've unhooked the parent theme function, you can add new functionality with your new custom function in your child theme.

1. Taking advantage of pluggable functions

Let's start with pluggable functions. What does the word *pluggable* actually mean? Pluggable means that the parent theme function is wrapped in a *function_exists()* if-condition to check whether the function already exists - if so, the function will not be executed, otherwise WordPress will run the function. Here is an example of a pluggable function with conditional tag:

Example of a pluggable function

```
if (!function_exists('my_parent_theme_function')) {
    function my_parent_theme_function() {
        // Code of your pluggable function
    }
}
```

Now remember what was mentioned earlier - child theme functions are being executed first in WordPress. That means you can simply take the function from the parent theme, without the conditional tag, put it in file *functions.php* of your child theme and let it run whatever you like. That way the function in your child theme will be executed instead of the function in your parent theme. And this is what the function without the if-statement would look like in your child theme:

Example of overriding pluggable function

```
function my_parent_theme_function() {
    // Code of your new function in the child theme
}
```

Please note, taking advantage of pluggable functions is not always advisable. Let's say you want to override certain functionality within a pluggable function. Overriding the entire parent function within your child theme could lead to problems in the future. If the parent function is updated one day, your child theme will load the *old* version of the function and override the new one, which may lead to compatibility issues or even security vulnerabilities on your WordPress website. For alternatives please have a look at: Making use of hooks and filters.

2. Specify priority for function execution

In case the WordPress theme you are using does not provide pluggable functions or those functions you are aiming to override are not pluggable, you need to help yourself with other methods. Functions in WordPress are being executed in a certain order and you can specify the priority for your custom function to determine the order of execution. By default, when nothing else is defined, this priority is 10. Here is an example of a function with default priority 10:

Example of function with default priority

```
function my_parent_theme_function() {
    // Code of your parent theme function
}
add_action('after_setup_theme', 'my_parent_theme_function');
```

In the following example you can define a higher priority of 20 for your function in a child theme, so the function with higher priority will run later and override the function in the parent theme:

Example of function with defined priority

```
function my_child_theme_function() {
    // Code of your child theme function
}
add_action('after_setup_theme', 'my_child_theme_function', 20);
```

3. Making use of hooks, actions and filters

If for some reason you want to remove the outcome of a certain function for good, then using priorities is not sufficient and not necessarily the correct approach. Furthermore, if there are no

pluggable functions available or if you do not want to override an entire pluggable function, you might want to take a close look at actions and filters.

With this approach you will remove the outcome of the parent theme function from the action hook it is attached to and thus prevent it from affecting your website. After that you can code custom functionality in your child theme and hook it accordingly for use on your website. Here is an example of a function attached to the *after_setup_theme* action hook:

Example of function attached to hook

```
function my_parent_theme_function() {
    // Code of your parent theme function
}
add_action('after_setup_theme', 'my_parent_theme_function');
```

To override this function in your custom child theme, you first would need to use *remove_action()* or *remove_filter()*, depending on what is given to unhook the function in your child theme. In this case the function has been added with *add_action()*, so you need to use *remove_action()*:

Example of removing function from hook

```
remove_action('after_setup_theme', 'my_parent_theme_function');
```

With this in mind, you can now code a custom function in your child theme to unhook the function from your parent theme. It's important to know that in order to unhook the parent theme function, you would either need to use an action hook that is called after the hook which is being used in the parent theme or you would need to work with priorities instead to ensure that your child theme function is executed after the parent theme function in order to unhook the function accordingly.

Example of unhooking the parent theme function

```
function remove_my_parent_theme_function() {
    remove_action('after_setup_theme', 'my_parent_theme_function');
}
add_action('wp_loaded', 'remove_my_parent_theme_function');
```

After you've successfully removed the parent theme function, you can know code a new custom function in your child theme based on your personal requirements.

Conclusion: Overriding parent theme functions in a child theme

To give a résumé, all these three methods can be used, but it always depends on the given scenario. Removing functions from hooks takes a little more effort, but is perfect for selectively overriding parent theme functions in child themes without the need to override entire pluggable functions. While keeping future theme compatibility and security in mind, using pluggable functions to override parent theme functions is advisable if the function runs a simple process only.

Have you ever tried to override parent theme functionality in a child theme? Which approach have you used and what were the issues you were facing while overriding the parent theme? We're excited to learn more about your experiences while coding child themes in the comments.