Actions

- Adding an Action
  - Create a callback function
  - Assign (hook) your callback function
  - Additional Parameters
    - Priority
    - Number of Arguments

Actions are one of the two types of Hooks. They provide a way for running a function at a specific point in the execution of WordPress Core, plugins, and themes. Callback functions for an Action do not return anything back to the calling Action hook. They are the counterpart to Filters. Here is a refresher of the difference between actions and filters.

Adding an Action

The process of adding an action includes two steps:

Create a *callback function*

First, create a *callback function*. This function will be run when the action it is hooked to is run.

The callback function is just like a normal function: it should be prefixed, and it should be in `functions.php` or somewhere callable. The parameters it should accept will be defined by the action you are hooking to; most hooks are well defined, so review the hooks docs to see what parameters the action you have selected will pass to your function.

Assign (*hook*) your callback function

Second, add your callback function to the action. This is called *hooking* and tells the action to run your callback function when the action is run.

When your callback function is ready, use add_action() to hook it to the action you have selected. At a minimum, `add_action()` requires two parameters:

1. `string $hook_name` which is the name of the action you're hooking to, and
2. `callable $callback` the name of your callback function.

The example below will run `wporg_callback()` when the `init` hook is executed:

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

You can refer to the Hooks chapter for a list of available hooks.

As you gain more experience, looking through WordPress Core source code will allow you to find the most appropriate hook.

Additional Parameters

`add_action()` can accept two additional parameters, `int $priority` for the priority given to the callback function, and `int $accepted_args` for the number of arguments that will be passed to the callback function.

Priority

Many callback functions can be hooked to a single action. The `init` hook for example gets a lot of use. There may be cases where you need to ensure that your callback function runs before or after other callback functions, even when those other functions may not yet have been hooked.

WordPress determines the order that callback functions are run based on two things: The first way is by manually setting the *priority*. This is done using the third argument to `add_action()`.

Here are some important facts about priorities:

- priorities are positive integers, typically between 1 and 20
- the default priority (meaning, the priority assigned when no `priority` value is manually supplied) is 10
- there is no theoretical upper limit on the priority value, but the realistic upper limit is 100

A function with a priority of 11 will run *after* a function with a priority of 10; and a function with a priority of 9 will run *before* a function with a priority of 10.

The second way that callback function order is determined is simply by the order in which it was registered *within the same priority value*. So if two callback functions are registered for the same hook with the same priority, they will be run in the order that they were registered to the hook.

For example, the following callback functions are all registered to the `init` hook, but with different priorities:

```
add_action( 'init', 'wporg_callback_run_me_late', 11 );
add_action( 'init', 'wporg_callback_run_me_normal' );
add_action( 'init', 'wporg_callback_run_me_early', 9 );
add_action( 'init', 'wporg_callback_run_me_later', 11 );
```

In the example above:

- The first function run will be `wporg_callback_run_me_early()`, because it has a manual priority of 9
- Next, `wporg_callback_run_me_normal(),` because it has no priority set and so its priority is 10
- Next, `wporg_callback_run_me_late()` is run because it has a manual priority of 11
- Finally, `wporg_callback_run_me_later()` is run: it also has a priority of 11, but it was hooked after `wporg_callback_run_me_late()`.

Number of Arguments

Sometimes it's desirable for a callback function to receive some extra data related to the action being hooked to.

For example, when WordPress saves a post and runs the `save_post` hook, it passes two parameters to the callback function: the ID of the post being saved, and the post object itself:

```
do_action( 'save_post', $post->ID, $post );
```

When a callback function is registered for the `save_post` hook, it can specify that it wants to receive those two parameters. It does so by telling `add_action` to expect them by (in this case) putting `2` as the fourth argument:

```
add_action( 'save_post', 'wporg_custom', 10, 2 );
```

In order to actually receive those parameters in your callback function, modify the parameters your callback function will accept, like this:

```
function wporg_custom( $post_id, $post ) {
  // do something
}
```

[tip]]It's good practice to give your callback function parameters the same name as the passed parameters, or as close as you can.[/tip]

□□□ #2
□ □□□ □□□ 2026-10-26 15:37:07 UTC
□ □□□ □□□ 2026-10-26 15:37:07 UTC