WP UNIT 5

```
add_action( string $tag, callable $function_to_add, int $priority = 10, int $accepted_args = 1 )
```

Hooks a function on to a specific action.

Description #Description

Actions are the hooks that the WordPress core launches at specific points during execution, or when specific events occur. Plugins can specify that one or more of its PHP functions are executed at these points, using the Action API.

Parameters #Parameters

$tag

(string) (Required) The name of the action to which the $function_to_add is hooked.

$function_to_add

(callable) (Required) The name of the function you wish to be called.

$priority

(int) (Optional) Used to specify the order in which the functions associated with a particular action are executed. Lower numbers correspond with earlier execution, and functions with the same priority are executed in the order in which they were added to the action.

Default value: 10

$accepted_args

(int) (Optional) The number of arguments the function accepts.

Default value: 1

Return #Return

(true) Will always return true.

More Information #More Information

Usage **#Usage**

```
1  add_action( $hook, $function_to_add, $priority, $accepted_args );
```

`add_filter( string $tag, callable $function_to_add, int $priority = 10, int $accepted_args = 1 )`
Hook a function or method to a specific filter action.

Description #Description

WordPress offers filter hooks to allow plugins to modify various types of internal data at runtime.

A plugin can modify data by binding a callback to a filter hook. When the filter is later applied, each bound callback is run in order of priority, and given the opportunity to modify a value by returning a new value.

The following example shows how a callback function is bound to a filter hook.

Note that `$example` is passed to the callback, (maybe) modified, then returned:

```
function example_callback( $example ) {
    // Maybe modify $example in some way.
    return $example;
}
add_filter( 'example_filter', 'example_callback' );
```

Bound callbacks can accept from none to the total number of arguments passed as parameters

Parameters #Parameters

$tag

(string) (Required) The name of the filter to hook the $function_to_add callback to.

$function_to_add

(callable) (Required) The callback to be run when the filter is applied.

$priority

(int) (Optional) Used to specify the order in which the functions associated with a particular action are executed. Lower numbers correspond with earlier execution, and functions with the same priority are executed in the order in which they were added to the action.
Default value: 10

$accepted_args

(int) (Optional) The number of arguments the function accepts.
Default value: 1

`register_post_type( string $post_type, array|string $args = array() )`
Registers a post type.

Description #Description

Note: Post type registrations should not be hooked before the ‘init’ action. Also, any taxonomy connections should be registered via the `$taxonomies` argument to ensure consistency when hooks such as ‘parse_query’ or ‘pre_get_posts’ are used.
Post types can support any number of built-in core features such as meta boxes, custom fields, post thumbnails, post statuses, comments, and more. See the `$supports` argument for a complete list of supported features.

Top ↑

Parameters #Parameters

$post_type

(string) (Required) Post type key. Must not exceed 20 characters and may only contain lowercase alphanumeric characters, dashes, and underscores.
See sanitize_key().

$args

(array|string) (Optional) Array or string of arguments for registering a post type.

- **'label'**
  (string) Name of the post type shown in the menu. Usually plural. Default is value of $labels['name'].
- **'labels'**
  (array) An array of labels for this post type. If not set, post labels are inherited for non-hierarchical types and page labels for hierarchical ones.
  See get_post_type_labels() for a full list of supported labels.
- **'description'**
  (string) A short descriptive summary of what the post type is.
- **'public'**
  (bool) Whether a post type is intended for use publicly either via the admin interface or by front-end users. While the default settings of $exclude_from_search, $publicly_queryable, $show_ui, and $show_in_nav_menus are inherited from public, each does not rely on this relationship and controls a very specific intention. Default false.

- **'hierarchical'**
  (bool) Whether the post type is hierarchical (e.g. page). Default false.

`register_taxonomy( string $taxonomy, array|string $object_type, array|string $args = array() )`

Creates or modifies a taxonomy object.

Description #Description

Note: Do not use before the 'init' hook.

A simple function for creating or modifying a taxonomy object based on the parameters given. If modifying an existing taxonomy object, note that the `$object_type` value from the original registration will be overwritten.

Parameters #Parameters

$taxonomy

(string) (Required) Taxonomy key, must not exceed 32 characters.

$object_type

(array|string) (Required) Object type or array of object types with which the taxonomy should be associated.

$args

(array|string) (Optional) Array or query string of arguments for registering a taxonomy.

- **'labels'**
  (array) An array of labels for this taxonomy. By default, Tag labels are used for non-hierarchical taxonomies, and Category labels are used for hierarchical taxonomies. See accepted values in get_taxonomy_labels().
- **'description'**
  (string) A short descriptive summary of what the taxonomy is for.
- **'public'**
  (bool) Whether a taxonomy is intended for use publicly either via the admin interface or by front-end users. The default settings of `$publicly_queryable`, `$show_ui`, and `$show_in_nav_menus` are inherited from `$public`.

`register_nav_menu( string $location, string $description )`

Registers a navigation menu location for a theme.

Parameters #Parameters

$location

(string) (Required) Menu location identifier, like a slug.

$description

(string) (Required) Menu location descriptive text.

More Information #More Information

- See register_nav_menus() for creating multiple menus at once.
- This function automatically registers custom menu support for the theme therefore you do **not** need to call `add_theme_support( 'menus' );`
- This function actually works by simply calling register_nav_menus() in the following way:

```
1  register_nav_menus( array( $location => $description ) );
```

- You may use wp_nav_menu() to display your custom menu.

`register_sidebar( array|string $args = array() )`

Builds the definition for a single sidebar and returns the ID.

Description #Description

Accepts either a string or an array and then parses that against a set of default arguments for the new sidebar. WordPress will automatically generate a sidebar ID and name based on the current number of registered sidebars if those arguments are not included.

When allowing for automatic generation of the name and ID parameters, keep in mind that the incrementor for your sidebar can change over time depending on what other plugins and themes are installed.

Parameters #Parameters

$args

(array|string) (Optional) Array or string of arguments for the sidebar being registered.

- **'name'**
  (string) The name or title of the sidebar displayed in the Widgets interface. Default 'Sidebar $instance'.
- **'id'**
  (string) The unique identifier by which the sidebar will be called. Default 'sidebar-$instance'.
- **'description'**
  (string) Description of the sidebar, displayed in the Widgets interface. Default empty string.
- **'class'**
  (string) Extra CSS class to assign to the sidebar in the Widgets interface.
- **'before_widget'**
  (string) HTML content to prepend to each widget's HTML output when assigned to this sidebar. Default is an opening list item element.

- **'after_widget'**
  (string) HTML content to append to each widget's HTML output when assigned to this sidebar. Default is a closing list item element.

Return #Return

(string) Sidebar ID added to $wp_registered_sidebars global.