# LEARNING jquery-plugins

#jquery-plugins

# Table of Contents

# About

# Chapter 1: Getting started with jquery-plugins

## Remarks

A jQuery plugin is simply a new method that we use to extend jQuery's prototype object. By extending the prototype object you enable all jQuery objects to inherit any methods that you add. As established, whenever you call `jQuery()` you're creating a new jQuery object, with all of jQuery's methods inherited.

The idea of a plugin is to do something with a collection of elements. You could consider each method that comes with the jQuery core a plugin, like `.fadeOut()` or `.addClass()`.

You can make your own plugins and use them privately in your code or you can release them to the public. There are thousands of jQuery plugins available online. The barrier to creating a plugin of your own is so low that you can author one easily!

## Examples

### Installation or Setup

jQuery plugins are typically installed via NPM or Yarn (if hosted there), or referencing an external script file containing the plugin, whether from a relative directory or a CDN.

```
<script type="text/javascript" src="/path/to/plugin.jquery.js"></script>
```

### Basic plugin which change color of text to green.

```
// plugin initialization
$.fn.greenify = function() {
    // within the function you can use any of the jQuery methods
    // and `this` context refers to jQuery object
    this.css( "color", "green" );
};

// apply plugin
$( "a" ).greenify();
```

### Chaining

This works, but there are a couple of things we need to do for our plugin to survive in the real world. One of jQuery's features is chaining, when you link five or six actions onto one selector. This is accomplished by having all jQuery object methods return the original jQuery object again (there are a few exceptions: `.width()` called without parameters returns the width of the selected element, and is not chainable). Making our plugin method chainable takes one line of code:

```
$.fn.greenify = function() {
    this.css( "color", "green" );
    // return the reference for chaining
    return this;
}

$( "a" ).greenify().addClass( "greenified" );
```

## Supporting options with defaults

You can make your plugin customizable by accepting options.

```
$.fn.colourize = function(options) {

    // This is one method to support default options
    var style = $.extend({
        color: "green",
        backgroundColor: "white"
    }, options);

    // Set the colours on the current selection based on the option parameters
    return this.css({
        color: style.color,
        backgroundColor: style.backgroundColor
    });
};
```

Example usage:

```
$("button").colourize({
    color: "orange"
});
```

The default value for the colour option "green" gets overridden by `$.extend()` to be "orange".

## Typical jQuery Plugin Structure

While writing jQuery plugins is simple, we want to enclose our plugins in a local scope. This will avoid namespace conflicts as well as polluting the global namespace, on top of ensuring that jQuery is loaded before our plugin extends it.

```
// Encapsulate our plugins in a local scope
(function($) {

    // Plugin definition
    $.fn.colourize = function() {

        // Plugin code

    };

// Pass the jQuery object into our local scope
}(jQuery));
```

The local scope wrapper may be omitted on other examples to keep them simple and concise.

## Using the each() Method

Your typical jQuery object will contain references to any number of DOM elements, and that's why jQuery objects are often referred to as collections. If you want to do any manipulating with specific elements (e.g. getting a data attribute, calculating specific positions) then you need to use .each() to loop through the elements.

```
$.fn.myNewPlugin = function() {
    return this.each(function() {
        // Do something to each element here.
    });

    // return the reference for chaining
    return this;
};
```

Read Getting started with jquery-plugins online: https://riptutorial.com/jquery-plugins/topic/8277/getting-started-with-jquery-plugins

# Credits

| S. No | Chapters | Contributors |
|---|---|---|
| 1 | Getting started with jquery-plugins | Community, Pranav C Balan, Samuel Liew |