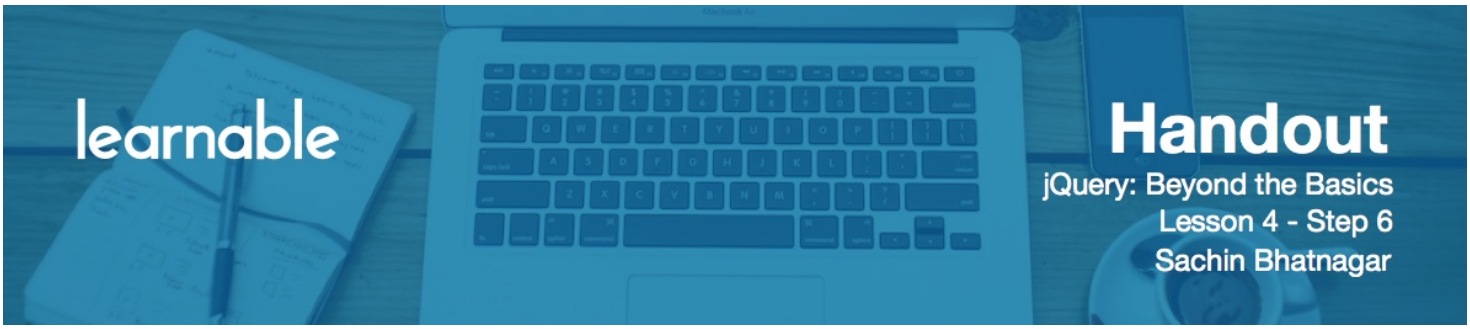

Introduction to writing jQuery plugins

We are going to examine the steps that it takes to build jQuery plugin. Writing your own plugin means that you can easily create reusable modules and features that you can drop into any website with ease.

A barebones jQuery plugin is an immediately invoked function expression where we pass in a reference to jQuery. Once that is done, we define the plugin by creating a function expression by calling `$.fn` followed by the name of the plugin. Whatever you want your plugin to do goes inside this function expression.

Building a formStyle plugin

Let's build a `formStyle` plugin that will take a bunch of input fields and style it. Additionally, we'll create options to let the user customize the border radius on the form fields and to invoke an optional validator.

Open up *customPlugin_Begin.htm* file.

In this example we have a `div` with a bunch of form fields that have not been styled yet. In the *css* folder, there is a stylesheet called *formStyleCore.css*, which contains the style definition for the form style class that will be applied to all form fields on which our plugin is invoked. Additionally, when the user clicks inside a form field to type, the active input style should set its background to green and animate its width to 350 pixels. We also have an optional style for when we want to check if the form field is empty or filled in.

So let's begin by bringing in this CSS file:

```
<link rel="stylesheet" href="css/formStyleCore.css" type="text/css" />
```

Next we'll create a script source tag to point to the *js/formStyle.js*:

```
<script src="js/formStyle.completed.js"></script>
```

This file doesn't exist yet so create it right away. Begin by creating an immediately invoked function expression that accepts a reference to jQuery, and declare our plugin:

```
(function($){
    $.fn.formStyle = function(options){}
}(jQuery));
```

Since the options will be optional we will create a set of default values for when the user does not specify any configuration. This would be done by creating a variable called `setOptions`:

```
(function($){
    $.fn.formStyle = function(options){
    var setOptions = $.extend({
        isEmpty:false,
        borderRadius:'0px'
    }, options);
    }
}(jQuery));
```

We are allowing to set two parameters:

- `isEmpty` is set to `false` by default. If enabled, it will highlight the right border of the form field as green if the field has been filled in with some data.
- `borderRadius` is set to zero pixels and will control the border radius.

Code the plugin:

```
(function($){
  $.fn.formStyle = function(options){
    var setOptions = $.extend({
      isEmpty:false,
      borderRadius:'0px'
    }, options);
    this
        .addClass("formStyle")
        .css("borderRadius", setOptions.borderRadius);
    this.on("focus", function(){
      $(this).addClass("activeInput");
    });
    this.on("blur", function(){
      $(this).removeClass("activeInput");
    });
    if(setOptions.isEmpty){
      this.on("keyup", function(){
        if($(this).val() != ''){
          $(this).addClass("filledInput");
        } else {
          $(this).removeClass("filledInput");
        }
      });
    }
    return this;
  };
}(jQuery));
```

To call this plugin, select all input fields and simply call the `formStyle` method:

```
$("input").formStyle({});
```

Invoking configuration options

Let's invoke our configuration options. Pass in an object to the `formStyle` method:

```
$("input").formStyle({
    isEmpty:true,
    borderRadius:'5px'
});
```

Check that in your browser.