jQuery®

Notes for Professionals

50+ pages

of professional hints and tricks

Contents

About

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:
https://goalkicker.com/jQueryBook

Please send feedback and corrections to web@petercv.com

Chapter 1: Getting started with jQuery

Version	Notes	Release Date
1.0	First stable release	2006-08-26
1.1		2007-01-14
1.2		2007-09-10
1.3	Sizzle introduced into core	2009-01-14
1.4		2010-01-14
1.5	Deferred callback management, ajax module rewrite	2011-01-31
1.6	Significant performance gains in the `attr()` and `val()` methods	2011-05-03
1.7	New Event APIs: `on()` and `off()`.	2011-11-03
1.8	Sizzle rewritten, improved animations and `$(html, props)` flexibility.	2012-08-09
1.9	Removal of deprecated interfaces and code cleanup	2013-01-15
1.10	Incorporated bug fixes and differences reported from both the 1.9 and 2.0 beta cycles	2013-05-24
1.11		2014-01-24
1.12		2016-01-08
2.0	Dropped IE 6–8 support for performance improvements and reduction in size	2013-04-18
2.1		2014-01-24
2.2		2016-01-08
3.0	Massive speedups for some jQuery custom selectors	2016-06-09
3.1	No More Silent Errors	2016-07-07
3.2	No More Silent Errors	2017-03-16
3.3	No More Silent Errors	2018-01-19

Section 1.1: Getting Started

Create a file `hello.html` with the following content:

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello, World!</title>
</head>
<body>
    <div>
        <p id="hello">Some random text</p>
    </div>
    <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
    <script>
        $(document).ready(function() {
            $('#hello').text('Hello, World!');
        });
    </script>
</body>
</html>
```

Live Demo on JSBin

Open this file in a web browser. As a result you will see a page with the text: `Hello, World!`

Explanation of code

1. Loads the jQuery library from the jQuery CDN:

   ```
   <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
   ```

   This introduces the $ global variable, an alias for the `jQuery` function and namespace.

   > *Be aware that one of the most common mistakes made when including jQuery is failing to load the library BEFORE any other scripts or libraries that may depend on or make use of it.*

2. Defers a function to be executed when the DOM (Document Object Model) is detected to be "ready" by jQuery:

   ```
   // When the `document` is `ready`, execute this function `...`
   $(document).ready(function() { ... });

   // A commonly used shorthand version (behaves the same as the above)
   $(function() { ... });
   ```

3. Once the DOM is ready, jQuery executes the callback function shown above. Inside of our function, there is only one call which does 2 main things:

   1. Gets the element with the `id` attribute equal to `hello` (our selector `#hello`). Using a selector as the passed argument is the core of jQuery's functionality and naming; the entire library essentially evolved from extending document.querySelectorAllMDN.
   2. Set the `text()` inside the selected element to `Hello, World!`.

      ```
      #    ↓ - Pass a `selector` to `$` jQuery, returns our element
      $('#hello').text('Hello, World!');
      #             ↑ - Set the Text on the element
      ```

For more refer to the jQuery - Documentation page.

Section 1.2: Avoiding namespace collisions

Libraries other than jQuery may also use $ as an alias. This can cause interference between those libraries and jQuery.

To release $ for use with other libraries:

```
jQuery.noConflict();
```

After calling this function, $ is no longer an alias for `jQuery`. However, you can still use the variable `jQuery` itself to access jQuery functions:

```
jQuery('#hello').text('Hello, World!');
```

Optionally, you can assign a different variable as an alias for jQuery:

```
var jqy = jQuery.noConflict();
```

```
jqy('#hello').text('Hello, World!');
```

Conversely, to prevent other libraries from interfering with jQuery, you can wrap your jQuery code in an immediately invoked function expression (IIFE) and pass in `jQuery` as the argument:

```
(function($) {
    $(document).ready(function() {
        $('#hello').text('Hello, World!');
    });
})(jQuery);
```

Inside this IIFE, $ is an alias for jQuery only.

Another simple way to **secure jQuery's $ alias and make sure DOM is ready**:

```
jQuery(function( $ ) { // DOM is ready
   // You're now free to use $ alias
   $('#hello').text('Hello, World!');
});
```

To summarize,

- `jQuery.noConflict()` : $ no longer refers to jQuery, while the variable `jQuery` does.
- `var jQuery2 = jQuery.noConflict()` - $ no longer refers to jQuery, while the variable `jQuery` does and so does the variable `jQuery2`.

Now, there exists a third scenario - What if we want jQuery to be available **only in `jQuery2`**? Use,

```
var jQuery2 = jQuery.noConflict(true)
```

This results in neither $ nor `jQuery` referring to jQuery.

This is useful when multiple versions of jQuery are to be loaded onto the same page.

```
<script src='https://code.jquery.com/jquery-1.12.4.min.js'></script>
<script>
    var jQuery1 = jQuery.noConflict(true);
</script>
<script src='https://code.jquery.com/jquery-3.1.0.min.js'></script>
<script>
    // Here, jQuery1 refers to jQuery 1.12.4 while, $ and jQuery refers to jQuery 3.1.0.
</script>
```

https://learn.jquery.com/using-jquery-core/avoid-conflicts-other-libraries/

Section 1.3: jQuery Namespace ("jQuery" and "$")

`jQuery` is the starting point for writing any jQuery code. It can be used as a function `jQuery(...)` or a variable `jQuery.foo`.

$ is an alias for `jQuery` and the two can usually be interchanged for each other (except where `jQuery.noConflict();` has been used - see Avoiding namespace collisions).

Assuming we have this snippet of HTML -

```
<div id="demo_div" class="demo"></div>
```

We might want to use jQuery to add some text content to this div. To do this we could use the jQuery `text()` function. This could be written using either `jQuery` or $. i.e. -

```
jQuery("#demo_div").text("Demo Text!");
```

Or -

```
$("#demo_div").text("Demo Text!");
```

Both will result in the same final HTML -

```
<div id="demo_div" class="demo">Demo Text!</div>
```

As $ is more concise than `jQuery` it is the generally the preferred method of writing jQuery code.

jQuery uses CSS selectors and in the example above an ID selector was used. For more information on selectors in jQuery see types of selectors.

Section 1.4: Loading jQuery via console on a page that does not have it

Sometimes one has to work with pages that are not using `jQuery` while most developers are used to have `jQuery` handy.

In such situations one can use `Chrome Developer Tools` console (F12) to manually add `jQuery` on a loaded page by running following:

```
var j = document.createElement('script');
j.onload = function(){ jQuery.noConflict(); };
j.src = "https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js";
document.getElementsByTagName('head')[0].appendChild(j);
```

Version you want might differ from above(`1.12.4`) you can get the link for one you need here.

Section 1.5: Include script tag in head of HTML page

To load **jQuery** from the official CDN, go to the jQuery website. You'll see a list of different versions and formats available.

Now, copy the source of the version of jQuery, you want to load. Suppose, you want to load **jQuery 2.X**, click **uncompressed** or **minified** tag which will show you something like this:

Copy the full code (or click on the copy icon) and paste it in the **<head>** or **<body>** of your html.

The best practice is to load any external JavaScript libraries at the head tag with the `async` attribute. Here is a demonstration:

```
<!DOCTYPE html>
    <html>
      <head>
         <title>Loading jquery-2.2.4</title>
         <script src="https://code.jquery.com/jquery-2.2.4.min.js" async></script>
      </head>
      <body>
          <p>This page is loaded with jquery.</p>
      </body>
   </html>
```

When using `async` attribute be conscious as the javascript libraries are then asynchronously loaded and executed as soon as available. If two libraries are included where second library is dependent on the first library is this case if second library is loaded and executed before first library then it may throw an error and application may break.

Section 1.6: The jQuery Object

Every time jQuery is called, by using `$()` or `jQuery()`, internally it is creating a **`new`** instance of `jQuery`. This is the source code which shows the new instance:

```
// Define a local copy of jQuery
jQuery = function( selector, context ) {

    // The jQuery object is actually just the init constructor 'enhanced'
    // Need init if jQuery is called (just allow error to be thrown if not included)
    return new jQuery.fn.init( selector, context );
}
```

Internally jQuery refers to its prototype as `.fn`, and the style used here of internally instantiating a jQuery object allows for that prototype to be exposed without the explicit use of **`new`** by the caller.

In addition to setting up an instance (which is how the jQuery API, such as `.each`, `children`,`filter`, etc. is exposed), internally jQuery will also create an array-like structure to match the result of the selector (provided that something other than nothing, `undefined`, **`null`**, or similar was passed as the argument). In the case of a single item, this array-like structure will hold only that item.

A simple demonstration would be to find an element with an id, and then access the jQuery object to return the underlying DOM element (this will also work when multiple elements are matched or present).

```
var $div = $("#myDiv");//populate the jQuery object with the result of the id selector
var div = $div[0];//access array-like structure of jQuery object to get the DOM Element
```

Chapter 2: Selectors

A jQuery selectors selects or finds a DOM (document object model) element in an HTML document. It is used to select HTML elements based on id, name, types, attributes, class and etc. It is based on existing CSS selectors.

Section 2.1: Overview

Elements can be selected by jQuery using jQuery Selectors. The function returns either an element or a list of elements.

Basic selectors

```
$("*")                      // All elements
$("div")                    // All <div> elements
$(".blue")                  // All elements with class=blue
$(".blue.red")              // All elements with class=blue AND class=red
$(".blue,.red")             // All elements with class=blue OR class=red
$("#headline")              // The (first) element with id=headline
$("[href]")                 // All elements with an href attribute
$("[href='example.com']")   // All elements with href=example.com
```

Relational operators

```
$("div span")          // All <span>s that are descendants of a <div>
$("div > span")        // All <span>s that are a direct child of a <div>
$("a ~ span")          // All <span>s that are siblings following an <a>
$("a + span")          // All <span>s that are immediately after an <a>
```

Section 2.2: Types of Selectors

In jQuery you can select elements in a page using many various properties of the element, including:

- Type
- Class
- ID
- Possession of Attribute
- Attribute Value
- Indexed Selector
- Pseudo-state

If you know CSS selectors you will notice selectors in jQuery are the same (with minor exceptions).

Take the following HTML for example:

```
<a href="index.html"></a>                  <!-- 1 -->
<a id="second-link"></a>                   <!-- 2 -->
<a class="example"></a>                    <!-- 3 -->
<a class="example" href="about.html"></a>  <!-- 4 -->
<span class="example"></span>              <!-- 5 -->
```

Selecting by Type:

The following jQuery selector will select all `<a>` elements, including 1, 2, 3 and 4.

```
$("a")
```

Selecting by Class

The following jQuery selector will select all elements of class `example` (including non-a elements), which are 3, 4 and 5.

```
$(".example")
```

Selecting by ID

The following jQuery selector will select the element with the given ID, which is 2.

```
$("#second-link")
```

Selecting by Possession of Attribute

The following jQuery selector will select all elements with a defined `href` attribute, including 1 and 4.

```
$("[href]")
```

Selecting by Attribute Value

The following jQuery selector will select all elements where the `href` attribute exists with a value of `index.html`, which is just 1.

```
$("[href='index.html']")
```

Selecting by Indexed Position (*Indexed Selector*)

The following jQuery selector will select only 1, the second **<a>** ie. the `second-link` because index supplied is `1` like `eq(1)` (Note that the index starts at `0` hence the second got selected here!).

```
$("a:eq(1)")
```

Selecting with Indexed Exclusion

To exclude an element by using its index `:not(:eq())`

The following selects **<a>** elements, except that with the class `example`, which is 1

```
$("a").not(":eq(0)")
```

Selecting with Exclusion

To exclude an element from a selection, use `:not()`

The following selects **<a>** elements, except those with the class `example`, which are 1 and 2.

```
$("a:not(.example)")
```

Selecting by Pseudo-state

You can also select in jQuery using pseudo-states, including `:first-child`, `:last-child`, `:first-of-type`, `:last-of-type`, etc.

The following jQuery selector will only select the first **<a>** element: number 1.

```
$("a:first-of-type")
```

Combining jQuery selectors

You can also increase your specificity by combining multiple jQuery selectors; you can combine any number of them or combine all of them. You can also select multiple classes, attributes and states at the same time.

```
$("a.class1.class2.class3#someID[attr1][attr2='something'][attr3='something']:first-of-type:first-child")
```

This would select an `<a>` element that:

- Has the following classes: `class1`, `class2`, and `class3`
- Has the following ID: `someID`
- Has the following Attribute: `attr1`
- Has the following Attributes and values: `attr2` with value `something`, `attr3` with value `something`
- Has the following states: `first-child` and `first-of-type`

You can also separate different selectors with a comma:

```
$("a, .class1, #someID")
```

This would select:

- All `<a>` elements
- All elements that have the class `class1`
- An element with the id `#someID`

Child and Sibling selection

jQuery selectors generally conform to the same conventions as CSS, which allows you to select children and siblings in the same way.

- To select a non-direct child, use a space
- To select a direct child, use a `>`
- To select an adjacent sibling following the first, use a `+`
- To select a non-adjacent sibling following the first, use a `~`

Wildcard selection

There might be cases when we want to select all elements but there is not a common property to select upon (class, attribute etc). In that case we can use the `*` selector that simply selects all the elements:

```
$('#wrapper *')    // Select all elements inside #wrapper element
```

Section 2.3: Caching Selectors

Each time you use a selector in jQuery the DOM is searched for elements that match your query. Doing this too often or repeatedly will decrease performance. If you refer to a specific selector more than once you should add it to the cache by assigning it to a variable:

```
var nav = $('#navigation');
nav.show();
```

This would replace:

```
$('#navigation').show();
```

Caching this selector could prove helpful if your website needs to show/hide this element often. If there are multiple elements with the same selector the variable will become an array of these elements:

```
<div class="parent">
    <div class="child">Child 1</div>
    <div class="child">Child 2</div>
</div>

<script>
    var children = $('.child');
    var firstChildText = children[0].text();
    console.log(firstChildText);

    // output: "Child 1"
</script>
```

NOTE: The element has to exist in the DOM at the time of its assignment to a variable. If there is no element in the DOM with a class called `child` you will be storing an empty array in that variable.

```
<div class="parent"></div>

<script>
    var parent   = $('.parent');
    var children = $('.child');
    console.log(children);

    // output: []

    parent.append('<div class="child">Child 1</div>');
    children = $('.child');
    console.log(children[0].text());

    // output: "Child 1"
</script>
```

Remember to reassign the selector to the variable after adding/removing elements in the DOM with that selector.

Note: When caching selectors, many developers will start the variable name with a $ to denote that the variable is a jQuery object like so:

```
var $nav = $('#navigation');
$nav.show();
```

Section 2.4: Combining selectors

Consider following DOM Structure

```
<ul class="parentUl">
    <li> Level 1
        <ul class="childUl">
            <li>Level 1-1 <span> Item - 1 </span></li>
            <li>Level 1-1 <span> Item - 2 </span></li>
        </ul>
    </li>
```

```
    <li> Level 2
        <ul class="childUl">
            <li>Level 2-1 <span> Item - 1 </span></li>
            <li>Level 2-1 <span> Item - 1 </span></li>
        </ul>
    </li>
</ul>
```

Descendant and child selectors

Given a parent `<ul>` - `parentUl` find its descendants (`<li>`),

1. Simple `$('parent child')`

   >> `$('ul.parentUl li')`

   This gets all matching descendants of the specified ancestor *all levels down*.

2. `>` - `$('parent > child')`

   >> `$('ul.parentUl > li')`

   This finds all matching children (*only 1st level down*).

3. Context based selector - `$('child','parent')`

   >> `$('li','ul.parentUl')`

   This works same as 1. above.

4. `find()` - `$('parent').find('child')`

   >> `$('ul.parentUl').find('li')`

   This works same as 1. above.

5. `children()` - `$('parent').find('child')`

   >> `$('ul.parentUl').children('li')`

   This works same as 2. above.

Other combinators
Group Selector : ","

Select all `<ul>` elements AND all `<li>` elements AND all `<span>` elements :

```
$('ul, li, span')
```

Multiples selector : "" (no character)

Select all `<ul>` elements with class `parentUl` :

```
$('ul.parentUl')
```

Adjacent Sibling Selector : "+"

Select all `<li>` elements that are placed immediately after another `<li>` element:

```
$('li + li')
```

General Sibling Selector : "~"

Select all `<li>` elements that are siblings of other `<li>` elements:

```
$('li ~ li')
```

Section 2.5: DOM Elements as selectors

jQuery accepts a wide variety of parameters, and one of them is an actual DOM element. Passing a DOM element to jQuery will cause the underlying array-like structure of the jQuery object to hold that element.

jQuery will detect that the argument is a DOM element by inspecting its nodeType.

The most common use of a DOM element is in callbacks, where the current element is passed to the jQuery constructor in order to gain access to the jQuery API.

Such as in the `each` callback (note: each is an iterator function).

```
$(".elements").each(function(){
    //the current element is bound to `this` internally by jQuery when using each
    var currentElement = this;

    //at this point, currentElement (or this) has access to the Native API

    //construct a jQuery object with the currentElement(this)
    var $currentElement = $(this);

    //now $currentElement has access to the jQuery API
});
```

Section 2.6: HTML strings as selectors

jQuery accepts a wide variety of parameters as "selectors", and one of them is an HTML string. Passing an HTML string to jQuery will cause the underlying array-like structure of the jQuery object to hold the resulting constructed HTML.

jQuery uses regex to determine if the string being passed to the constructor is an HTMLstring, and also that it *must* start with `<`. That regex is defined as `rquickExpr = /^(?:\s*(<[\w\W]+>)[^>]*|#([\w-]*))$/` (explanation at regex101.com).

The most common use of an HTML string as a selector is when sets of DOM elements need to be created in code only, often this is used by libraries for things like Modal popouts.

For example, a function which returned an anchor tag wrapped in a div as a template

```
function template(href,text){
    return $("<div><a href='" + href + "'>" + text + "</a></div>");
```

```
}
```

Would return a jQuery object holding

```
<div>
    <a href="google.com">Google</a>
</div>
```

if called as `template("google.com","Google")`.

Chapter 3: Each function

Section 3.1: jQuery each function

HTML:

```
<ul>
  <li>Mango</li>
  <li>Book</li>
</ul>
```

Script:

```
$( "li" ).each(function( index ) {
  console.log( index + ": " + $( this ).text() );
});
```

A message is thus logged for each item in the list:

0: Mango

1: Book

Chapter 4: Attributes

Section 4.1: Differece between attr() and prop()

`attr()` gets/sets the HTML attribute using the DOM functions `getAttribute()` and `setAttribute()`. `prop()` works by setting the DOM property without changing the attribute. In many cases the two are interchangeable, but occasionally one is needed over the other.

To set a checkbox as checked:

```
$('#tosAccept').prop('checked', true); // using attr() won't work properly here
```

To remove a property you can use the `removeProp()` method. Similarly `removeAttr()` removes attributes.

Section 4.2: Get the attribute value of a HTML element

When a single parameter is passed to the `.attr()` function it returns the value of passed attribute on the selected element.

Syntax:

```
$([selector]).attr([attribute name]);
```

Example:

HTML:

```
<a href="/home">Home</a>
```

jQuery:

```
$('a').attr('href');
```

Fetching `data` attributes:

jQuery offers `.data()` function in order to deal with data attributes. `.data` function returns the value of the data attribute on the selected element.

Syntax:

```
$([selector]).data([attribute name]);
```

Example:

Html:

```
<article data-column="3"></article>
```

jQuery:

```
$("article").data("column")
```

Note:

> jQuery's data() method will give you access to data-* attributes, BUT, it clobbers the case of the attribute name. Reference

Section 4.3: Setting value of HTML attribute

If you want to add an attribute to some element you can use the `attr(attributeName, attributeValue)` function. For example:

```
$('a').attr('title', 'Click me');
```

This example will add mouseover text `"Click me"` to all links on the page.

The same function is used to change attributes' values.

Section 4.4: Removing attribute

To remove an attribute from an element you can use the function `.removeAttr(attributeName)`. For example:

```
$('#home').removeAttr('title');
```

This will remove `title` attribute from the element with ID `home`.

Chapter 5: document-ready event

Section 5.1: What is document-ready and how should I use it?

jQuery code is often wrapped in `jQuery(function($) { ... });` so that it only runs after the DOM has finished loading.

```
<script type="text/javascript">
  jQuery(function($) {
    // this will set the div's text to "Hello".
    $("#myDiv").text("Hello");
  });
</script>

<div id="myDiv">Text</div>
```

This is important because jQuery (and JavaScript generally) cannot select a DOM element that has not been rendered to the page.

```
<script type="text/javascript">
  // no element with id="myDiv" exists at this point, so $("#myDiv") is an
  // empty selection, and this will have no effect
  $("#myDiv").text("Hello");
</script>

<div id="myDiv">Text</div>
```

Note that you can alias the jQuery namespace by passing a custom handler into the `.ready()` method. This is useful for cases when another JS library is using the same shortened $ alias as *jQuery*, which create a conflict. To avoid this conflict, you must call `$.noConflict();` - This forcing you to use only the default *jQuery* namespace (Instead of the short $ alias).
By passing a custom handler to the `.ready()` handler, you will be able to choose the alias name to use *jQuery*.

```
$.noConflict();

jQuery( document ).ready(function( $ ) {
   // Here we can use '$' as jQuery alias without it conflicting with other
   // libraries that use the same namespace
   $('body').append('<div>Hello</div>')
});

jQuery( document ).ready(function( jq ) {
   // Here we use a custom jQuery alias 'jq'
   jq('body').append('<div>Hello</div>')
});
```

Rather than simply putting your jQuery code at the bottom of the page, using the `$(document).ready` function ensures that all HTML elements have been rendered and the entire Document Object Model (DOM) is ready for JavaScript code to execute.

Section 5.2: jQuery 2.2.3 and earlier

These are all equivalent, the code inside the blocks will run when the document is ready:

```
$(function() {
  // code
```

```
});

$().ready(function() {
  // code
});

$(document).ready(function() {
  // code
});
```

Because these are equivalent the first is the recommended form, the following is a version of that with the `jQuery` keyword instead of the `$` which produce the same results:

```
jQuery(function() {
  // code
});
```

Section 5.3: jQuery 3.0

Notation

As of jQuery 3.0, only this form is recommended:

```
jQuery(function($) {
  // Run when document is ready
  // $ (first argument) will be internal reference to jQuery
  // Never rely on $ being a reference to jQuery in the global namespace
});
```

All other document-ready handlers are deprecated in jQuery 3.0.

Asynchronous

As of jQuery 3.0, the ready handler will always be called asynchronously. This means that in the code below, the log 'outside handler' will always be displayed first, regardless whether the document was ready at the point of execution.

```
$(function() {
  console.log("inside handler");
});
console.log("outside handler");
```

> > outside handler
> > inside handler

Section 5.4: Attaching events and manipulating the DOM inside ready()

Example uses of `$(document).ready()`:

1. **Attaching event handlers**
   Attach jQuery event handlers

```
$(document).ready(function() {
  $("button").click(function() {
```

```
    // Code for the click function
  });
});
```

2. **Run jQuery code after the page structure is created**

```
jQuery(function($) {
// set the value of an element.
   $("#myElement").val("Hello");
});
```

3. **Manipulate the loaded DOM structure**
   For example: hide a `div` when the page loads for the first time and show it on the click event of a button

```
$(document).ready(function() {
  $("#toggleDiv").hide();
  $("button").click(function() {
    $("#toggleDiv").show();
  });
});
```

Section 5.5: Difference between $(document).ready() and $(window).load()

`$(window).load()` was **deprecated in jQuery version 1.8 (and completely removed from jQuery 3.0)** and as such should not be used anymore. The reasons for the deprecation are noted on the jQuery page about this event

> Caveats of the load event when used with images
>
> A common challenge developers attempt to solve using the `.load()` shortcut is to execute a function when an image (or collection of images) have completely loaded. There are several known caveats with this that should be noted. These are:
>
> - It doesn't work consistently nor reliably cross-browser
> - It doesn't fire correctly in WebKit if the image `src` is set to the same `src` as before
> - It doesn't correctly bubble up the DOM tree
> - Can cease to fire for images that already live in the browser's cache

If you still wish to use `load()` it is documented below:

`$(document).ready()` waits until the full DOM is availble -- all the elements in the HTML have been parsed and are in the document. However, resources such as images may not have fully loaded at this point. If it is important to wait until all resources are loaded, `$(window).load()` **and you're aware of the significant limitations of this event** then the below can be used instead:

```
$(document).ready(function() {
  console.log($("#my_large_image").height()); // may be 0 because the image isn't available
});

$(window).load(function() {
  console.log($("#my_large_image").height()); // will be correct
});
```

Section 5.6: Difference between jQuery(fn) and executing your code before </body>

Using the document-ready event can have small performance drawbacks, with delayed execution of up to ~300ms. Sometimes the same behavior can be achieved by execution of code just before the closing `</body>` tag:

```
<body>
  <span id="greeting"></span> world!
  <script>
    $("#greeting").text("Hello");
  </script>
</body>
```

will produce similar behavior but perform sooner than as it does not wait for the document ready event trigger as it does in:

```
<head>
  <script>
    jQuery(function($) {
      $("#greeting").text("Hello");
    });
  </script>
</head>
<body>
  <span id="greeting"></span> world!
</body>
```

Emphasis on the fact that first example relies upon your knowledge of your page and placement of the script just prior to the closing `</body>` tag and specifically after the `span` tag.

Chapter 6: Events

Section 6.1: Delegated Events

Let's start with example. Here is a very simple example HTML.

Example HTML

```
<html>
    <head>
    </head>
    <body>
        <ul>
            <li>
                <a href="some_url/">Link 1</a>
            </li>
            <li>
                <a href="some_url/">Link 2</a>
            </li>
            <li>
                <a href="some_url/">Link 3</a>
            </li>
        </ul>
    </body>
</html>
```

The problem

Now in this example, we want to add an event listener to all `<a>` elements. The problem is that the list in this example is dynamic. `<li>` elements are added and removed as time passes by. However, the page does not refresh between changes, which would allow us to use simple click event listeners to the link objects (i.e. `$('a').click()`).

The problem we have is how to add events to the `<a>` elements that come and go.

Background information - Event propagation

Delegated events are only possible because of event propagation (often called event bubbling). Any time an event is fired, it will bubble all the way up (to the document root). They *delegate* the handling of an event to a non-changing ancestor element, hence the name "delegated" events.

So in example above, clicking `<a>` element link will trigger 'click' event in these elements in this order:

- a
- li
- ul
- body
- html
- document root

Solution

Knowing what event bubbling does, we can catch one of the wanted events which are propagating up through our HTML.

A good place for catching it in this example is the `<ul>` element, as that element does is not dynamic:

```
$('ul').on('click', 'a', function () {
  console.log(this.href); // jQuery binds the event function to the targeted DOM element
                          // this way `this` refers to the anchor and not to the list
  // Whatever you want to do when link is clicked
});
```

In above:

- We have 'ul' which is the recipient of this event listener
- The first parameter ('click') defines which events we are trying to detect.
- The second parameter ('a') is used to declare where the event needs to *originate* from (of all child elements under this event listener's recipient, ul).
- Lastly, the third parameter is the code that is run if first and second parameters' requirements are fulfilled.

In detail how solution works

1. User clicks `<a>` element
2. That triggers click event on `<a>` element.
3. The event start bubbling up towards document root.
4. The event bubbles first to the `<li>` element and then to the `<ul>` element.
5. The event listener is run as the `<ul>` element has the event listener attached.
6. The event listener first detects the triggering event. The bubbling event is 'click' and the listener has 'click', it is a pass.
7. The listener checks tries to match the second parameter ('a') to each item in the bubble chain. As the last item in the chain is an 'a' this matches the filter and this is a pass too.
8. The code in third parameter is run using the matched item as it's `this`. If the function does not include a call to `stopPropagation()`, the event will continue propagating upwards towards the root (`document`).

Note: If a suitable non-changing ancestor is not available/convenient, you should use `document`. As a habit do not use `'body'` for the following reasons:

- `body` has a bug, to do with styling, that can mean mouse events do not bubble to it. This is browser dependant and can happen when the calculated body height is 0 (e.g. when all child elements have absolute positions). Mouse events always bubble to `document`.
- `document` *always* exists to your script, so you can attach delegated handlers to `document` outside of a *DOM-ready handler* and be certain they will still work.

Section 6.2: Attach and Detach Event Handlers

Attach an Event Handler

Since version 1.7 jQuery has the event API `.on()`. This way any standard javascript event or custom event can be bound on the currently selected jQuery element. There are shortcuts such as `.click()`, but `.on()` gives you more options.

HTML

```
<button id="foo">bar</button>
```

jQuery

```
$( "#foo" ).on( "click", function() {
  console.log( $( this ).text() ); //bar
```

```
});
```

Detach an Event Handler

Naturally you have the possibility to detach events from your jQuery objects too. You do so by using `.off( events [, selector ] [, handler ] )`.

HTML

```
<button id="hello">hello</button>
```

jQuery

```
$('#hello').on('click', function(){
    console.log('hello world!');
    $(this).off();
});
```

When clicking the button `$(this)` will refer to the current jQuery object and will remove all attached event handlers from it. You can also specify which event handler should be removed.

jQuery

```
$('#hello').on('click', function(){
    console.log('hello world!');
    $(this).off('click');
});

$('#hello').on('mouseenter', function(){
    console.log('you are about to click');
});
```

In this case the `mouseenter` event will still function after clicking.

Section 6.3: Switching specific events on and off via jQuery. (Named Listeners)

Sometimes you want to switch off all previously registered listeners.

```
//Adding a normal click handler
$(document).on("click",function(){
    console.log("Document Clicked 1")
});
//Adding another click handler
$(document).on("click",function(){
    console.log("Document Clicked 2")
});
//Removing all registered handlers.
$(document).off("click")
```

An issue with this method is that ALL listeners binded on `document` by other plugins etc would also be removed.

More often than not, we want to detach all listeners attached only by us.

To achieve this, we can bind named listeners as,

```
//Add named event listener.
```

```
$(document).on("click.mymodule",function(){
    console.log("Document Clicked 1")
});
$(document).on("click.mymodule",function(){
    console.log("Document Clicked 2")
});

//Remove named event listener.
$(document).off("click.mymodule");
```

This ensures that any other click listener is not inadvertently modified.

Section 6.4: originalEvent

Sometimes there will be properties that aren't available in jQuery event. To access the underlying properties use `Event.originalEvent`

Get Scroll Direction

```
$(document).on("wheel",function(e){
    console.log(e.originalEvent.deltaY)
    // Returns a value between -100 and 100 depending on the direction you are scrolling
})
```

Section 6.5: Events for repeating elements without using ID's

Problem

There is a series of repeating elements in page that you need to know which one an event occurred on to do something with that specific instance.

Solution

- Give all common elements a common class
- Apply event listener to a class. `this` inside event handler is the matching selector element the event occurred on
- Traverse to outer most repeating container for that instance by starting at `this`
- Use `find()` within that container to isolate other elements specific to that instance

HTML

```
<div class="item-wrapper" data-item_id="346">
   <div class="item"><span class="person">Fred</span></div>
   <div class="item-toolbar">
      <button class="delete">Delete</button>
   </div>
</div>
<div class="item-wrapper" data-item_id="393">
   <div clss="item"><span class="person">Wilma</span></div>
   <div class="item-toolbar">
      <button class="delete">Delete</button>
   </div>
</div>
```

jQuery

```
$(function() {
  $('.delete').on('click', function() {
```

```
    // "this" is element event occurred on
    var $btn = $(this);
    // traverse to wrapper container
    var $itemWrap = $btn.closest('.item-wrapper');
    // look within wrapper to get person for this button instance
    var person = $itemWrap.find('.person').text();
    // send delete to server and remove from page on success of ajax
    $.post('url/string', { id: $itemWrap.data('item_id')}).done(function(response) {
      $itemWrap.remove()
    }).fail(function() {
      alert('Ooops, not deleted at server');
    });
  });
});
```

Section 6.6: Document Loading Event .load()

If you want your script to wait until a certain resource was loaded, such as an image or a PDF you can use `.load()`, which is a shortcut for shortcut for `.on( "load", handler)`.

HTML

```
<img src="image.jpeg" alt="image" id="image">
```

jQuery

```
$( "#image" ).load(function() {
  // run script
});
```

Chapter 7: DOM Manipulation

Section 7.1: Creating DOM elements

The `jQuery` function (usually aliased as $) can be used both to select elements and to create new elements.

```
var myLink = $('<a href="http://stackexchange.com"></a>');
```

You can optionally pass a second argument with element attributes:

```
var myLink = $('<a>', { 'href': 'http://stackexchange.com' });
```

`'<a>'` --> The first argument specifies the type of DOM element you want to create. In this example it's an anchor but could be anything on this list. See the specification for a reference of the `a` element.

`{ 'href': 'http://stackexchange.com' }` --> the second argument is a JavaScript Object containing attribute name/value pairs.

the 'name':'value' pairs will appear between the `< >` of the first argument, for example `<a name:value>` which for our example would be `<a href="http://stackexchange.com"></a>`

Section 7.2: Manipulating element classes

Assuming the page includes an HTML element like:

```
<p class="small-paragraph">
  This is a small <a href="https://en.wikipedia.org/wiki/Paragraph">paragraph</a>
  with a <a class="trusted" href="http://stackexchange.com">link</a> inside.
</p>
```

jQuery provides useful functions to manipulate DOM classes, most notably `hasClass()`, `addClass()`, `removeClass()` and `toggleClass()`. These functions directly modify the **`class`** attribute of the matched elements.

```
$('p').hasClass('small-paragraph'); // true
$('p').hasClass('large-paragraph'); // false

// Add a class to all links within paragraphs
$('p a').addClass('untrusted-link-in-paragraph');

// Remove the class from a.trusted
$('a.trusted.untrusted-link-in-paragraph')
.removeClass('untrusted-link-in-paragraph')
.addClass('trusted-link-in-paragraph');
```

Toggle a class

Given the example markup, we can add a class with our first `.toggleClass()`:

```
$(".small-paragraph").toggleClass("pretty");
```

Now this would return **`true`**: `$(".small-paragraph").hasClass("pretty")`

`toggleClass` provides the same effect with less code as:

```
if($(".small-paragraph").hasClass("pretty")){
```

```
   $(".small-paragraph").removeClass("pretty");}
else {
   $(".small-paragraph").addClass("pretty"); }
```

toggle Two classes:

```
$(".small-paragraph").toggleClass("pretty cool");
```

Boolean to add/remove classes:

```
$(".small-paragraph").toggleClass("pretty",true); //cannot be truthy/falsey

$(".small-paragraph").toggleClass("pretty",false);
```

Function for class toggle (see example further down to avoid an issue)

```
$( "div.surface" ).toggleClass(function() {
  if ( $( this ).parent().is( ".water" ) ) {
    return "wet";
  } else {
    return "dry";
  }
});
```

Used in examples:

```
// functions to use in examples
function stringContains(myString, mySubString) {
  return myString.indexOf(mySubString) !== -1;
}
function isOdd(num) { return num % 2;}
var showClass = true; //we want to add the class
```

Examples:

Use the element index to toggle classes odd/even

```
$( "div.gridrow" ).toggleClass(function(index,oldClasses, false), showClass ) {
  showClass
  if ( isOdd(index) ) {
    return "wet";
  } else {
    return "dry";
  }
});
```

More complex `toggleClass` example, given a simple grid markup

```
<div class="grid">
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow">row</div>
  <div class="gridrow gridfooter">row but I am footer!</div>
</div>
```

Simple functions for our examples:

```
function isOdd(num) {
  return num % 2;
}

function stringContains(myString, mySubString) {
  return myString.indexOf(mySubString) !== -1;
}
var showClass = true; //we want to add the class
```

Add an odd/even class to elements with a `gridrow` class

```
$("div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {
  if (isOdd(index)) {
    return "odd";
  } else {
    return "even";
  }
  return oldClasses;
}, showClass);
```

If the row has a `gridfooter` class, remove the odd/even classes, keep the rest.

```
$("div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {
  var isFooter = stringContains(oldClasses, "gridfooter");
  if (isFooter) {
    oldClasses = oldClasses.replace('even', ' ').replace('odd', ' ');
    $(this).toggleClass("even odd", false);
  }
  return oldClasses;
}, showClass);
```

The classes that get returned are what is effected. Here, if an element does not have a `gridfooter`, add a class for even/odd. This example illustrates the return of the OLD class list. If this `else return oldClasses;` is removed, only the new classes get added, thus the row with a `gridfooter` class would have all classes removed had we not returned those old ones - they would have been toggled (removed) otherwise.

```
$("div.gridrow").toggleClass(function(index, oldClasses, showThisClass) {
  var isFooter = stringContains(oldClasses, "gridfooter");
  if (!isFooter) {
    if (isOdd(index)) {
      return "oddLight";
    } else {
      return "evenLight";
    }
  } else return oldClasses;
}, showClass);
```

Section 7.3: Other API Methods

jQuery offers a variety of methods that can be used for DOM manipulation.

The first is the .empty() method.

Imagine the following markup:

```
<div id="content">
  <div>Some text</div>
```

```
</div>
```

By calling `$('#content').empty();`, the inner div would be removed. This could also be achieved by using `$('#content').html('');`.

Another handy function is the .closest() function:

```
<tr id="row_1">
  <td><button type="button" class="delete">Delete</button>
</tr>
```

If you wanted to find the closest row to a button that was clicked within one of the row cells then you could do this:

```
$('.delete').click(function() {
  $(this).closest('tr');
});
```

Since there will probably be multiple rows, each with their own **`delete`** buttons, we use `$(this)` within the .click() function to limit the scope to the button we actually clicked.

If you wanted to get the `id` of the row containing the **`Delete`** button that you clicked, you could so something like this:

```
$('.delete').click(function() {
  var $row = $(this).closest('tr');
  var id = $row.attr('id');
});
```

It is usually considered good practise to prefix variables containing jQuery objects with a $ (dollar sign) to make it clear what the variable is.

An alternative to `.closest()` is the .parents() method:

```
$('.delete').click(function() {
  var $row = $(this).parents('tr');
  var id = $row.attr('id');
});
```

and there is also a .parent() function as well:

```
$('.delete').click(function() {
  var $row = $(this).parent().parent();
  var id = $row.attr('id');
});
```

`.parent()` only goes up one level of the DOM tree so it is quite inflexible, if you were to change the delete button to be contained within a `span` for example, then the jQuery selector would be broken.

Chapter 8: DOM Traversing

Section 8.1: Select children of element

To select the children of an element you can use the `children()` method.

```
<div class="parent">
    <h2>A headline</h2>
    <p>Lorem ipsum dolor sit amet...</p>
    <p>Praesent quis dolor turpis...</p>
</div>
```

Change the color of *all* the children of the `.parent` element:

```
$('.parent').children().css("color", "green");
```

The method accepts an optional `selector` argument that can be used to filter the elements that are returned.

```
// Only get "p" children
$('.parent').children("p").css("color", "green");
```

Section 8.2: Get next element

To get the next element you can use the `.next()` method.

```
<ul>
    <li>Mark</li>
    <li class="anna">Anna</li>
    <li>Paul</li>
</ul>
```

If you are standing on the "Anna" element and you want to get the next element, "Paul", the `.next()` method will allow you to do that.

```
// "Paul" now has green text
$(".anna").next().css("color", "green");
```

The method takes an optional `selector` argument, which can be used if the next element must be a certain kind of element.

```
// Next element is a "li", "Paul" now has green text
$(".anna").next("li").css("color", "green");
```

If the next element is not of the type `selector` then an empty set is returned, and the modifications will not do anything.

```
// Next element is not a ".mark", nothing will be done in this case
$(".anna").next(".mark").css("color", "green");
```

Section 8.3: Get previous element

To get the previous element you can use the `.prev()` method.

```
<ul>
```

```
    <li>Mark</li>
    <li class="anna">Anna</li>
    <li>Paul</li>
</ul>
```

If you are standing on the "Anna" element and you want to get the previous element, "Mark", the `.prev()` method will allow you to do that.

```
// "Mark" now has green text
$(".anna").prev().css("color", "green");
```

The method takes an optional `selector` argument, which can be used if the previous element must be a certain kind of element.

```
// Previous element is a "li", "Mark" now has green text
$(".anna").prev("li").css("color", "green");
```

If the previous element is not of the type `selector` then an empty set is returned, and the modifications will not do anything.

```
// Previous element is not a ".paul", nothing will be done in this case
$(".anna").prev(".paul").css("color", "green");
```

Section 8.4: Filter a selection

To filter a selection you can use the `.filter()` method.

The method is called on a selection and returns a new selection. If the filter matches an element then it is added to the returned selection, otherwise it is ignored. If no element is matched then an empty selection is returned.

The HTML

This is the HTML we will be using.

```
<ul>
    <li class="zero">Zero</li>
    <li class="one">One</li>
    <li class="two">Two</li>
    <li class="three">Three</li>
</ul>
```

Selector

Filtering using selectors is one of the simpler ways to filter a selection.

```
$("li").filter(":even").css("color", "green"); // Color even elements green
$("li").filter(".one").css("font-weight", "bold"); // Make ".one" bold
```

Function

Filtering a selection using a function is useful if it is not possible to use selectors.

The function is called for each element in the selection. If it returns a **`true`** value then the element will be added to the returned selection.

```
var selection = $("li").filter(function (index, element) {
    // "index" is the position of the element
    // "element" is the same as "this"
    return $(this).hasClass("two");
});
selection.css("color", "green"); // ".two" will be colored green
```

Elements

You can filter by DOM elements. If the DOM elements are in the selection then they will be included in the returned selection.

```
var three = document.getElementsByClassName("three");
$("li").filter(three).css("color", "green");
```

Selection

You can also filter a selection by another selection. If an element is in both selections then it will be included in the returned selection.

```
var elems = $(".one, .three");
$("li").filter(elems).css("color", "green");
```

Section 8.5: find() method

.find() method allows us to search through the descendants of these elements in the DOM tree and construct a new jQuery object from the matching elements.

HTML

```
    <div class="parent">
       <div class="children" name="first">
           <ul>
               <li>A1</li>
               <li>A2</li>
               <li>A3</li>
           </ul>
       </div>
       <div class="children" name="second">
           <ul>
               <li>B1</li>
               <li>B2</li>
               <li>B3</li>
           </ul>
       </div>
      </div>
```

jQuery

```
$('.parent').find('.children[name="second"] ul li').css('font-weight','bold');
```

Output

- A1
- A2
- A3

- **B1**
- **B2**
- **B3**

Section 8.6: Iterating over list of jQuery elements

When you need to iterate over the list of jQuery elements.

Consider this DOM structure:

```
<div class="container">
    <div class="red one">RED 1 Info</div>
    <div class="red two">RED 2 Info</div>
    <div class="red three">RED 3 Info</div>
</div>
```

To print the text present in all the `div` elements with a class of `red`:

```
$(".red").each(function(key, ele){
    var text = $(ele).text();
    console.log(text);
});
```

Tip: `key` is the index of the `div.red` element we're currently iterating over, within its parent. `ele` is the HTML element, so we can create a jQuery object from it using `$()` or `jQuery()`, like so: `$(ele)`. After, we can call any jQuery method on the object, like `css()` or `hide()` etc. In this example, we just pull the text of the object.

Section 8.7: Selecting siblings

To select siblings of an item you can use the `.siblings()` method.

A typical example where you want to modify the siblings of an item is in a menu:

```
<ul class="menu">
    <li class="selected">Home</li>
    <li>Blog</li>
    <li>About</li>
</ul>
```

When the user clicks on a menu item the `selected` class should be added to the clicked element and removed from its *siblings*:

```
$(".menu").on("click", "li", function () {
    $(this).addClass("selected");
    $(this).siblings().removeClass("selected");
});
```

The method takes an optional `selector` argument, which can be used if you need to narrow down the kinds of siblings you want to select:

```
$(this).siblings("li").removeClass("selected");
```

Section 8.8: closest() method

Returns the first element that matches the selector starting at the element and traversing up the DOM tree.

HTML

```
<div id="abc" class="row">
    <div id="xyz" class="row">
    </div>
    <p id="origin">
      Hello
    </p>
</div>
```

jQuery

```
var target = $('#origin').closest('.row');
console.log("Closest row:", target.attr('id') );

var target2 = $('#origin').closest('p');
console.log("Closest p:", target2.attr('id') );
```

Output

```
"Closest row: abc"
"Closest p: origin"
```

first() method : The first method returns the first element from the matched set of elements.

HTML

```
<div class='.firstExample'>
   <p>This is first paragraph in a div.</p>
   <p>This is second paragraph in a div.</p>
   <p>This is third paragraph in a div.</p>
   <p>This is fourth paragraph in a div.</p>
   <p>This is fifth paragraph in a div.</p>
</div>
```

JQuery

```
var firstParagraph = $("div p").first();
console.log("First paragraph:", firstParagraph.text());
```

Output:

```
First paragraph: This is first paragraph in a div.
```

Chapter 9: CSS Manipulation

Section 9.1: CSS – Getters and Setters

CSS Getter

The `.css()` **getter** function can be applied to every DOM element on the page like the following:

```
// Rendered width in px as a string. ex: `150px`
// Notice the `as a string` designation - if you require a true integer,
// refer to `$.width()` method
$("body").css("width");
```

This line will return the **computed width** of the specified element, each CSS property you provide in the parentheses will yield the value of the property for this `$("selector")` DOM element, if you ask for CSS attribute that doesn't exist you will get `undefined` as a response.

You also can call the **CSS getter** with an array of attributes:

```
$("body").css(["animation","width"]);
```

this will return an object of all the attributes with their values:

```
Object {animation: "none 0s ease 0s 1 normal none running", width: "529px"}
```

CSS Setter

The `.css()` **setter** method can also be applied to every DOM element on the page.

```
$("selector").css("width", 500);
```

This statement set the `width` of the `$("selector")` to `500px` and return the jQuery object so you can chain more methods to the specified selector.

The `.css()` **setter** can also be used passing an Object of CSS properties and values like:

```
$("body").css({"height": "100px", width:100, "padding-top":40, paddingBottom:"2em"});
```

All the changes the setter made are appended to the DOM element `style` property thus affecting the elements' styles (unless that style property value is already defined as `!important` somewhere else in styles).

Section 9.2: Increment/Decrement Numeric Properties

Numeric CSS properties can be incremented and decremented with the `+=` and `-=` syntax, respectively, using the `.css()` method:

```
// Increment using the += syntax
$("#target-element").css("font-size", "+=10");

// You can also specify the unit to increment by
$("#target-element").css("width", "+=100pt");
$("#target-element").css("top", "+=30px");
$("#target-element").css("left", "+=3em");
```

```
// Decrementing is done by using the -= syntax
$("#target-element").css("height", "-=50pt");
```

Section 9.3: Set CSS property

Setting only one style:

```
$('#target-element').css('color', '#000000');
```

Setting multiple styles at the same time:

```
$('#target-element').css({
    'color': '#000000',
    'font-size': '12pt',
    'float': 'left',
});
```

Section 9.4: Get CSS property

To get an element's CSS property you can use the `.css(propertyName)` method:

```
var color    = $('#element').css('color');
var fontSize = $('#element').css('font-size');
```

Chapter 10: Element Visibility

Parameter	Details
Duration	When passed, the effects of `.hide()`, `.show()` and `.toggle()` are animated; the element(s) will gradually fade in or out.

Section 10.1: Overview

```
$(element).hide()            // sets display: none
$(element).show()            // sets display to original value
$(element).toggle()          // toggles between the two
$(element).is(':visible')    // returns true or false
$('element:visible')         // matches all elements that are visible
$('element:hidden')          // matches all elements that are hidden

$('element').fadeIn();          // display the element
$('element').fadeOut();          // hide the element

$('element').fadeIn(1000);          // display the element using timer
$('element').fadeOut(1000);          // hide the element using timer

// display the element using timer and a callback function
$('element').fadeIn(1000, function(){
 // code to execute
});

// hide the element using timer and a callback function
$('element').fadeOut(1000, function(){
   // code to execute
});
```

Section 10.2: Toggle possibilities

Simple `toggle()` case

```
function toggleBasic() {
  $(".target1").toggle();
}
```

With specific *duration*

```
function toggleDuration() {
  $(".target2").toggle("slow"); // A millisecond duration value is also acceptable
}
```

...and *callback*

```
function toggleCallback() {
  $(".target3").toggle("slow",function(){alert('now do something');});
}
```

...or with *easing* and callback.

```
function toggleEasingAndCallback() {
  // You may use jQueryUI as the core only supports linear and swing easings
  $(".target4").toggle("slow","linear",function(){alert('now do something');});
```

```
}
```

...or with a variety of *options*.

```
function toggleWithOptions() {
  $(".target5").toggle(
    { // See all possible options in: api.jquery.com/toggle/#toggle-options
      duration:1000, // milliseconds
      easing:"linear",
      done:function(){
        alert('now do something');
      }
    }
  );
}
```

It's also possible to use a *slide* as animation with `slideToggle()`

```
function toggleSlide() {
  $(".target6").slideToggle(); // Animates from top to bottom, instead of top corner
}
```

...or fade in/out by changing opacity with `fadeToggle()`

```
function toggleFading() {
  $( ".target7" ).fadeToggle("slow")
}
```

...or toggle a class with `toggleClass()`

```
function toggleClass() {
  $(".target8").toggleClass('active');
}
```

A common case is to use `toggle()` in order to show one element while hiding the other (same class)

```
function toggleX() {
  $(".targetX").toggle("slow");
}
```

All the above examples can be found here

Chapter 11: Append

Parameters	Details
content	Possible types: Element, HTML string, text, array, object or even a function returning a string.

Section 11.1: Efficient consecutive .append() usage

Starting out:

HTML

```
<table id='my-table' width='960' height='500'></table>
```

JS

```
var data = [
    { type: "Name", content: "John Doe" },
    { type: "Birthdate", content: "01/01/1970" },
    { type: "Salary", content: "$40,000,000" },
    // ...300 more rows...
    { type: "Favorite Flavour", content: "Sour" }
];
```

Appending inside a loop

You just received a big array of data. Now it's time to loop through and render it on the page.

Your first thought may be to do something like this:

```
var i;                      // <- the current item number
var count = data.length;    // <- the total
var row;                    // <- for holding a reference to our row object

// Loop over the array
for ( i = 0; i < count; ++i ) {
    row = data[ i ];

    // Put the whole row into your table
    $('#my-table').append(
        $('<tr></tr>').append(
            $('<td></td>').html(row.type),
            $('<td></td>').html(row.content)
        )
    );
}
```

This is *perfectly valid* and will render exactly what you'd expect, but...

DO NOT do this.

Remember those **300+** rows of data?

Each one will force the browser to re-calculate every element's width, height and positioning values, along with any other styles - unless they are separated by a layout boundary, which unfortunately for this example (as they are descendants of a `<table>` element), they cannot.

At small amounts and few columns, this performance penalty will certainly be negligible. But we want every millisecond to count.

Better options

1. **Add to a separate array, append after loop completes**

```
/**
 * Repeated DOM traversal (following the tree of elements down until you reach
 * what you're looking for - like our <table>) should also be avoided wherever possible.
 */

// Keep the table cached in a variable then use it until you think it's been removed
var $myTable = $('#my-table');

// To hold our new <tr> jQuery objects
var rowElements = [];

var count = data.length;
var i;
var row;

// Loop over the array
for ( i = 0; i < count; ++i ) {
    rowElements.push(
        $('<tr></tr>').append(
            $('<td></td>').html(row.type),
            $('<td></td>').html(row.content)
        )
    );
}

// Finally, insert ALL rows at once
$myTable.append(rowElements);
```

Out of these options, this one relies on jQuery the most.

2. **Using modern Array.* methods**

```
var $myTable = $('#my-table');

// Looping with the .map() method
//  - This will give us a brand new array based on the result of our callback function
var rowElements = data.map(function ( row ) {

    // Create a row
    var $row = $('<tr></tr>');

    // Create the columns
    var $type = $('<td></td>').html(row.type);
    var $content = $('<td></td>').html(row.content);

    // Add the columns to the row
    $row.append($type, $content);

    // Add to the newly-generated array
    return $row;
});

// Finally, put ALL of the rows into your table
$myTable.append(rowElements);
```

Functionally equivalent to the one before it, only easier to read.

3. **Using strings of HTML (instead of jQuery built-in methods)**

```
// ...
var rowElements = data.map(function ( row ) {
    var rowHTML = '<tr><td>';
    rowHTML += row.type;
    rowHTML += '</td><td>';
    rowHTML += row.content;
    rowHTML += '</td></tr>';
    return rowHTML;
});

// Using .join('') here combines all the separate strings into one
$myTable.append(rowElements.join(''));
```

Perfectly valid but again, **not recommended**. This forces jQuery to parse a very large amount of text at once and is not necessary. jQuery is very good at what it does when used correctly.

4. **Manually create elements, append to document fragment**

```
var $myTable = $(document.getElementById('my-table'));

/**
 * Create a document fragment to hold our columns
 * - after appending this to each row, it empties itself
 *   so we can re-use it in the next iteration.
 */
var colFragment = document.createDocumentFragment();

/**
 * Loop over the array using .reduce() this time.
 * We get a nice, tidy output without any side-effects.
 *  - In this example, the result will be a
 *    document fragment holding all the <tr> elements.
 */
var rowFragment = data.reduce(function ( fragment, row ) {

    // Create a row
    var rowEl = document.createElement('tr');

    // Create the columns and the inner text nodes
    var typeEl = document.createElement('td');
    var typeText = document.createTextNode(row.type);
    typeEl.appendChild(typeText);

    var contentEl = document.createElement('td');
    var contentText = document.createTextNode(row.content);
    contentEl.appendChild(contentText);

    // Add the columns to the column fragment
    // - this would be useful if columns were iterated over separately
    //   but in this example it's just for show and tell.
    colFragment.appendChild(typeEl);
    colFragment.appendChild(contentEl);
```

```
    rowEl.appendChild(colFragment);

    // Add rowEl to fragment - this acts as a temporary buffer to
    // accumulate multiple DOM nodes before bulk insertion
    fragment.appendChild(rowEl);

    return fragment;
}, document.createDocumentFragment());

// Now dump the whole fragment into your table
$myTable.append(rowFragment);
```

My personal favorite. This illustrates a general idea of what jQuery does at a lower level.

Dive deeper

- jQuery source viewer
- Array.prototype.join()
- Array.prototype.map()
- Array.prototype.reduce()
- document.createDocumentFragment()
- document.createTextNode()
- Google Web Fundamentals - Performance

Section 11.2: jQuery append

HTML

```
<p>This is a nice </p>
<p>I like </p>

<ul>
  <li>List item 1</li>
  <li>List item 2</li>
  <li>List item 3</li>
</ul>

<button id="btn-1">Append text</button>
<button id="btn-2">Append list item</button>
```

Script

```
$("#btn-1").click(function(){
    $("p").append(" <b>Book</b>.");
});
$("#btn-2").click(function(){
    $("ul").append("<li>Appended list item</li>");
 });
});
```

Section 11.3: Appending an element to a container

Solution 1:

```
$('#parent').append($('#child'));
```

Solution 2:

```
$('#child').appendTo($('#parent'));
```

Both solutions are appending the element `#child` (adding at the end) to the element `#parent`.

Before:

```
<div id="parent">
  <span>other content</span>
</div>
<div id="child">

</div>
```

After:

```
<div id="parent">
  <span>other content</span>
  <div id="child">

  </div>
</div>
```

Note: When you append content that already exsists in the document, this content will be removed from its original parent container and appended to the new parent container. So you can't use `.append()` or `.appendTo()` to clone an element. If you need a clone use `.clone()` -> [http://api.jquery.com/clone/][1]

Chapter 12: Prepend

Section 12.1: Prepending an element to a container

Solution 1:

```
$('#parent').prepend($('#child'));
```

Solution 2:

```
$('#child').prependTo($('#parent'));
```

Both solutions are prepending the element `#child` (adding at the beginning) to the element `#parent`.

Before:

```
<div id="parent">
  <span>other content</span>
</div>
<div id="child">

</div>
```

After:

```
<div id="parent">
  <div id="child">

  </div>
  <span>other content</span>
</div>
```

Section 12.2: Prepend method

`prepend()` - Insert content, specified by the parameter, to the beginning of each element in the set of matched elements.

1. `prepend( content [, content ] )`

```
// with html string
jQuery('#parent').prepend('<span>child</span>');
// or you can use jQuery object
jQuery('#parent').prepend($('#child'));
// or you can use comma separated multiple elements to prepend
jQuery('#parent').prepend($('#child1'),$('#child2'));
```

2. `prepend(function)`

JQuery *version: 1.4* onwards you can use callback function as the argument. Where you can get arguments as index position of the element in the set and the old HTML value of the element. Within the function, **this** refers to the current element in the set.

```
jQuery('#parent').prepend(function(i,oldHTML){
     // return the value to be prepend
     return  '<span>child</span>';
```

```
});
```

Chapter 13: Getting and setting width and height of an element

Section 13.1: Getting and setting width and height (ignoring border)

Get width and height:

```
var width = $('#target-element').width();
var height = $('#target-element').height();
```

Set width and height:

```
$('#target-element').width(50);
$('#target-element').height(100);
```

Section 13.2: Getting and setting innerWidth and innerHeight (ignoring padding and border)

Get width and height:

```
var width = $('#target-element').innerWidth();
var height = $('#target-element').innerHeight();
```

Set width and height:

```
$('#target-element').innerWidth(50);
$('#target-element').innerHeight(100);
```

Section 13.3: Getting and setting outerWidth and outerHeight (including padding and border)

Get width and height (excluding margin):

```
var width = $('#target-element').outerWidth();
var height = $('#target-element').outerHeight();
```

Get width and height (including margin):

```
var width = $('#target-element').outerWidth(true);
var height = $('#target-element').outerHeight(true);
```

Set width and height:

```
$('#target-element').outerWidth(50);
$('#target-element').outerHeight(100);
```

Chapter 14: jQuery .animate() Method

Parameter	Details
properties	An object of CSS properties and values that the animation will move toward
duration	(default: 400) A string or number determining how long the animation will run
easing	(default: swing) A string indicating which easing function to use for the transition
complete	A function to call once the animation is complete, called once per matched element.
start	specifies a function to be executed when the animation begins.
step	specifies a function to be executed for each step in the animation.
queue	a Boolean value specifying whether or not to place the animation in the effects queue.
progress	specifies a function to be executed after each step in the animation.
done	specifies a function to be executed when the animation ends.
fail	specifies a function to be executed if the animation fails to complete.
specialEasing	a map of one or more CSS properties from the styles parameter, and their corresponding easing functions.
always	specifies a function to be executed if the animation stops without completing.

Section 14.1: Animation with callback

Sometimes we need to change words position from one place to another or reduce size of the words and change the color of words automatically to improve the attraction of our website or web apps. JQuery helps a lot with this concept using `fadeIn()`, `hide()`, `slideDown()` but its functionality are limited and it only done the specific task which assign to it.

Jquery fix this problem by providing an amazing and flexible method called `.animate()`. This method allows to set custom animations which is used css properties that give permission to fly over borders. for example if we give css style property as `width:200;` and current position of the DOM element is 50, animate method reduce current position value from given css value and animate that element to 150.But we don't need to bother about this part because animation engine will handle it.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
<script>
    $("#btn1").click(function(){
        $("#box").animate({width: "200px"});
    });
</script>

<button id="btn1">Animate Width</button>
<div id="box" style="background:#98bf21;height:100px;width:100px;margin:6px;"></div>
```

List of css style properties that allow in `.animate()` method.

```
backgroundPositionX, backgroundPositionY, borderWidth, borderBottomWidth, borderLeftWidth,
borderRightWidth, borderTopWidth, borderSpacing, margin, marginBottom, marginLeft, marginRight,
marginTop, outlineWidth, padding, paddingBottom, paddingLeft, paddingRight, paddingTop, height,
width, maxHeight, maxWidth, minHeight, minWidth, fontSize, bottom, left, right, top, letterSpacing,
 wordSpacing, lineHeight, textIndent,
```

Speed specified in `.animate()` method.

```
milliseconds (Ex: 100, 1000, 5000, etc.),
"slow",
```

```
"fast"
```

Easing specified in `.animate()` method.

"swing"
"linear"

Here is some examples with complex animation options.

Eg 1:

```
$( "#book" ).animate({
  width: [ "toggle", "swing" ],
  height: [ "toggle", "swing" ],
  opacity: "toggle"
 }, 5000, "linear", function() {
    $( this ).after( "<div>Animation complete.</div>" );
});
```

Eg 2:

```
  $("#box").animate({
     height: "300px",
     width: "300px"
     }, {
     duration: 5000,
     easing: "linear",
     complete: function(){
        $(this).after("<p>Animation is complete!</p>");
     }
  });
```

Chapter 15: jQuery Deferred objects and Promises

jQuery promises are a clever way of chaining together asynchronous operations in a building-block manner. This replaces old-school nesting of callbacks, which are not so easily reorganised.

Section 15.1: jQuery ajax() success, error VS .done(), .fail()

success and Error : A **success** callback that gets invoked upon successful completion of an Ajax request.

A **failure** callback that gets invoked in case there is any error while making the request.

Example:

```
$.ajax({
        url: 'URL',
        type: 'POST',
        data: yourData,
        datatype: 'json',
        success: function (data) { successFunction(data); },
        error: function (jqXHR, textStatus, errorThrown) { errorFunction(); }
    });
```

.done() and .fail() :

.ajax().done(function(data, textStatus, jqXHR){}); Replaces method .success() which was deprecated in jQuery 1.8.This is an alternative construct for the success callback function above.

.ajax().fail(function(jqXHR, textStatus, errorThrown){}); Replaces method .error() which was deprecated in jQuery 1.8.This is an alternative construct for the complete callback function above.

Example:

```
$.ajax({
    url: 'URL',
    type: 'POST',
    data: yourData,
    datatype: 'json'
})
.done(function (data) { successFunction(data); })
.fail(function (jqXHR, textStatus, errorThrown) { serrorFunction(); });
```

Section 15.2: Basic promise creation

Here is a very simple example of a function that "*promises* to proceed when a given time elapses". It does that by creating a new `Deferred` object, that is resolved later and returning the Deferred's promise:

```
function waitPromise(milliseconds){

   // Create a new Deferred object using the jQuery static method
   var def = $.Deferred();

   // Do some asynchronous work - in this case a simple timer
   setTimeout(function(){
```

```
        // Work completed... resolve the deferred, so it's promise will proceed
        def.resolve();
    }, milliseconds);

    // Immediately return a "promise to proceed when the wait time ends"
    return def.promise();
}
```

And use like this:

```
waitPromise(2000).then(function(){
     console.log("I have waited long enough");
});
```

Chapter 16: Ajax

Parameter	Details
url	Specifies the URL to which the request will be sent
settings	an object containing numerous values that affect the behavior of the request
type	The HTTP method to be used for the request
data	Data to be sent by the request
success	A callback function to be called if the request succeeds
error	A callback to handle error
statusCode	An object of numeric HTTP codes and functions to be called when the response has the corresponding code
dataType	The type of data that you're expecting back from the server
contentType	Content type of the data to sent to the server. Default is "application/x-www-form-urlencoded; charset=UTF-8"
context	Specifies the context to be used inside callbacks, usually `this` which refers to the current target.

Section 16.1: Handling HTTP Response Codes with $.ajax()

In addition to `.done`, `.fail` and `.always` promise callbacks, which are triggered based on whether the request was successful or not, there is the option to trigger a function when a specific HTTP Status Code is returned from the server. This can be done using the `statusCode` parameter.

```
$.ajax({
    type: {POST or GET or PUT etc.},
    url:  {server.url},
    data: {someData: true},
    statusCode: {
        404: function(responseObject, textStatus, jqXHR) {
            // No content found (404)
            // This code will be executed if the server returns a 404 response
        },
        503: function(responseObject, textStatus, errorThrown) {
            // Service Unavailable (503)
            // This code will be executed if the server returns a 503 response
        }
    }
})
.done(function(data){
    alert(data);
})
.fail(function(jqXHR, textStatus){
    alert('Something went wrong: ' + textStatus);
})
.always(function(jqXHR, textStatus) {
   alert('Ajax request was finished')
});
```

As official jQuery documentation states:

> If the request is successful, the status code functions take the same parameters as the success callback; if it results in an error (including 3xx redirect), they take the same parameters as the `error` callback.

Section 16.2: Using Ajax to Submit a Form

Sometimes you may have a form and want to submit it using ajax.

Suppose you have this simple form -

```
<form id="ajax_form" action="form_action.php">
  <label for="name">Name :</label>
  <input name="name" id="name" type="text" />
  <label for="name">Email :</label>
  <input name="email" id="email" type="text" />
  <input type="submit" value="Submit" />
</form>
```

The following jQuery code can be used (within a `$(document).ready` call) -

```
$('#ajax_form').submit(function(event){
  event.preventDefault();
  var $form = $(this);

  $.ajax({
      type: 'POST',
      url: $form.attr('action'),
      data: $form.serialize(),
      success: function(data) {
        // Do something with the response
      },
      error: function(error) {
        // Do something with the error
      }
  });
});
```

Explanation

- `var $form = $(this)` - the form, cached for reuse
- `$('#ajax_form').submit(function(event){` - When the form with ID "ajax_form" is submitted run this function and pass the event as a parameter.
- `event.preventDefault();` - Prevent the form from submitting normally (Alternatively we can use `return false` after the `ajax({});` statement, which will have the same effect)
- `url: $form.attr('action'),` - Get the value of the form's "action" attribute and use it for the "url" property.
- `data: $form.serialize(),` - Converts the inputs within the form into a string suitable for sending to the server. In this case it will return something like "name=Bob&email=bob@bobsemailaddress.com"

Section 16.3: All in one examples

Ajax Get:

Solution 1:

```
$.get('url.html', function(data){
    $('#update-box').html(data);
});
```

Solution 2:

```
$.ajax({
    type: 'GET',
    url: 'url.php',
}).done(function(data){
    $('#update-box').html(data);
}).fail(function(jqXHR, textStatus){
    alert('Error occurred: ' + textStatus);
});
```

Ajax Load: Another ajax get method created for simplcity

```
$('#update-box').load('url.html');
```

.load can also be called with additional data. The data part can be provided as string or object.

```
$('#update-box').load('url.php', {data: "something"});
$('#update-box').load('url.php', "data=something");
```

If .load is called with a callback method, the request to the server will be a post

```
$('#update-box').load('url.php', {data: "something"}, function(resolve){
    //do something
});
```

Ajax Post:

Solution 1:

```
$.post('url.php',
    {date1Name: data1Value, date2Name: data2Value},  //data to be posted
    function(data){
        $('#update-box').html(data);
    }
);
```

Solution 2:

```
$.ajax({
    type: 'Post',
    url: 'url.php',
    data: {date1Name: data1Value, date2Name: data2Value}  //data to be posted
}).done(function(data){
    $('#update-box').html(data);
}).fail(function(jqXHR, textStatus){
    alert('Error occurred: ' + textStatus);
});
```

Ajax Post JSON:

```
var postData = {
    Name: name,
    Address: address,
    Phone: phone
};

$.ajax({
    type: "POST",
    url: "url.php",
```

```
    dataType: "json",
    data: JSON.stringfy(postData),
    success: function (data) {
        //here variable data is in JSON format
    }
});
```

Ajax Get JSON:

Solution 1:

```
$.getJSON('url.php', function(data){
    //here variable data is in JSON format
});
```

Solution 2:

```
$.ajax({
     type: "Get",
     url: "url.php",
     dataType: "json",
     data: JSON.stringfy(postData),
     success: function (data) {
         //here variable data is in JSON format
     },
     error: function(jqXHR, textStatus){
         alert('Error occurred: ' + textStatus);
     }
 });
```

Section 16.4: Ajax File Uploads

1. A Simple Complete Example

We could use this sample code to upload the files selected by the user every time a new file selection is made.

```
<input type="file" id="file-input" multiple>
```

```
var files;
var fdata = new FormData();
$("#file-input").on("change", function (e) {
    files = this.files;

    $.each(files, function (i, file) {
        fdata.append("file" + i, file);
    });

    fdata.append("FullName", "John Doe");
    fdata.append("Gender", "Male");
    fdata.append("Age", "24");

    $.ajax({
        url: "/Test/Url",
        type: "post",
        data: fdata, //add the FormData object to the data parameter
        processData: false, //tell jquery not to process data
```

```
        contentType: false, //tell jquery not to set content-type
        success: function (response, status, jqxhr) {
            //handle success
        },
        error: function (jqxhr, status, errorMessage) {
            //handle error
        }
    });
});
```

Now let's break this down and inspect it part by part.

2. Working With File Inputs

This MDN Document (Using files from web applications) is a good read about various methods on how to handle file inputs. Some of these methods will also be used in this example.

Before we get to uploading files, we first need to give the user a way to select the files they want to upload. For this purpose we will use a `file input`. The `multiple` property allows for selecting more than one files, you can remove it if you want the user to select one file at a time.

```
<input type="file" id="file-input" multiple>
```

We will be using input's `change event` to capture the files.

```
var files;
$("#file-input").on("change", function(e){
    files = this.files;
});
```

Inside the handler function, we access the files through the files property of our input. This gives us a FileList, which is an array like object.

3. Creating and Filling the FormData

In order to upload files with Ajax we are going to use FormData.

```
var fdata = new FormData();
```

FileList we have obtained in the previous step is an array like object and can be iterated using various methods including for loop, for...of loop and jQuery.each. We will be sticking with the jQuery in this example.

```
$.each(files, function(i, file) {
  //...
});
```

We will be using the append method of FormData to add the files into our formdata object.

```
$.each(files, function(i, file) {
  fdata.append("file" + i, file);
});
```

We can also add other data we want to send the same way. Let's say we want to send some personal information we have received from the user along with the files. We could add this this information into our formdata object.

```
fdata.append("FullName", "John Doe");
```

```
fdata.append("Gender", "Male");
fdata.append("Age", "24");
//...
```

4. Sending the Files With Ajax

```
$.ajax({
    url: "/Test/Url",
    type: "post",
    data: fdata, //add the FormData object to the data parameter
    processData: false, //tell jquery not to process data
    contentType: false, //tell jquery not to set content-type
    success: function (response, status, jqxhr) {
        //handle success
    },
    error: function (jqxhr, status, errorMessage) {
        //handle error
    }
});
```

We set `processData` and `contentType` properties to **`false`**. This is done so that the files can be send to the server and be processed by the server correctly.

Chapter 17: Checkbox Select all with automatic check/uncheck on other checkbox change

I've used various Stackoverflow examples and answers to come to this really simple example on how to manage "select all" checkbox coupled with an automatic check/uncheck if any of the group checkbox status changes. Constraint: The "select all" id must match the input names to create the select all group. In the example, the input select all ID is cbGroup1. The input names are also cbGroup1

Code is very short, not plenty of if statement (time and resource consuming).

Section 17.1: 2 select all checkboxes with corresponding group checkboxes

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<p>
<input id="cbGroup1" type="checkbox">Select all
<input name="cbGroup1" type="checkbox" value="value1_1">Group1 value 1
<input name="cbGroup1" type="checkbox" value="value1_2">Group1 value 2
<input name="cbGroup1" type="checkbox" value="value1_3">Group1 value 3
</p>
<p>
<input id="cbGroup2" type="checkbox">Select all
<input name="cbGroup2" type="checkbox" value="value2_1">Group2 value 1
<input name="cbGroup2" type="checkbox" value="value2_2">Group2 value 2
<input name="cbGroup2" type="checkbox" value="value2_3">Group2 value 3
</p>

<script type="text/javascript" language="javascript">
    $("input").change(function() {
      $('input[name=\''+this.id+'\']').not(this).prop('checked', this.checked);
      $('#'+this.name).prop('checked', $('input[name=\''+this.name+'\']').length ===
$('input[name=\''+this.name+'\']').filter(':checked').length);
    });
</script>
```

Chapter 18: Plugins

Section 18.1: Plugins - Getting Started

The jQuery API may be extended by adding to its prototype. For example, the existing API already has many functions available such as `.hide()`, `.fadeIn()`, `.hasClass()`, etc.

The jQuery prototype is exposed through `$.fn`, the source code contains the line

```
jQuery.fn = jQuery.prototype
```

Adding functions to this prototype will allow those functions to be available to be called from any constructed jQuery object (which is done implicitly with each call to jQuery, or each call to $ if you prefer).

A constructed jQuery object will hold an internal array of elements based on the selector passed to it. For example, `$('.active')` will construct a jQuery object that holds elements with the active class, at the time of calling (as in, this is not a live set of elements).

The `this` value inside of the plugin function will refer to the constructed jQuery object. As a result, `this` is used to represent the matched set.

Basic Plugin:

```
$.fn.highlight = function() {
    this.css({ background: "yellow" });
};

// Use example:
$("span").highlight();
```

jsFiddle example

Chainability & Reusability

Unlike the example above, jQuery Plugins are expected to be **Chainable**.
What this means is the possibility to chain multiple Methods to a same Collection of Elements like `$(".warn").append("WARNING! ").css({color:"red"})` (see how we used the `.css()` method after the `.append()`, both methods apply on the same `.warn` Collection)

Allowing one to use the same plugin on different Collections passing different customization options plays an important role in **Customization / Reusability**

```
(function($) {
  $.fn.highlight = function( custom ) {

    // Default settings
    var settings = $.extend({
        color : "",              // Default to current text color
        background : "yellow"    // Default to yellow background
    }, custom);

    return this.css({     // `return this` maintains method chainability
        color : settings.color,
```

```
        backgroundColor : settings.background
    });

  };
}( jQuery ));


// Use Default settings
$("span").highlight();    // you can chain other methods

// Use Custom settings
$("span").highlight({
    background: "#f00",
    color: "white"
});
```

jsFiddle demo

Freedom

The above examples are in the scope of understanding basic Plugin creation. Keep in mind to not restrict a user to a limited set of customization options.

Say for example you want to build a `.highlight()` Plugin where you can pass a desired **text** String that will be highlighted and allow maximal freedom regarding styles:

```
//...
// Default settings
var settings = $.extend({
   text : "",          // text to highlight
   class : "highlight" // reference to CSS class
}, custom);

return this.each(function() {
   // your word highlighting logic here
});
//...
```

the user can now pass a desired **text** and have complete control over the added styles by using a custom CSS class:

```
$("#content").highlight({
    text : "hello",
    class : "makeYellowBig"
});
```

jsFiddle example

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content, more changes can be sent to web@petercv.com for new content to be published or updated

A.J	Chapters 1, 4 and 8
acdcjunior	Chapters 1 and 4
Alex	Chapter 15
Alex Char	Chapter 10
Alon Eitan	Chapter 5
amflare	Chapters 1 and 5
Andrew Brooke	Chapter 16
Anil	Chapter 1
Arun Prasad E S	Chapter 16
Ashiquzzaman	Chapter 15
Ashkan Mobayen Khiabani	Chapters 9, 11, 12, 13 and 16
Assimilater	Chapter 7
Athafoud	Chapter 16
ban17	Chapter 4
Ben H	Chapter 16
Bipon	Chapters 3 and 11
Brandt Solovij	Chapter 9
Brock Davis	Chapter 7
bwegs	Chapter 1
Castro Roy	Chapter 2
charlietfl	Chapter 6
csbarnes	Chapter 16
Darshak	Chapter 11
David	Chapter 2
DefyGravity	Chapter 7
DelightedD0D	Chapter 2
Deryck	Chapters 7 and 11
devlin carnate	Chapter 2
dlsso	Chapter 8
Dr. J. Testington	Chapter 16
Emanuel Vintilă	Chapter 5
empiric	Chapters 11 and 12
Flyer53	Chapter 11
Fueled By Coffee	Chapter 1
Gone Coding	Chapters 6 and 15
hasan	Chapters 2 and 18
Horst Jahns	Chapter 6
Iceman	Chapter 2
Igor Raush	Chapter 1
J F	Chapter 5
j08691	Chapter 9
Jatniel Prinsloo	Chapter 6
jkdev	Chapters 1 and 5
JLF	Chapter 2
John C	Chapters 1 and 16
John Slegers	Chapter 2
Jonathan Michalik	Chapter 9

Joram van den Boezem	Chapter 5
kapantzak	Chapter 2
Kevin Katzke	Chapter 1
Keyslinger	Chapter 2
Lacrioque	Chapter 16
Liam	Chapter 5
Luca Putzu	Chapters 1 and 6
Mark Schultheiss	Chapters 5 and 7
mark.hch	Chapter 8
martincarlin87	Chapter 7
Matas Vaitkevicius	Chapter 1
Melanie	Chapter 2
Mottie	Chapter 1
Neal	Chapter 1
Nhan	Chapter 5
ni8mr	Chapter 1
Nico Westerdale	Chapter 5
Nirav Joshi	Chapter 16
NotJustin	Chapter 6
Nux	Chapter 2
ochi	Chapter 4
Ozan	Chapter 16
Pranav C Balan	Chapter 12
Proto	Chapter 11
Renier	Chapter 3
rmondesilva	Chapter 8
Roko C. Buljan	Chapters 1, 9 and 18
Rupali Pemare	Chapter 10
Scimonster	Chapters 4 and 5
secelite	Chapter 5
SGS Venkatesh	Chapter 8
Shaunak D	Chapters 1, 2 and 16
Shekhar Pankaj	Chapter 2
Shlomi Haver	Chapter 9
Simplans	Chapter 14
Sorangwala Abbasali	Chapters 2 and 9
ssb	Chapter 2
still_learning	Chapter 7
sucil	Chapter 8
Suganya	Chapter 1
Sunny R Gupta	Chapter 6
Sverri M. Olsen	Chapter 8
TheDeadMedic	Chapter 5
Theodore K.	Chapter 10
The_Outsider	Chapters 8 and 10
Travis J	Chapters 1, 2 and 18
Upal Roy	Chapter 5
user1851673	Chapter 17
user2314737	Chapter 10
VJS	Chapter 14
Washington Guedes	Chapter 6
WOUNDEDStevenJones	Chapter 2
Yosvel Quintero	Chapters 1 and 16

Zakaria Acharki	Chapter 6
Zaz	Chapters 2 and 10

You may also like