JQUERY

TRICKSHOTS

100 ADVANCED TECHNIQUES

A BOOK BY TUTORIALZINE

Contents

Introduction

Hey reader, thank you for purchasing this book! As you probably know, jQuery is the most popular JavaScript library in use today. It gives web developers a great deal of power by abstracting away the complex tasks of traversing the DOM, creating elements, handling events and more, while at the same time leveling the play field by working around browser differences.

In this book, you will find a collection of **100 advanced jQuery and JavaScript techniques,** presented as a series of easy to follow tips that we have developed over the years as JavaScript programmers and authors for Tutorialzine.

If you are an experienced jQuery developer, or a beginner just starting out with JavaScript, this book is for you. You will learn neat tricks, find out about awesome plugins, and discover how to use (and abuse) our favorite library to your advantage.

To try out the examples in the chapters, you will have to paste the code into an HTML document and open it in a browser. Some of the examples will not work when opened directly as a file (due to browsers' security restrictions), so it would be best to open them through a locally running web server like Apache (you can install one of the available packages like XAMPP, MAMP and others). Along with the book, you have received a zip with all the examples as separate HTML files that you can run and edit directly in your browser, which will be much easier.

At any time, you can visit the book home page, or reach out to us on twitter if you have any suggestions or questions about the book.

Now, let's begin our journey!

I. Dom Manipulation

jQuery is the library of choice for DOM manipulation. It gives you a friendly CSS syntax for selecting elements, the ability to filter and traverse them and to modify them in more ways you can think of. In this chapter we will start with the basics and gradually move to more challenging tricks.

1. The DOM Ready Event

The first step to manipulating the DOM is subscribing to the DOM Ready event. This is not only a matter of convention – the event is fired only when all page elements are present and accessible, so your jQuery selectors actually return anything. But do you know that there is a neat way to subscribe to the ready event that works consistently cross-browser and doesn't even depend on jQuery? Here is how to do it:

```
// with jQuery
$(document).ready(function(){ /* ... */ });

// shorter jQuery version
$(function(){ /* ... */ });

// without jQuery (doesn't work in older IEs)
document.addEventListener('DOMContentLoaded', function(){
    // your code goes here
}, false);

// and here's the trick (works everywhere):

r(function(){
    alert('DOM Ready!');
});

function r(f){/in/.test(document.readyState)?setTimeout('r('+f+')',9):f()}
```

The trick here, as explained by the original author, is that we are checking the **document.readyState** property. If it contains the string **in** (as in Loading) we set a timeout and check again. Otherwise we execute the passed function.

2. Execute Page Specific Code

This trick is strictly about code organization, but it is good to know nonetheless. If you write lots of JavaScript it can quickly get difficult to manage. This is especially true when most of your code resides in a single file that is included in all the pages of your site. You end up with what is know as *DOM Spaghetti*. Here is one simple way to keep your code in check and avoid bugs: write a page routing function that executes only the JS that is needed for that particular page.

```
var route = {
    _routes : {},      // The routes will be stored here

    add    : function(url, action){
        this._routes[url] = action;
    },

    run : function(){
        jQuery.each(this._routes, function(pattern){
            if(location.href.match(pattern)){
                // "this" points to the function to be executed
                this();
            }
        });
    }
}

// Will execute only on this page:
route.add('002.html', function(){
    alert('Hello there!');
});

route.add('products.html', function(){
    alert("this won't be executed :(");
});

// You can even use regex-es:
route.add('.*.html', function(){
    alert('This is using a regex!');
});

route.run();
```

Each function is executed only when the path is matched. You can even use regexes for greater control.

3. Use the JavaScript AND Trick

The JavaScript logical **AND** operator doesn't evaluate the second expression if the first is false. You can use this to your advantage and save yourself from having to write a full if-statement:

```
// Instead of writing this:
if($('#elem').length){
    // do something
}

// You can write this:

$('#elem').length && alert("doing something");
```

This works best when checking single boolean variables. Using it in place of more complex conditionals is to be avoided as it will make your code difficult to comprehend.

4. Use the jQuery is() method

The **is()** method is more powerful than you think. Here are a few examples:

HTML

```
<div id="elem"></div>
```

JS

```
// First, cache the element into a variable:
var elem = $('#elem');

// Is this a div?
elem.is('div') && console.log("it's a div");

// Does it have the bigbox class?
elem.is('.bigbox') && console.log("it has the bigbox class!");

// Is it visible? (we are hiding it in this example)
elem.is(':not(:visible)') && console.log("it is hidden!");

// Animating
elem.animate({'width':200},1);

// is it animated?
elem.is(':animated') && console.log("it is animated!");
```

Result

```
it's a div
it is hidden!
it is animated!
```

You can use all kinds of selectors as the basis for your **is()** checks. Here we are also using the JavaScript && trick we discussed in tip #3, which is the reason that you see only three lines in the log.

5. Find out how many elements your page has

The more elements your page has, the slower it is to download and render. This won't affect your website loading speed that much, but it's a good indicator to check every once and a while, especially when doing a redesign - after all, you would want your new design to be leaner, not heavier.

Here is how to do it with jQuery:

```
// How many elements does your page have?
console.log('This page has ' + $('*').length + ' elements!');
```

This simply passes the universal selector (which matches everything) to jQuery, and logs the number of elements found.

6. Define an exists() function

By now you are probably used to checking the length property of jQuery objects to determine whether the element you attempted to select exists. The following trick will make your code a bit more expressive and easier to read:

HTML

```
<div id="elem"></div>
```

JS

```
// Old way
console.log($('#elem').length == 1 ? "exists!" : "doesn't exist!");

// The trick:

jQuery.fn.exists = function(){ return this.length > 0; }

console.log($('#elem').exists() ? "exists!" : "doesn't exist!");
```

Once you start using it, it feels like this method should have always been part of jQuery.

7. Use the second argument to the $() function

The **jQuery()** function takes two arguments. Do you know what the second one does? Let's say we have the following markup on our page:

```
<ul id="firstList">
    <li>one</li>
    <li>two</li>
    <li>three</li>
</ul>

<ul id="secondList">
    <li>blue</li>
    <li>green</li>
</ul>
```

We can now write some jQuery to demonstrate what the second argument does:

```
// Select an element. The #firstList is the context that limit the search.
// You can use a selector, a jQuery object or a dom element

$('li','#firstList').each(function(){
    console.log($(this).html());
});

console.log('-----');
```

```
// Create an element. The second argument is an
// object with jQuery methods to be called.

var div = $('<div>',{
    "class": "bigBlue",
    "css": {
        "background-color":"purple"
    },
    "width" : 20,
    "height": 20,
    "animate" : {   // You can use any jQuery method as a property!
        "width": 200,
        "height":50
    }
});

div.appendTo('body');
```

The context that we give in the first case, is roughly equivalent to using the **find()** method:

```
$('#firstList').find('li');
```

The second example saves us from having to call each of the methods individually.

8. Mark external links with an icon

For better usability, you might want to consider adding an icon next to links that lead to external websites. This is easy to do with jQuery:

```
<ul id="links">
     <li><a href="007.html">The previous tip</a></li>
     <li><a href="./009.html">The next tip</a></li>

     <!-- External Link: -->
     <li><a href="http://www.google.com/">Google</a></li>
</ul>
```

JS

```
// Loop through all the links
$('#links a').each(function(){

	if(this.hostname != location.hostname){
		// The link is external
		$(this).append('<img src="external.png" />')
			.attr('target','_blank');
	}

});
```

If you want to, you could even bind an event listener for the click event on these links and display a notification informing people where the link would take them.

9. Master the end() method

Chaining method calls is the bread and butter of effective jQuery development. The **end()** method is a powerful tool that you can impress your friends with. What it does, is to restore your jQuery collection to what it was before the last time it was modified (this includes filtering, finding, slicing and more). Here is an example:

HTML

```
<ul id="meals">
	<li>
		<ul class="breakfast">
			<li class="eggs">No</li>
			<li class="toast">No</li>
			<li class="juice">No</li>
		</ul>
	</li>
</ul>
```

JS

```
var breakfast = $('#meals .breakfast');

breakfast.find('.eggs').text('Yes')
			.end() // back to breakfast
			.find('.toast').text('Yes')
```

```
                        .end()
                        .find('.juice').toggleClass('juice coffee')
                        .text('Yes');

breakfast.find('li').each(function(){
    console.log( this.className + ': ' + this.textContent );
});
```

And here is the result:

```
eggs: Yes
toast: Yes
coffee: Yes
```

10. Prevent right clicking

If you want to give your web application a more native feel, you might want to disable right clicking. When a right click occurs, browsers emit the **contextmenu** event and as with any other event, you can listen for it, and call the preventDefault() method. Here is the trick:

```
$(function(){
    $(document).on("contextmenu",function(e){
          e.preventDefault();
    });
});
```

You can take this a few steps further and even display an entirely custom context menu by using the coordinates that are passed as properties of the **e** event object.

11. Break out of iframes

Some sites like StumbleUpon and Linkedin show your site with their bar on top of the page. This is done by including your page in an iframe. Here is how to break out of it:

```
if(window != window.top){
    window.top.location = window.location;
}
```

You only need to compare the window object of your page, to window.top. Normally, they are the same object, but if your site is being shown in an iframe, they will differ. Then you simply redirect the browser directly to your site.

12. Parse URLs with Anchor Elements

Parsing a URL into parts is a major pain in the behind. Your first thought might be to find a regex on the web that does this, but there is a much easier technique - by using a hyperlink!

```
// You want to parse this address into parts:
var url = 'http://tutorialzine.com/books/jquery-trickshots?trick=12#comments';

// The trick; create a new link with the url as its href:
var a = $('<a>',{ href: url });

console.log('Host name: ' + a.prop('hostname'));
console.log('Path: ' + a.prop('pathname'));
console.log('Query: ' + a.prop('search'));
console.log('Protocol: ' + a.prop('protocol'));
console.log('Hash: ' + a.prop('hash'));
```

The browser automatically parses the address and turns the URL parts into properties of the anchor object. The result is that you have all the hard work done for you:

```
Host name: tutorialzine.com
Path: /books/jquery-trickshots
Query: ?trick=12
Protocol: http:
Hash: #comments
```

13. Make your stylesheets editable

This is a fun trick that will show you that inline style blocks can be manipulated in the same way as other elements. They can be made visible and even editable with a small jQuery snippet.

For example let's assume that this is your inline stylesheet:

```
<style type="text/css" id="regular-style-block">

     html{
            background-color:#222229;
            position:relative;
     }

     body{
            font:14px/1.3 'Segoe UI', Arial, sans-serif;
            color:#e4e4e9;
            min-height:500px;
     }

</style>
```

To make the style block visible and editable, execute this code in your console:

```
$('#regular-style-block').css({'display':'block', 'white-space':'pre'})
                           .attr('contentEditable',true);
```

The changes you make to the stylesheet affect the page in real time. Recently this live-editing stopped working in Chrome, so you will have to try it out in Firefox to see the results.

14. Prevent text from being selectable

In certain situations, you might want to prevent text on the page from being selectable. This is useful when building interfaces that can be dragged or reordered, as you don't want users to select text from the page by accident. Here is a snippet that does this and works in all browsers:

```
$('p.descr').attr('unselectable', 'on')
            .css('user-select', 'none')
            .on('selectstart', false);
```

II. Performance

Web developers not only have to make their sites work, but to work fast. Nobody likes pages that seem to load forever. In this chapter, you will find tips and tricks on making your JavaScript faster and speeding up your web apps in the process. If you are up to the task, read on.

15. Include jQuery from a CDN

The best first step to improving the JavaScript performance of your web site is to simply include the latest version of jQuery, as every new release brings more optimizations and bug fixes. Requesting jQuery from a CDN is also a good choice, as it will minimize the loading time of your site (a CDN is faster to serve the library, and many people could have it in their caches in the first place):

```
<!-- Case 1 - requesting jQuery from the official CDN -->
<script src="http://code.jquery.com/jquery-1.10.2.min.js"></script>

<!-- Case 2 - requesting jQuery from Googles CDN (notice the protocol) -->
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js"></script>

<!-- Case 3 - requesting the latest minor 1.10.x version (only cached for an hour) -->
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.10/jquery.min.js"></script>

<!-- Case 4 - requesting the absolute latest jQuery version (use with caution) -->
<script src="http://code.jquery.com/jquery.min.js"></script>
```

Instead of including a specific version of the library, you can opt in for a specific major version with the latest minor release available at the moment, or even request the absolutely newest version of jQuery. Use the latter option with caution, as your code could stop working if there are breaking changes between major releases.

16. Keep DOM touches to a minimum

Despite all the advances in JavaScript performance, manipulating the dom is still expensive and should be kept to a minimum. This is especially important when inserting lots of elements into the page at once:

HTML

```
<div id="elem"></div>
```

JS

```
// Bad
//var elem = $('#elem');
//for(var i = 0; i < 100; i++){
//   elem.append('<li>element '+i+'</li>');
//}

// Good
var elem = $('#elem'),
    arr = [];

for(var i = 0; i < 100; i++){
    arr.push('<li>element '+i+'</li>');
}

elem.append(arr.join(''));
```

Grouping inserts into a single operation is faster as it causes the page to be redrawn only once. The same goes for style properties, where it's better to assign a css class than apply multiple styles.

17. Don't be afraid to use vanilla JS

Creating new jQuery objects carries overhead. This is why, if you are conscious about performance, you should use plain JavaScript where possible. In some places it is even easier and takes less code. Here is an example:

```
// Print the IDs of all LI items
$('#colors li').each(function(){

    // Access the ID directly, instead
    // of using jQuery's $(this).attr('id')

    console.log(this.id);

});
```

For performance-critical code like games, canvas manipulation, or other calculations, you should ditch jQuery and use plain *for* loops.

18. Optimize your selectors

If you are in need for some extra performance, but you still want to use jQuery, you should try optimizing your selectors. Here is a benchmark using the *time* and *timeEnd* methods of your browser's console (read more about the console it trick #47):

HTML

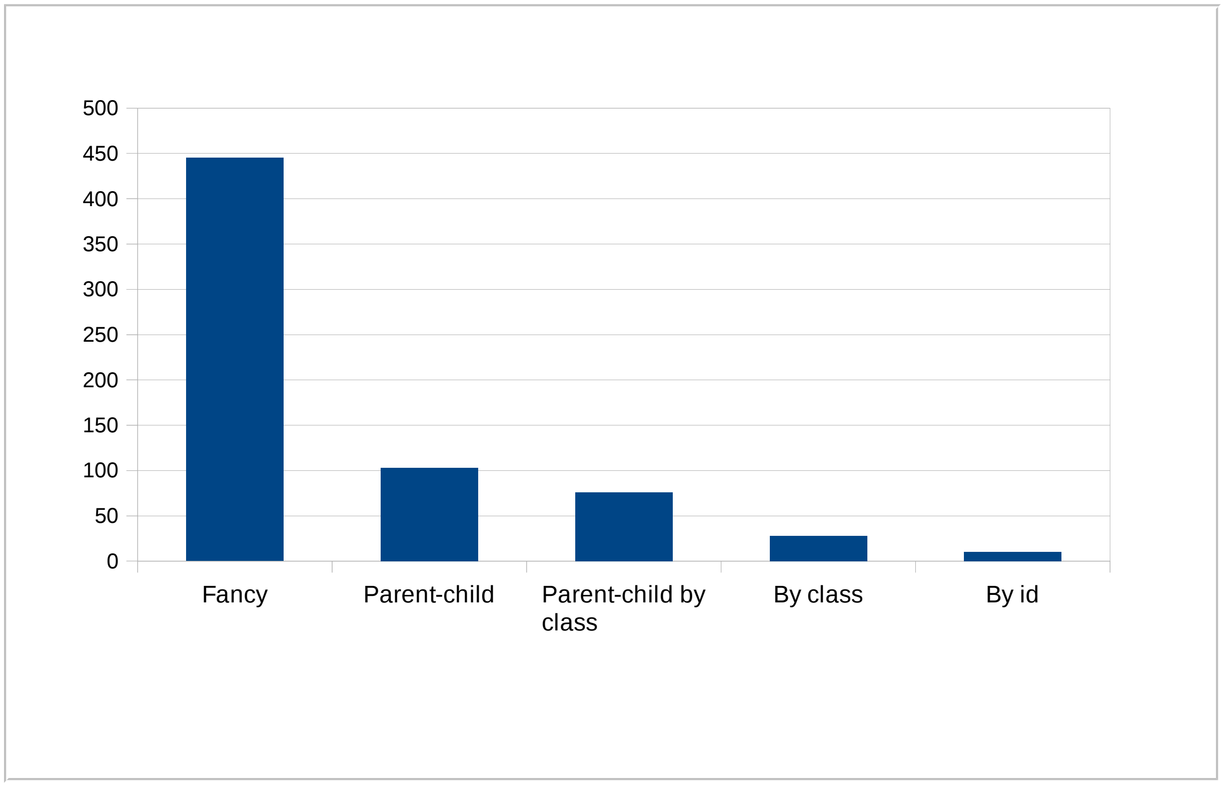
```
<div id="peanutButter">
     <div id="jelly" class=".jellyTime"></div>
</div>
```

JS

```
// Let's try some benchmarks!

var iterations = 10000, i;

console.time('Fancy');

for(i=0; i < iterations; i++){
     // This falls back to a SLOW JavaScript dom traversal
     $('#peanutButter div:first');
}

console.timeEnd('Fancy');


console.time('Parent-child');

for(i=0; i < iterations; i++){
     // Better, but still slow
     $('#peanutButter div');
}

console.timeEnd('Parent-child');


console.time('Parent-child by class');

for(i=0; i < iterations; i++){
     // Some browsers are a bit faster on this one
     $('#peanutButter .jellyTime');
}

console.timeEnd('Parent-child by class');


console.time('By class name');
```

```
for(i=0; i < iterations; i++){
    // Even better
    $('.jellyTime');
}

console.timeEnd('By class name');


console.time('By id');

for(i=0; i < iterations; i++){
    // Best
    $('#jelly');
}

console.timeEnd('By id');
```

Running this example in a modern browser, will produce results similar to the ones presented in the table below. The data shows that selecting by id is multiple times faster than the rest, and the more complex the selector, the slower the operation.



Of course, if you are only running the selector once, there isn't going to be much difference in the performance of your app. Of course, this implies that you should make use of another trick – caching jQuery objects.

19. Cache things

Every time you construct a new jQuery object by passing a selector, the Sizzle engine, which powers jQuery, traverses the DOM and matches the selector to real elements. This is slow when done with JavaScript, but the situation is improved in modern browsers which have built-in support for the **document.querySelector** functions which return elements matching CSS selectors directly. Unfortunately, this is not the case with older ones like IE 8 and below.

One good practice is to reuse your jQuery objects by assigning them to variables (or to "cache" them):

HTML

```
<ul id="pancakes">
    <li>first</li>
    <li>second</li>
    <li>third</li>
    <li>fourth</li>
    <li>fifth</li>
</ul>
```

JS

```
// Bad:
// $('#pancakes li').eq(0).remove();
// $('#pancakes li').eq(1).remove();
// $('#pancakes li').eq(2).remove();

// Good:
var pancakes = $('#pancakes li');

pancakes.eq(0).remove();
pancakes.eq(1).remove();
pancakes.eq(2).remove();

// Alternatively:
// pancakes.eq(0).remove().end()
//             .eq(1).remove().end()
//             .eq(2).remove().end();
```

This not only improves the performance of your app, but it also makes your code easier to read and manage.

20. Define repeated functions once

Another optimization that is good to know, is to define event listener functions beforehand if you are going to assign them to multiple elements, and to only pass them as a variable:

HTML

```
<button id="menuButton">Show Menu!</button>
<a href="#" id="menuLink">Show Menu!</a>
```

JS Example 1

```
// This results in multiple copies of the
// callback function taking up memory:

$('#menuButton, #menuLink').click(function(){
    // ...
});
```

JS Example 2

```
// This is better:

function showMenu(){
    alert('Showing menu!');
    // Doing something complex here
}

$('#menuButton').click(showMenu);
$('#menuLink').click(showMenu);
```

If you define your callback function inline, and the jQuery object contains more than one element (as in the commented example above), a copy of the callback will be kept in memory for every element in the collection.

21. Treat jQuery objects as arrays

You might have not noticed it, but there is a price to the elegance of jQuery's each method in terms of performance. There is a trick that you can use to loop through a jQuery object much faster - treat it as a regular array. It has a length property and numerically indexed values:

HTML

```
<ul id="testList">
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <!-- add 50 more -->
</ul>
```

JS

```
var arr = $('li'),
    iterations = 100000;

console.time('Native Loop');

for(var z=0;z<iterations;z++){

    var length = arr.length;
    for(var i=0; i < length; i++){
      arr[i];
    }
}
console.timeEnd('Native Loop');

console.time('jQuery Each');

for(z=0;z<iterations;z++){

    arr.each(function(i, val) {
      this;
    });
}
```

```
console.timeEnd('jQuery Each');
```

The native loop can be as much as 5 times faster than the jQuery each method! We aren't doing much else inside the loops, simply accessing the dom elements. If you are doing something more computationally expensive, the difference between the **$.each()** and **for()** variants will be less pronounced, as the interpreter will spend more of its time elsewhere.

22. Detach elements when doing complex modifications on them

Modifying a DOM element requires the page to be repainted on every style change, which can be expensive. If you want to squeeze every bit of performance, you can try detaching the element from the page when doing intense modifications on it.

HTML

```
<div id="elem" style="background:blue"></div>
```

JS

```
var i = 0, iterations = 1000;

// Modifying in place
var elem = $('#elem');

console.time('In place');

for(i=0; i < iterations; i++){

     elem.width(Math.round(100*Math.random()));
     elem.height(Math.round(100*Math.random()));

}

console.timeEnd('In place');

var parent = elem.parent();

// Detaching first
console.time('Detached');
```

```
elem.detach();

for(i=0; i < iterations; i++){

    elem.width(Math.round(100*Math.random()));
    elem.height(Math.round(100*Math.random()));

}

elem.appendTo(parent);

console.timeEnd('Detached');
```

Detaching elements beforehand can be as much as twice as fast. This doesn't apply to width and height only – every style or content change triggers a redraw of the element or even the entire page.

23. Don't wait for the load event

We've grown accustomed to placing all our code in the **$(document).ready()** event handler. However, if you have a heavy page, document ready may be delayed. But there is a trick to work around this , which improves the responsiveness of the page – by using event delegation we can bind events even if the page is not yet ready:

```
// jQuery is loaded at this point. We can use
// event delegation right away to bind events
// even before $(document).ready() is fired:

$(document).on('click', '#clickMe', function(){
    alert('Hey handsome!');
});

$(document).ready(function(){

    // This is where you would usually bind event handlers,
    // but as we are using delegation, there is no need to.

    // $('#clickMe').click(function(){ alert('Hey!'); });
});
```

Outside of document ready you can initialize objects, perform calculations, run AJAX requests and more.

24. Create a stylesheet when styling multiple elements

As we mentioned before, touching the DOM is slow. One way to work around this with regard to changing style rules, is by creating a stylesheet with the properties you want, and to add it to the document.

HTML

```
<ul id="testList">
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <li>Item</li>
    <!-- 100 more -->
</ul>
```

JS

```
var style = $('<style>');

// Add your CSS properties as text:
style.text('#testList li{ color:red; font-size:20px;}');

// Placing it before the result section so it affects the elements
style.prependTo('body');
```

This is faster, because browsers detect when a redraw is needed, and update all the elements at once. Something you simply can't match when looping through the elements and updating their styles manually.

III. Events

Easy event handling is the other feature that secured jQuery's place as the most popular JavaScript library. Every modern web application depends on events to get its job done. In this chapter, I will present a number of time saving tips and tricks that will help you make a better use of jQuery's functionality and to solve real problems you might encounter.

25. Assign a "JS" class to the <html> element

Web apps often depend on large amounts of JavaScript to interact with the user. However, JavaScript might be disabled or not available. In this case you should at least make sure that your application is usable.

There is one trick that can make this easier – in the load event, assign a class name to the <html> element. With CSS, you then show specific elements on the page that depend on JavaScript – like charts, holders for AJAX content and more – only when that class is present:

CSS

```
#message {display:none;}
html.JS #message { display:block; }
```

HTML

```
<p id="message">
    This is a message that is only shown when JavaScript is available.
</p>
```

JS

```
// That is all there is to it!
$(document).ready(function(){
    $('html').addClass('JS');
});
```

You can use this technique to not only hide, but change the appearance of your site entirely, depending on the capabilities of the device. (For more on detecting browser capabilities check tip #65).

26. Listen for events on elements that do not yet exist

Novice jQuery programmers often find themselves in the situation where they insert new elements to the page as the result of an AJAX request, only to find that their event handlers are not called. jQuery has an advanced event handling mechanism, exposed through the **on()** method that can solve this problem through event delegation (it can also enhance your site's performance, as we saw in tip #23):

HTML

```
<ul id="testList">
    <li>Old</li>
    <li>Old</li>
    <li>Old</li>
    <li>Old</li>
</ul>
```

JS

```
var list = $('#testList');

// Binding an event listener to the list, but listening for events on the li items:
list.on('click','li',function(){
    $(this).remove();
});


// This allows us to create li elements at a later time,
// while keeping the functionality in the event listener

list.append('<li>New item (click me!)</li>');
```

The event listener is set up on the **#testList** element, but as events bubble up the DOM tree, events originating in this element's LI children also notify the parent.

27. Single use event listener

Sometimes, you need to bind an event listener that should be executed only once. jQuery's got you covered – use the **one()** method:

HTML

```
<button id="press">Press me!</ul>
```

JS

```
var press = $('#press');
```

```
// There is a method that does exactly that, the one():
press.one('click',function(){
     alert('This alert will pop up only once');
});
```

What this method does, is call **on()** behind the scenes with a 1 as the last argument:

```
press.on('click',null,null,function(){alert('I am the one and only!');}, 1);
```

This is an implementation detail, so you shouldn't depend on this in production code as it may change in future versions.

28. Event simulation

As well as listening for events, jQuery gives you a very easy way to trigger them. This comes handy when dealing with third party code that doesn't expose a proper API. For example, you could simulate a click on the arrows of a slider to trigger a transition from your code.

HTML

```
<button id="press">Press me!</ul>
```

JS

```
var press = $('#press');

// Just a regular event listener:
press.on('click',function(e, how){
     how = how || '';
     alert('The buton was clicked ' + how + '!');
});

// Trigger the click event
press.trigger('click');

// Trigger it with an argument
press.trigger('click',['fast']);
```

To the original code this will appear as a regular click on the button. You can even pass additional parameters to the event listener, which will be available as arguments in the event handling callback.

29. Working with touch events

Someday touch will become the main way we interact with websites. Working with Touch events is not much different than the corresponding mouse ones. Clicks and hovers are already simulated by mobile browsers, but if you need more advanced functionality like detecting gestures or dragging, you will have to do a bit extra work. See the code below for an example.

JS

```
// Define some variables
var ball = $('<div id="ball"></div>').appendTo('body'),
startPosition = {}, elementPosition = {};

// Listen for mouse and touch events
ball.on('mousedown touchstart',function(e){
    e.preventDefault();

    // Normalizing the touch event object
    e = (e.originalEvent.touches) ? e.originalEvent.touches[0] : e;

    // Recording current positions
    startPosition = {x: e.pageX, y: e.pageY};
    elementPosition = {x: ball.offset().left, y: ball.offset().top};

    // These event listeners will be removed later
    ball.on('mousemove.rem touchmove.rem',function(e){
        e = (e.originalEvent.touches) ? e.originalEvent.touches[0] : e;

        ball.css({
            top:elementPosition.y + (e.pageY - startPosition.y),
            left: elementPosition.x + (e.pageX - startPosition.x),
        });

    });
});

ball.on('mouseup touchend',function(){
    // Removing the heavy *move listeners
    ball.off('.rem');
});
```

You have to listen for the corresponding mouse and touch events. Notice that I am attaching an event listener for the mouse and touch move events only when the drag starts, and remove them when it completes. These are expensive functions and it would be a poor choice to have them always active.

The result of this code is that you are able move the ball div around the screen on any device. As for detecting gestures, it would be best to use a dedicated library like hammer.js.

30. Know your events

jQuery simplified event handling with version 1.7, when it introduced the **on()** / **off()** methods. They have already come up in a number of tricks, but do you know how to use them properly?

HTML

```
<div id="holder">
    <button id="button1">1</button>
    <button id="button2">2</button>
    <button id="button3">3</button>
    <button id="button4">4</button>

    <button id="clear" style="float: right;">Clear</button>
</div>
```

JS

```
// Lets cache some selectors

var button1 = $('#button1'),
    button2 = $('#button2'),
    button3 = $('#button3'),
    button4 = $('#button4'),
    clear = $('#clear'),
    holder = $('#holder');

// Case 1: Direct event handling
button1.on('click',function(){
    console.log('Click');
```

```
});

// Case 2: Direct event handling of multiple events
button2.on('mouseenter mouseleave',function(){
    console.log('In/Out');
});

// Case 3: Data passing
button3.on('click', Math.round(Math.random()*20), function(e){

    // This will print the same number over and over again,
    // as the random number above is generated only once:
    console.log('Random number: ' + e.data);

});

// Case 4: Events with a namespace
button4.on('click.temp', function(e){
    console.log('Temp event!');
});

button2.on('click.temp', function(e){
    console.log('Temp event!');
});

// Case 5: Using event delegation
$('#holder').on('click', '#clear', function(){
    // This command is available only in FF:
    console.clear();
});

// Case 6: Passing an event map
var t; // timer

clear.on({

    'mousedown':function(){

        t = new Date();

    },

    'mouseup':function(){

        if(new Date() - t > 1000){

            // The button has been held pressed
            // for more than a second. Turn off
```

```
                // the temp events

                $('button').off('.temp');
                alert('The .temp events were cleared!');
            }

        }
});
```

The **on()** method unifies the old bind(), live(), and delegate() into one and you should start using it in their place. The older shortcuts like **click()** are still available for brevity's sake.

31. Quickly prevent default events actions

You are probably familiar with the **preventDefault()** method, called on the event object inside a listener. This is done to stop the browser from executing default actions like following links, submitting forms, checking boxes and more. There is also the **stopPropagation()** method that prevents the event from bubbling up the DOM tree. Returning **false** from the event listener will call both. However there is even a shorter way:

HTML

```
<a href="http://google.com/" id="goToGoogle">Go To Google</a>
```

JS

```
// Instead of this:

$('#goToGoogle').click(function(){
    return false;
});

// You can do this trick:

$('#goToGoogle').click(false);
```

You don't need to write out a full function – simply pass false as the sole argument. And bang! That's a second saved.

32. Chain event handlers with event.result

It is not uncommon to have more than one event handler bound to the same element. With **event.result**, you can pass data from one to the other, almost like chaining them together:

HTML

```
<button id="press">Press me!</button>
```

JS

```
var press = $('#press');
press.on('click',function(){
    return 'Hip';
});

// The second event listener has access
// to what was returned from the first

press.on('click',function(e){
    console.log(e.result + ' Hop!');
});
```

This technique would be even more powerful when passing state and other arguments between the functions, to enable advanced behavior in your front ends.

33. Create custom events

You can use jQuery's event handling infrastructure to your advantage. You can throw your own, custom events, and subscribe to them using the regular on/off methods. Some authors refer to this as the Pub/Sub pattern:

HTML

```
<button id="button1">Jump</button>
<button id="button2">Punch</button>
<button id="button3">Click</button>
<div id="eventDiv"></div>
```

JS

```
var button1 = $('#button1'),
     button2 = $('#button2'),
     button3 = $('#button3'),
    div = $('#eventDiv');

div.on({
    jump : function(){
        console.log('Jumped!');
    },

    punch : function(e,data){
        console.log('Punched '+data+'!');
    },

    click : function(){
        console.log('Simulated click!');
    }

});

button1.click(function(){
    div.trigger('jump');
});

button2.click(function(){
    // Pass data along with the event
    div.trigger('punch',['hard']);
});

button3.click(function(){
    div.trigger('click');
});
```

Pressing the buttons one after the other will give the following result:

Result

```
Jumped!
Punched hard!
Simulated click!
```

This technique will allow you to separate your code into clearly cut actions that are independent from the event handling of your interface.

IV. AJAX

AJAX is a fundamental building block for web apps. It allows you to send only the data that you need, saving bandwidth and speeding things up, making your sites feel native-like. In this chapter I will show you a number of tricks that you can use to enhance your applications and I'll explain a few of the new things that recent jQuery versions introduced.

34. Display file sizes next to download links

Did you know that you can send a HEAD request with AJAX and get the size of a file without downloading it? With jQuery this is very easy:

HTML

```
<a href="001.html" class="fetchSize">First Trickshot</a>
<a href="034.html" class="fetchSize">This Trickshot</a>
<a href="ball.png" class="fetchSize">Ball.png</a>
```

JS

```
// Loop all .fetchSize links
$('a.fetchSize').each(function(){

    // Issue an AJAX HEAD request for each one
    var link = this;

    $.ajax({
        type: 'HEAD',
        url: link.href,
        complete: function(xhr){

          var size = humanize(xhr.getResponseHeader('Content-Length'));

          // Append the filesize to each
         $(link).append(' (' + size + ')');

        }
    });

});


function humanize(size){
    var units = ['bytes','KB','MB','GB','TB','PB'];

    var ord = Math.floor( Math.log(size) / Math.log(1024) );
    ord = Math.min( Math.max(0,ord), units.length-1);

    var s = Math.round((size / Math.pow(1024,ord))*100)/100;
    return s + ' ' + units[ord];
}
```

This snippet places the size of the file in braces next to its name. The script issues a HEAD request, which only returns the headers and not the actual content of the file, which means that these requests are fast and lightweight.

```
First Trickshot (871 bytes)
This Trickshot (1.27 KB)
Ball.png (12.49 KB)
```

35. Simplify your Ajax calls with deferreds

Deferreds are a powerful tool. jQuery returns a new deferred object for every AJAX request, which makes them easier to work with. Here is how you can use deferreds to make your code more readable:

JS

```
// This is equivalent to passing a callback as the
// second argument (executed on success):

$.get('1.json').done(function(r){
    console.log(r.message);
});

// Requesting a file that does not exist. This will trigger
// the failure response. To handle it, you would normally have to
// use the full $.ajax method and pass it as a failure callback,
// but with deferreds you can can simply use the fail method:

$.get('non-existing.json').fail(function(r){
    console.log('Oops! The file is missing!');
});
```

As you will see in tip #55, deferreds have a lot of power behind them.

36. Run multiple AJAX requests in parallel

When working with APIs, you sometimes need to issue multiple AJAX requests to different endpoints. Instead of waiting for one request to complete before issuing the next, you can speed things up with jQuery by requesting the data in parallel, by using jQuery's **$.when()** function:

JS

```
$.when($.get('1.json'), $.get('2.json')).then(function(r1, r2){
    console.log(r1[0].message + " " + r2[0].message);
});
```

The callback function is executed when both of these GET requests finish successfully. **$.when()** takes the promises returned by two $.get() calls, and constructs a new promise object. The r1 and r2 arguments of the callback are arrays, whose first elements contain the server responses.

37. Get your IP with jQuery

Did you know you can get your public IP address with one line of JS? There is a free service that offers this for you and a get request is all that you need to do:

```
$.get('http://jsonip.com/', function(r){ console.log(r.ip); });
```

For the above snippet to work, your browser will have to support CORS (cross-origin request sharing). Otherwise a security exception would be thrown. In older browsers, you can use this version, which uses a JSON-P request:

```
$.getJSON('http://jsonip.com/?callback=?', function(r){ console.log(r.ip); });
```

38. The simplest AJAX request

jQuery offers a shorthand method for quickly loading content into an element via AJAX – the **.load()** method.

HTML

```
<p class="content"></p>
<p class="content"></p>
```

JS

```
var contentDivs = $('.content');

// Fetch the contents of a text file:
contentDivs.eq(0).load('1.txt');

// Fetch the contents of a HTML file, and display a specific element:
contentDivs.eq(1).load('1.html #header');
```

Unlike the rest of the AJAX methods, here you can specify a CSS selector that will limit the result. Also the data isn't returned, but directly replaces the content of the element.

39. Serializing objects

If you are serious about AJAX, you should be familiar with jQuery's methods for encoding forms and plain objects. The result is a URL-friendly representation that you can include in your AJAX request. Here is how to use them:

HTML

```
<form id="gform">
  <input type="text" name="performer" value="Gangnam K. Style" />
  <input type="text" name="email" value="psy@w3c.org" />
  <textarea name="lyrics">Na je nun ta sa ro un in gan jo gin yo ja</textarea>
</form>
```

JS

```
var form = $('#gform');

// Turn all form fields into a URL friendly key/value string.
// This can be passed as argument of AJAX requests, or URLs.
console.log(form.serialize());

console.log('---------');
```

```
// You can also encode your own objects with the $.param method:
console.log($.param({'pet':'cat', 'name':'snowbell'}));

console.log('---------');

// With $.serializeArray() you can encode the form to an object
console.log(form.serializeArray());
```

Result

```
performer=Gangnam+K.+Style&email=psy
%40w3c.org&lyrics=Na+je+nun+ta+sa+ro+un+in+gan+jo+gin+yo+ja
---------
pet=cat&name=snowbell
---------
[{"name":"performer","value":"Gangnam K. Style"},
{"name":"email","value":"psy@w3c.org"},{"name":"lyrics","value":"Na je nun ta sa ro un
in gan jo gin yo ja"}]
```

A lot of the times these conversions are made behind the scenes, when you pass an object as the argument to one of the AJAX functions.

40. File uploads with jQuery

Modern browsers support the FormData API, which, among other things, allows you to send binary data easily through AJAX. Combine this with the HTML5 File API, and you get an easy way for uploading files without the need for additional plugins.

The idea is to obtain a reference to a file through a file input field. Then we will create a new FormData object and append the file reference. Lastly, we will pass the FormData object directly to jQuery's AJAX method. For the last step to work though, we will have to set two of its properties to false to prevent jQuery from processing the data.

HTML

```html
<input type="file" />
<button id="upload">Upload it!</button>
```

JS

```js
var fileInput = $('input[type=file]'),
    button = $('#upload');

button.on('click', function(){

    // Access the files property, which holds
    // an array with the selected files

    var files = fileInput.prop('files');

    // No file was chosen!
    if(files.length == 0) {
        alert('Please choose a file to upload!');
        return false;
    }

    // Create a new FormData object
    var fd = new FormData();

    fd.append('file', files[0]);

    // Upload the file to assets/php/upload.php, which only prints the filename

    $.ajax({
        url: './assets/php/upload.php',
        data: fd,
        contentType:false,
        processData:false,
        type:'POST',
        success: function(m){
            console.log(m);
        }
    });
});
```

Accessing the files property of the file input field will give us an array with file objects from the HTML5 File API. We can add them to the FormData object, and assign them as the data property of the AJAX request. The tricky part is setting ***contentType*** and ***processData*** to false, so that jQuery doesn't attempt to serialize the data (which would only break the request) and leave it to the browser. No file upload plugins needed!

41. Work with Facebook's Graph

The Graph API is a very powerful interface to Facebook's huge pool of social data. There still are some parts of it that are publicly available. The API is available under the graph.facebook.com subdomain. Here is an example with Tutorialzine's FB page:

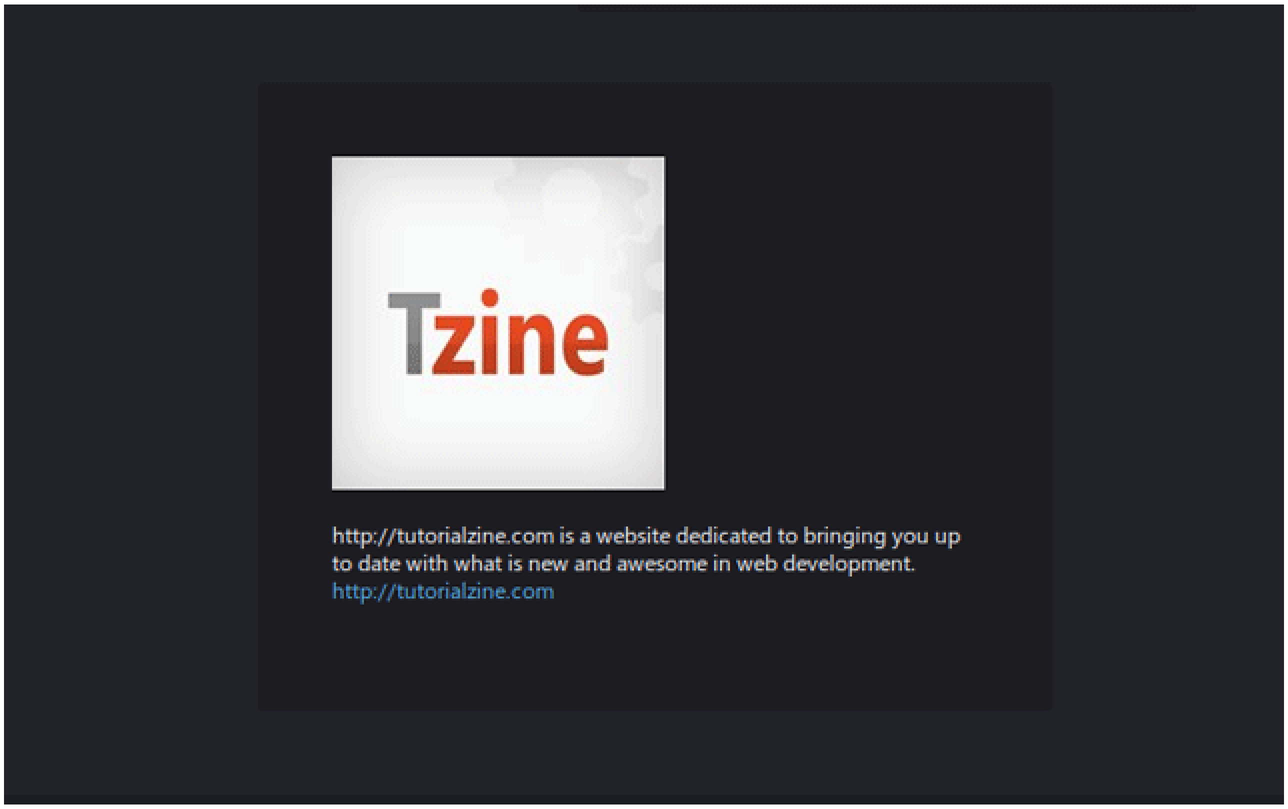
HTML

```
<div id="fbdata"></div>
```

JS

```
// Fetch the publicly accessible data on Tutorialzine's Page
var api = 'http://graph.facebook.com/Tutorialzine/?callback=?',
    holder = $('#fbdata');


$.getJSON(api, function(r){

     // This will always give the current picture
     holder.append('<img src="http://graph.facebook.com/Tutorialzine/picture/?
type=large">');

     holder.append('<p>'+ r.about +'</p>')
     holder.append('<a href="'+ r.website +'">'+ r.website +'</a>');

});
```

This prints the description of the page, link and the profile photo.


http://tutorialzine.com is a website dedicated to bringing you up to date with what is new and awesome in web development.
http://tutorialzine.com

42. Access weather information

Weather data is another place where a good API is crucial. Open Weather Map provides free weather information that you can use through their JSON API. Here is an example:

```
// Request weather data:
var api = 'http://openweathermap.org/data/2.1/find/name?q=paris,france&callback=?';


$.getJSON(api, function(r){

    // This will always give the current picture
    console.log(r.list[0].name + ', ' + r.list[0].sys.country);
    console.log(r.list[0].main);

    // Temperatures are in kelvin, subtract 273.15 to convert to Celsius,
    // or use the formula (kelvin*⁹⁄₅ - 459.67) for Fahrenheit
});
```

Result

```
Paris, FR
{"temp":277.02,"pressure":1020,"humidity":80,"temp_min":276.15,"temp_max":277.59}
```

You get real time data. Notice that the temperature is in Kelvin; you will need to manually convert it to a format for presenting to users.

43. Fetch your latest Tumblr posts

In the spirit of the great APIs of old, the popular blogging service Tumblr offers an easy way for you to fetch the posts of any blog as JSON. Here is how their API is used:

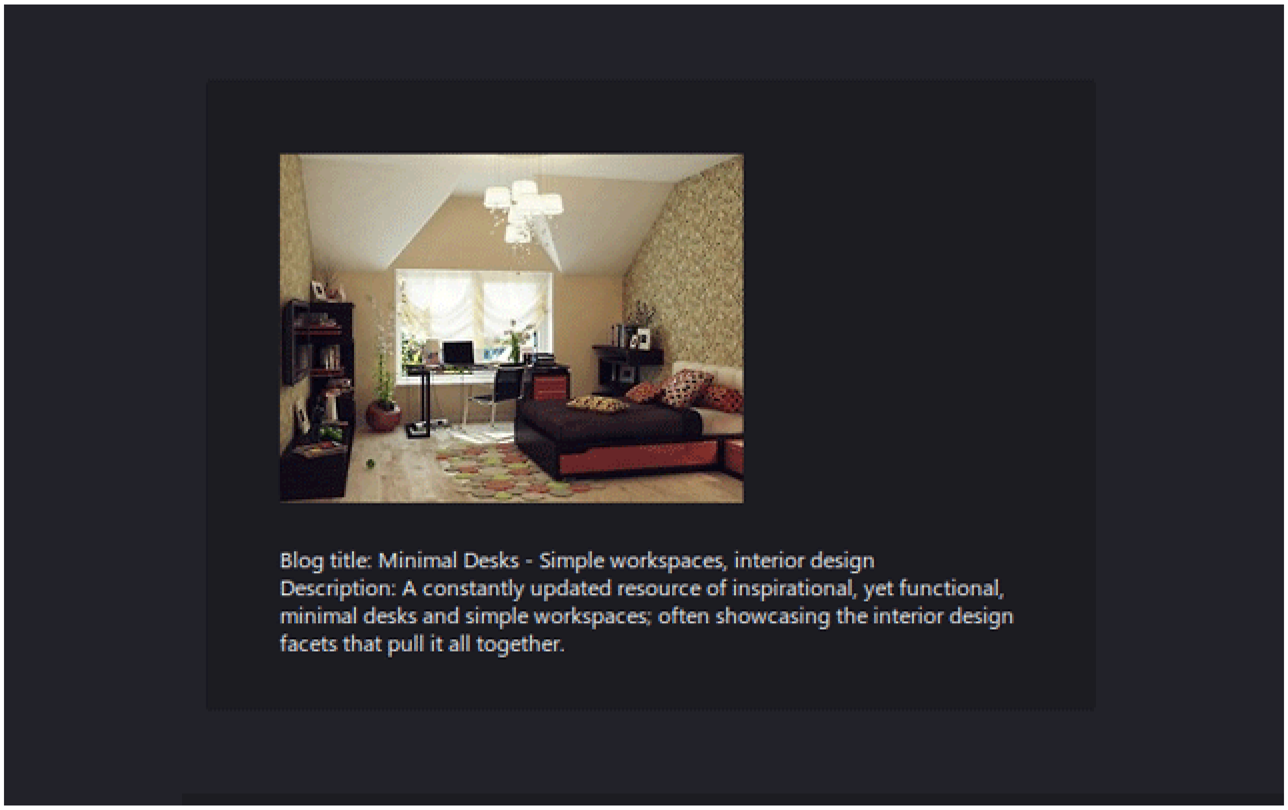
HTML

```
<div id="post"></div>
```

JS

```
// Define some variables
var blog = 'minimaldesks.tumblr.com',
    api  = 'http://' + blog + '/api/read/json?callback=?',
    post = $('#post');


$.getJSON(api, function(r){

    console.log('Blog title: ' + r.tumblelog.title);
    console.log('Description: ' + r.tumblelog.description);

    // If this post has a photo, show it
    if(r.posts[0]['photo-url-250']){
            post.append('<img src="' + r.posts[0]['photo-url-250'] + '" />');
    }
    else{
            console.log('Latest post: ' + r.posts[0]['regular-title']);
    }

});
```

This gives you details about the blog and the last post with a photo. It even works with custom domains – simply replace the tumblr URL with the domain.


Blog title: Minimal Desks - Simple workspaces, interior design
Description: A constantly updated resource of inspirational, yet functional, minimal desks and simple workspaces; often showcasing the interior design facets that pull it all together.

44. Find the geographic location of an IP address

There are services online that can tell you the city and country of an IP address. Here is how to use one of them – freegeoip.net:

```
// Define some variables
var ip = '', // you can optionally put an ip address here
    api  = 'http://freegeoip.net/json/' + ip + '?callback=?';


$.getJSON(api, function(r){

    console.log('How is the weather in ' + r.city + ', ' + r.country_name + '?');

});
```

If no IP address is provided, the API will assume the address of the client making the request.

45. Scrape sites with YQL

YQL is the ultimate API for the JavaScript developer. It makes it possible to work with all kinds of third party APIs through a SQL-like interface. Here is how to use it to fetch and parse HTML from remote sites:

```
// Define variables

var query = 'select * from data.html.cssselect where
url="http://www.chucknorrisfacts.com/chuck-norris-top-50-facts" and css=".field-
content a"';
var yqlAPI = 'http://query.yahooapis.com/v1/public/yql?q=' + encodeURIComponent(query)
+ ' &format=json&env=store%3A%2F%2Fdatatables.org%2Falltableswithkeys&callback=?';


$.getJSON(yqlAPI, function(r){

    console.log('Chuck Norris Facts:');

    $.each(r.query.results.results.a, function(){
          console.log('----------');
          console.log(this.content);
    });

});
```

The results speak for themselves:

```
Chuck Norris Facts:
----------
Chuck Norris has the right to keep and arm bears.
----------
Chuck Norris can turn the lights off by clapping his eyelids twice.
----------
When Alexander Bell invented the telephone he had 3 missed calls from Chuck Norris
----------
Fear of spiders is aracnaphobia, fear of tight spaces is chlaustraphobia, fear of
Chuck Norris is called Logic
----------
Chuck Norris doesn't call the wrong number. You answer the wrong phone.
```

46. Use the global AJAX methods

You can simplify how your web app handles AJAX requests (and save yourself from writing some code), by utilizing the ajax global methods:

```
// Create an indicator that would be shown whenever an AJAX request occurs:

var preloader = $('<div>',{ 'class':'preloader' }).appendTo('body');
var doc = $(document);

// Show the preloader whenever you are making an AJAX request:

doc.ajaxStart(function(){
     preloader.fadeIn();
});

// Hide it when the ajax request finishes

doc.ajaxComplete(function(){
     // Keep it visible for 0.8 seconds after the request completes
     preloader.delay(800).fadeOut();
});

// It will show automatically every time you do an AJAX request:

$.get('1.json');
```

These methods are called every time that one of jQuery's AJAX methods is used, which makes them perfect for showing and hiding indicators.

V. Master Class

Congratulations warrior! You have been accepted to the master class. This chapter contains some advanced tips about building plugins and extending jQuery that will help you with your day-to-day tasks as a kick-ass JavaScript developer.

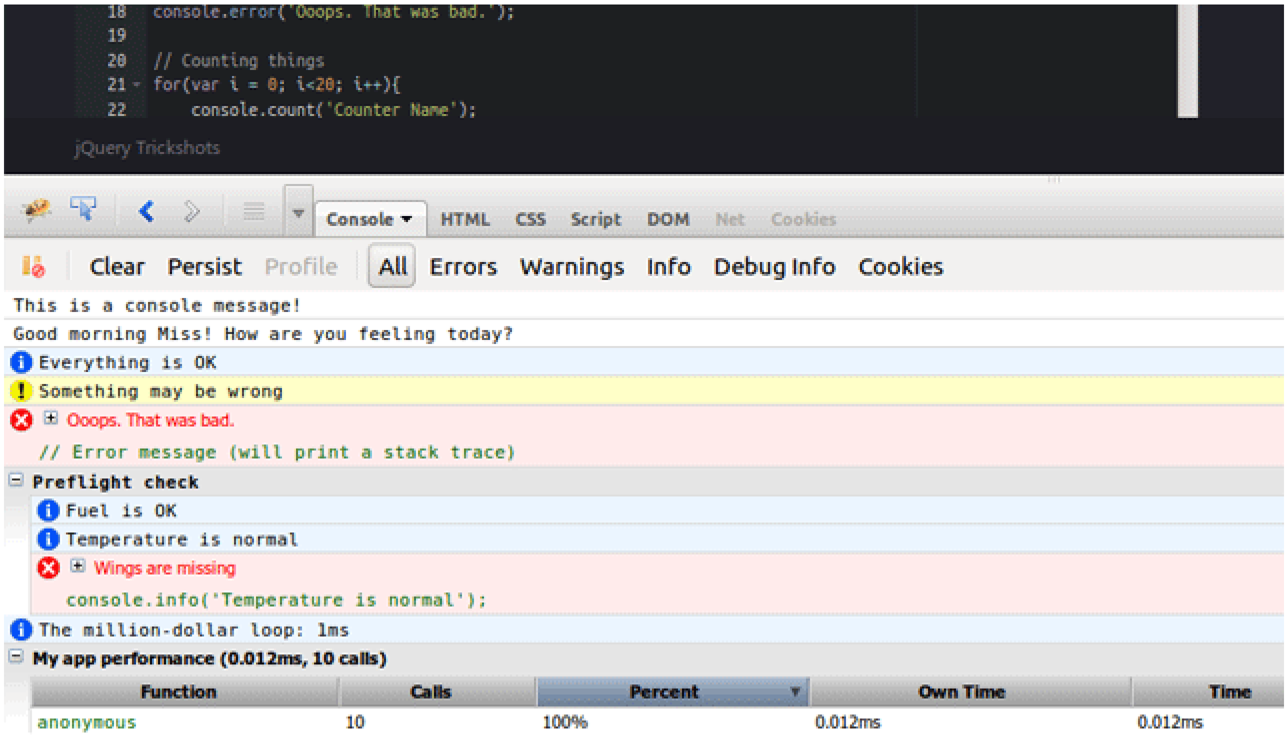
47. Learn to love the console

By now you probably know that there are better ways to debug your code than by using alerts. Chrome and Safari give you Webkit's powerful web developer tools, Firefox has Firebug and even IE has its own set of tools. All of them expose a **console** object, which is a good beginner-friendly way to debug your code.

Here is an example with all of the methods of the console object:

```
// The simple case. Use this instead of alert():
console.log('This is a console message!');

// It supports embedding of variables as well:
var a = 'morning', b = 'Miss';
console.log('Good %s %s! How are you feeling today?', a, b);

// Interactively browse the properties of a method (similar to console.log):
console.dir(window);

// Information message (in webkit it looks like console.log)
console.info('Everything is OK');

// Warning message
console.warn('Something may be wrong');

// Error message (will print a stack trace)
console.error('Ooops. That was bad.');

// Counting things
for(var i = 0; i<20; i++){
    console.count('Counter Name');
}

// Starts a collapsable group of log messages
console.group("Preflight check");
console.info('Fuel is OK');
console.info('Temperature is normal');
console.error('Wings are missing');
console.groupEnd()

// Timing things
console.time('The million-dollar loop')

var dollars = 0;

for(var i=0;i<100000; i++){
    dollars+=10;
}

console.timeEnd('The million-dollar loop');

// Profiling code (it will show up in your console's Profile tab)
console.profile('My app performance');

var arr = [];
$.each([0,1,2,3,4,5,6,7,8,9],function(){
    arr.push(this+1);
});

console.profileEnd('My app performance');
```

You can print logs, errors, warnings, time things and even profile your JS's performance. Read through the Chrome's developer tools guide or Firefox's firebug guide for more info on the other powerful features of these debuggers.



```
console.error('Ooops. That was bad.');

// Counting things
for(var i = 0; i<20; i++){
    console.count('Counter Name');
```

jQuery Trickshots

```
This is a console message!
Good morning Miss! How are you feeling today?
Everything is OK
Something may be wrong
Ooops. That was bad.
  // Error message (will print a stack trace)
Preflight check
  Fuel is OK
  Temperature is normal
  Wings are missing
  console.info('Temperature is normal');
The million-dollar loop: 1ms
My app performance (0.012ms, 10 calls)
```

Function	Calls	Percent	Own Time	Time
anonymous	10	100%	0.012ms	0.012ms

48. Convert code to plugins to promote reuse

The wealth of available plugins makes jQuery the best JavaScript library around. Have a piece of jQuery code that you copy/paste between projects? Consider converting it to a plugin. It is a lot simpler than you think, you only have to assign a function to the **$.fn** object.

Here is an example in which we are bringing placeholders (default text that shows up in a field when it is empty) to older browsers:

HTML

```
<input id="testInput" placeholder="Your Name" />
```

JS

```
// Define the placeholder plugin

$.fn.placeholder = function(){

    if ('placeholder' in document.createElement('input')){

        // This browser already supports placeholders.
        // Nothing to do here.
        return this;
    }

    this.each(function(){
        var input = $(this);

        input.on('focus', function(){

            if(input.val() == input.attr('placeholder')){

                 input.val('');
            }

        }).on('blur', function(){

            if(input.val() == ''){
                 input.val(input.attr('placeholder'));
            }

        });
        // Show the placeholder on load
        input.trigger('blur');
    });

    return this;
};

// And here is how to use it:
$('#testInput').placeholder();
```

Plugins, when done right, make your code more modular and self-contained. This makes them very easy to use across project and to share with the world. For more information, check out jQuery's plugin development guidelines.

49. Use anonymous functions to isolate code

Defining global variables and functions is a bad practice and can lead to nasty bugs. The better approach is to isolate your code in blocks using anonymous functions. This is also the technique that a lot of jQuery plugins use to keep their variables and functions private. Here is a short (and rather useless) example:

JS

```
// Isolating a block of code:

(function($){

    // Declare a variable. It will only be visible in this block.
    var c = 1;

    // Define a simple plugin

    $.fn.count = function(){

          // Increment and log the counter
          console.log(c++);

          return this;
    };

})(jQuery);
```

```
// The c variable is only visible for the plugin and will keep
// its value between invocations:

$(document).count();

$('body').count().count();
```

Result

```
1
2
3
```

We are declaring a function without a name, and execute it directly. As an argument, we are passing the jQuery object. This makes it available within the function as the **$** variable even in cases where jQuery.noConflict() is used and the dollar sign is undefined. Variables defined in the block are not visible from the outside. They also are defined only once and keep their value between invocations of the jQuery plugin.

50. Merge objects with extend

A good jQuery plugin is customizable. But how do you allow your code to be customized if it is supposed to be isolated? This is usually done by letting users pass parameters that modify your plugin's behavior. This leads to another problem – you want to have some default values of those parameters, so that even if the user doesn't pass anything the plugin still works as expected. This is handled by jQuery's powerful **extend()** method.

Here are some of the things that it can do:

```
// Combine properties (useful in plugins).
// The defaults are passed as the first argument.

var supplied = { height: 400 };

var options = $.extend({
     color  : 'blue',
     width  : 200,
     height : 150
}, supplied);
```

```
console.log('New options:', options);

// You can also pass more than one object

console.log('Three parents:', $.extend({a:2}, {b:3}, {c:4}) );

console.log('-------');


// Cloning objects.
// To clone an object, simply pass an empty one
// as the first argument

var original = {a:123, b:'#fff'};
var clone = $.extend({}, original);

console.log('Clone:', clone);

console.log('-------');


// Extending jQuery.
// You can define plugins with extend

$.extend($.fn, {
    plugin1: function(){
            console.log('Plugin 1');
            return this;
    },
    plugin2: function(){
            console.log('Plugin 2');
            return this;
    }
});

$('body').plugin1().plugin2();

console.log('-------');

// If you pass only one arguments to $.extend,
// it will add the properties to the jQuery object

$.extend({ dontDoThis : 123});

console.log($.dontDoThis);

console.log('-------');
```

```
// Deep cloning.
// If you have nested objects, you will have to
// pass one additional argument to extend:

var obj1 = { a: 1, b: 2, c: {d: 3} };
var obj2 = { c: {e: 4}, f:5};

// This won't work
// $.extend(obj1, obj2);

// This will
$.extend(true, obj1,obj2);

console.log('Deep clone:', obj1);
```

Result

```
New options: {"color":"blue","width":200,"height":400}
Three parents: {"a":2,"b":3,"c":4}
-------
Clone: {"a":123,"b":"#fff"}
-------
Plugin 1
Plugin 2
-------
123
-------
Deep clone: {"a":1,"b":2,"c":{"d":3,"e":4},"f":5}
```

Mastering the $.extend() method is a must for effective jQuery plugin development, especially if you want to share your plugins with the world.

51. Use jQuery.type()

Knowing the type of a variable is immensely useful. JavaScript has the build in **typeof** operator, but it is severely lacking in comparison to the one provided by jQuery:

JS

```
// The built-in operator is at the top,
// the jQuery version is below
```

```
console.log( typeof null );
console.log( $.type(null) );

console.log('---');

console.log( typeof undefined );
console.log( $.type(undefined) );

console.log('---');

console.log( typeof 'asdf' );
console.log( $.type('asdf') );

console.log('---');

console.log( typeof 123 );
console.log( $.type(123) );

console.log('---');

console.log( typeof [] );
console.log( $.type([]) );

console.log('---');

console.log( typeof /abc/ );
console.log( $.type(/abc/) );
```

Result

```
object
null
---
undefined
undefined
---
string
string
---
number
number
---
object
array
---
```

```
object
regexp
```

Knowing the exact type of a variable can come handy in a lot of places. Inside plugins, you can treat variables differently and provide advanced functionality, which can make your code more versatile.

52. The map method

Another useful, but not well known method in the jQuery library, is **map()**. It loops through all elements in a jQuery object, and assembles a new array. Here is how it is used:

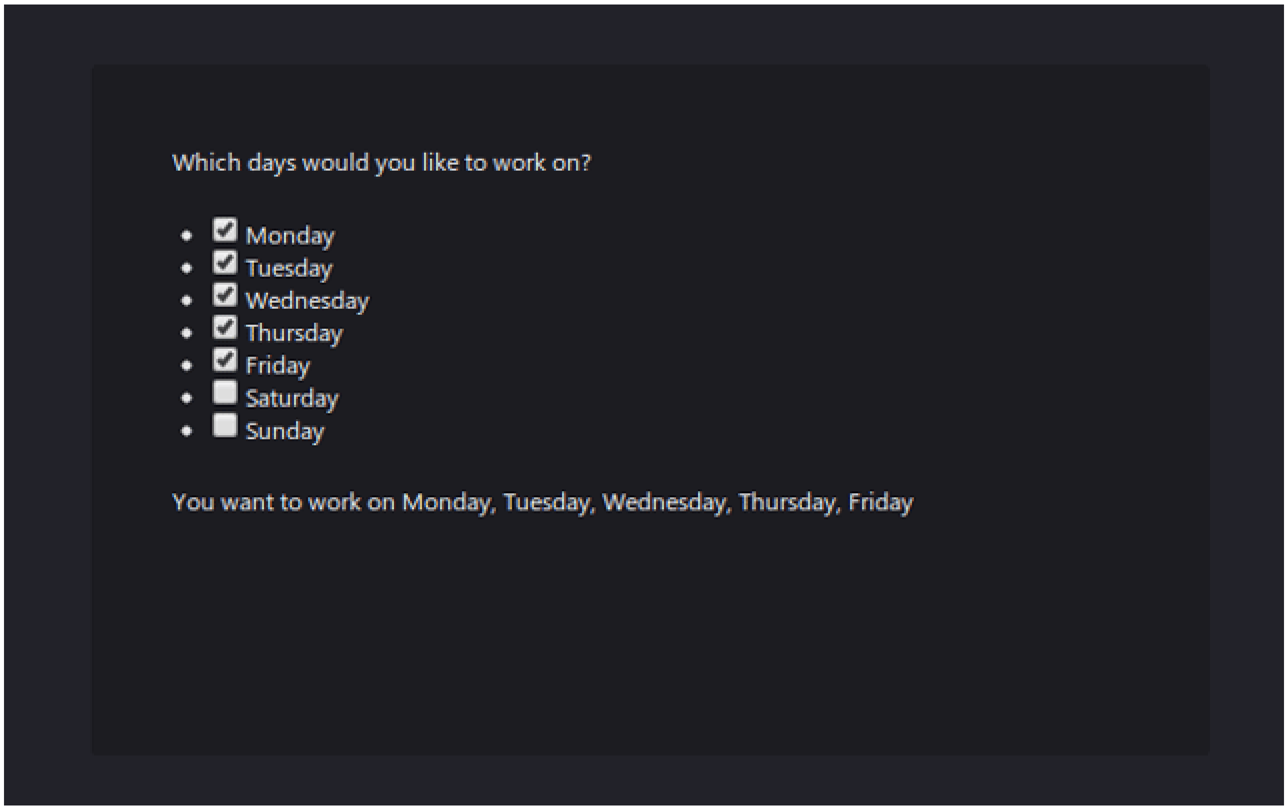
HTML

```
Which days would you like to work on?
<ul id="week">
    <li><input type="checkbox" value="Monday" class="day" checked /> Monday </li>
    <li><input type="checkbox" value="Tuesday" class="day" checked /> Tuesday </li>
    <li><input type="checkbox" value="Wednesday" class="day" checked />Wednesday </li>
    <li><input type="checkbox" value="Thursday" class="day" checked /> Thursday </li>
    <li><input type="checkbox" value="Friday" class="day" checked /> Friday </li>
    <li><input type="checkbox" value="Saturday" class="day" /> Saturday </li>
    <li><input type="checkbox" value="Sunday" class="day" /> Sunday </li>
</ul>

<p id="workdays"></p>
```

JS

```
// Find the checkboxes:
var checkboxes = $('#week .day');

function callback(){

    // Loop through all selected checkboxes and output their values.

    var days = checkboxes.filter(':checked').map(function(){
          // The returned value becomes part of the new array
          return $(this).val();
    }).get();

    $('#workdays').text('You want to work on ' + days.join(', '));
}

checkboxes.on('click',callback);

// Execute it on load
callback();
```

This example shows seven checkboxes, one for each day of the week. Once you click on a checkbox, the #workdays paragraph is updated with the names of the days you want to work on.



Which days would you like to work on?

- ☑ Monday
- ☑ Tuesday
- ☑ Wednesday
- ☑ Thursday
- ☑ Friday
- ☐ Saturday
- ☐ Sunday

You want to work on Monday, Tuesday, Wednesday, Thursday, Friday

53. The $.grep() method

Although jQuery's main power is in making DOM manipulation and event handling easy, it also has useful utility functions for working with arrays. One of them is the **$.grep()** function – it filters the elements of an array with a callback:

JS

```
// The original array

var arr = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];

// If the numbers in the array represented days in the current month,
// which of them would be a Sunday?

var sundays = $.grep(arr, function(day, index){
     var d = new Date();
     d.setDate(day);

     return d.getDay() == 0;   // Sundays are 0!
});

console.log('Sundays:', sundays.join(', '));
```

Result

```
Sundays: 3, 10
```

Returning true from the filter function will make the object part of the new array. In the example above I am using the JavaScript Date/Time functions to check whether a specific day of the current month is a Sunday.

54. Sort the elements inside a jQuery collection

This trick revolves around the use of a non-documented jQuery method - **sort**. This works the same way as the array's sort functionality:

HTML

```
<button id="sort">Sort!</button>
<ul id="numbers">
     <li>32</li>
     <li>12</li>
     <li>4</li>
     <li>1</li>
     <li>33</li>
     <li>121</li>
```

```
    <li>-2</li>
    <li>45</li>
    <li>7</li>
    <li>15</li>
</ul>
```

JS

```
// Listen for button clicks:
$('#sort').click(function(){

    // Select all the li items
   var numbers = $('#numbers li');

    numbers.sort(function(a,b){
          return parseInt(a.textContent, 10) > parseInt(b.textContent, 10);
    }).appendTo(numbers.parent());

});
```

This snippet compares the values of the LI elements and sorts them in ascending order. It is important to convert them to numbers before comparing, otherwise they will be sorted as strings.

55. More about jQuery Deferreds

We already mentioned jQuery Deferreds in the AJAX section of this book. They are a tool that can simplify how we work with asynchronous events. Let's take a more in depth look and try to create a JSON_Reader object, that returns a deferred:

```
function JSON_Reader(name){
    var d = new $.Deferred();

    $.ajax({
          url: '' + name + '.json',
          dataType: 'json',
          success: function(data){
                d.resolve(data);
          },
          error: function(){
                d.reject();
```

```
            }

        });

        // You should return a promise object, so that
        // third party code can attach event handlers:

        return d.promise();
}

// Let's use it

var one = new JSON_Reader('1');

one.done(function(d){
    console.log('Data received:', d);
});

one.fail(function(){
    console.log('The file does not exist!');
});
```

Result

```
Data received: {"message":"Hello"}
```

First, we construct a deferred and issue an AJAX request. Depending on its outcome, we will either resolve or reject the deferred (which will respectively execute the **done** or **fail** callbacks that are set near the end of the fragment). The promise object we return is a stripped down version of the deferred that doesn't have the resolve/reject methods and can only be used for subscribing to events.

Note that jQuery's AJAX methods already return a promise, so if it weren't for the example we could have just as well replaced the entire JSON_Reader class with a call to $.getJSON().

Of course, one example is not enough to fully understand a complex concept as Deferreds. You can read through the articles at jQuery's documentation site for more.

56. Call jQuery methods conditionally

This is an old JavaScript trick, that can be easily applied to jQuery. It uses the fact that all JavaScript object properties are also available with an array accessor:

HTML

```
<div id="rectangle" style="display:none;
background-color:white;width:100px;height:100px;"></div>
```

JS

```
// Define some variables:

var element = $('#rectangle');
var direction = 'down';

// Instead of this:

if(direction == 'up'){
    element.slideUp();
}
else{
    element.slideDown();
}

// You can do this:

element[ direction == 'up' ? 'slideUp' : 'slideDown' ]();
```

57. jQuery callback lists

jQuery's $.Callbacks() method gives you a powerful way to manage a large number of callback functions. You can add, remove, and disable individual callbacks, and to fire the entire list. Here is how it is used:

HTML

```
<button id="btn1">1</button>
<button id="btn2">2</button>
<input type="checkbox" id="checkBox" checked />
```

JS

```
// First example - Using the callbacks object.

// Create a new callbacks object
var cb = $.Callbacks();

// Add a callback
cb.add(function(message){
    console.log(message);
});

// Add one more
cb.add(function(){
    console.log('---');
});

// Trigger the callbacks
$('#btn1').click(function(){

    // Output three lines
    cb.fire("Line 1");
    cb.fire("Line 2");
    cb.fire("Line 3");
});

// Second example - Using flags

// Will stop executing callbacks on false
var cflags = $.Callbacks('stopOnFalse')

cflags.add(function(){
    // Is the checkbox checked?
    return $('#checkBox').is(':checked');
});

cflags.add(function(){
    console.log('Checkbox is checked!');
});

// Trigger the callbacks
$('#btn2').click(function(){
    cflags.fire();
});
```

Hitting the first button will produce the following result:

```
Line 1
---
Line 2
---
Line 3
---
```

And as we've defined the second callback list to stop on false, the second button will only print its message if the checkbox input is checked.

58. Make an image black and white

HTML5 brought us the canvas element. With it, we can manipulate the individual pixels of the image, apply transformations and draw shapes. One easy trick that is possible with the canvas element, is to make an image black and white. Here is how this is done:

(Note that for this to work, you will have to create a new HTML document that is opened through a locally running webserver like Apache. Accessing it directly with the file:// protocol will result in a security exception.)

HTML

```
<button id="btn">Make it black and white</button>
<canvas id="can" width="220" height="319" />
```

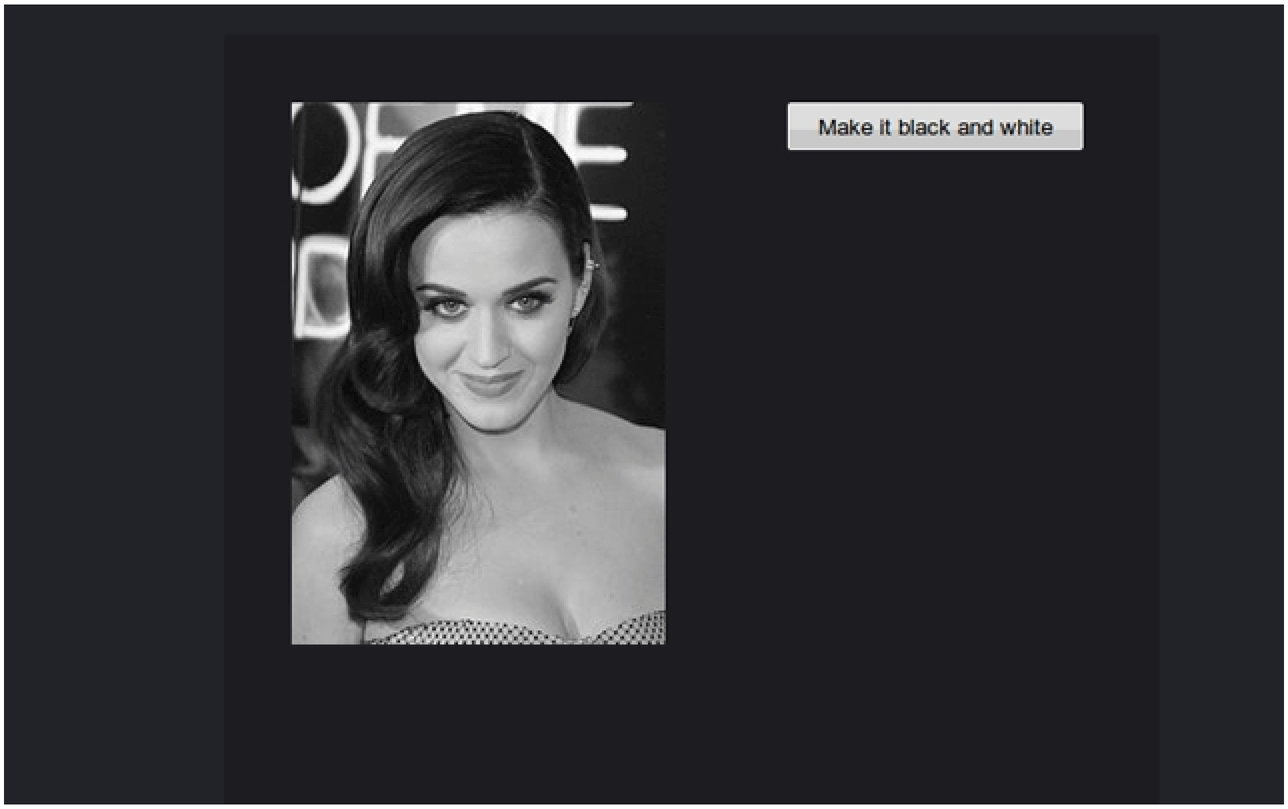
JS

```
// Define an image element and listen for the load event

var img = $('<img>');

var canvas = $('#can')[0],
     context = canvas.getContext('2d');

img.load(function(){

     context.drawImage(this, 0, 0);

});
```

```
img.attr('src','Katy_Perry.jpg');


// When the button is pressed, convert
// all the pixels to grayscale:

$('#btn').click(function(){

     var imageData = context.getImageData(0, 0, canvas.width, canvas.height),
            px = imageData.data, gray = 0;

    for (var i = 0, n = px.length; i < n; i += 4) {

        gray = (px[i] + px[i+1] + px[i+2]) / 3;

        px[i]   = gray;   // red
        px[i+1] = gray;   // green
        px[i+2] = gray;   // blue
    }

    context.putImageData(imageData, 0, 0);

});
```

First you draw an existing image onto the canvas with the **drawImage()** method. You then have read/write access to the individual pixels as elements of an array. For this example, we simply find the average value of the red, green and blue elements (the fourth is the alpha transparency, which we won't use). Setting the three of these colors to the same value will make the pixel gray scale.

The Mozilla Developer Network provides a great overview of the canvas element's features and JavaScript APIs.



59. Create a custom pseudo selector

The jQuery library gives you APIs that you can use to reach deep down and extend it with custom features. One of these is the custom selector API. Let's use it to define a selector that filters elements which contain specific text:

HTML

```
<p class="lipsum">Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
<p class="lipsum">Aenean dapibus turpis ut justo congue id sodales mi dignissim.</p>
<p class="lipsum">In sed lorem elit, sit amet mollis enim. Integer at feugiat
orci.</p>
<p class="lipsum">Integer dignissim, neque eu varius dignissim, risus lacus accumsan
tellus, eu rutrum felis turpis sit amet felis.</p>
<p class="lipsum">Proin tempus, tortor a sagittis auctor, urna felis ullamcorper
tellus, vitae consectetur dui ligula eget nisi.</p>
```

JS

```
// To create a custom selector, add it to the $.expr.pseudos property

$.expr.pseudos.icontain = $.expr.createPseudo(function(arg) {

    return function(elem) {
         return (elem.textContent
                  || elem.innerText
                  || jQuery(elem).text()
                  || '')
         .toLowerCase()
         .indexOf(arg.toLowerCase()) >= 0;
    };
});

// Here is how it is used:
console.log( 'Paragraphs that have the string "lorem" in them:',
$('.lipsum:icontain(lorem)').length );
```

Result

```
Paragraphs that have the string "lorem" in them: 2
```

Keep in mind that custom selectors can not use the browser's built-int querySelector() function calls, which makes them inherently slower than simpler queries.

60. Custom easing functions

Another customization that jQuery allows, is to extend its animation capabilities by providing a custom animation (easing) function. Here is how it is done:

HTML

```
<button id="btnShow">Show</button> <button id="btnHide">Hide</button>
<img id="football" src="ball.png"
style="position:fixed;left:50%;bottom:500px;opacity:0;z-index:1000" />
```

JS

```
jQuery.easing.easeOutBounce = function (x, t, b, c, d) {
    if ((t/=d) < (1/2.75)) {
        return c*(7.5625*t*t) + b;
    } else if (t < (2/2.75)) {
        return c*(7.5625*(t-=(1.5/2.75))*t + .75) + b;
    } else if (t < (2.5/2.75)) {
        return c*(7.5625*(t-=(2.25/2.75))*t + .9375) + b;
    } else {
        return c*(7.5625*(t-=(2.625/2.75))*t + .984375) + b;
    }
};

var ball = $('#football');

$('#btnShow').click(function(){

    // Pass the name of the easing function as a third argument
    ball.stop().animate({opacity:1, bottom: 0}, 1000, 'easeOutBounce');

    // Don't forget to stop() before starting animations
});

$('#btnHide').click(function(){

    ball.stop().animate({opacity:0, bottom: 500});

});
```

This easing function will give the animation a neat bounce effect. It was copied from the jQuery easing plugin. For a full list of easing functions and live previews, you can try easings.net.

61. (Mis)use jQuery's Animation Queue

Animations in jQuery are added to a queue and run in succession. Do you know you can add your own functions there?

HTML

```
<div id="elem" style="background:blue;width:20px;height:20px;
text-align:center;line-height:40px;"></div>
```

JS

```
// Fetch the animated element
var elem = $('#elem');

console.log('---');

// The current queue (an empty array):
console.log( elem.queue() );

elem.animate({width:100, height:40}, 1000);

console.log('---');
console.log( 'New length:', elem.queue().length );


// The trick: add a custom function to the queue.
// It will be executed after the animation finishes.

elem.queue(function(next){
     $.getJSON('1.json', function(r){
            elem.text(r.message);

            // Call next() so the queue proceeds
            // with the next animation:
            next();
     });
});

// This animation will be executed after the AJAX call above
elem.animate({borderRadius:20});
```

In the example above, I am placing an AJAX function inside the animation queue. It important to make the **next()** call, so that the queue proceeds with the next function. The result is that the animations wait for the json file to be downloaded and shown.

62. Custom CSS properties using hooks

jQuery gives you an API for creating your custom properties that will be recognized by the .css() method. Let's make a custom *rotate* property:

HTML

```
<div id="box" style="width:30px;height:30px;background-color:purple"></div>
```

JS

```
// Create a custom "rotate" property:

$.cssHooks["rotate"] = {
     get: function( elem, computed, extra ) {

            var transform = $.css(elem, 'transform') || '',     // this will return a
matrix(a,b,c,d,e,f)
                   parts = transform.match(/([\d\.\-]+(?:px)?)/g);

            if(!parts || !parts.length || parts.length != 6){
                   return 0;
            }

            return Math.round(Math.atan2(parts[1], parts[0]) * (180/Math.PI));

     },

     set: function( elem, value ) {

            $(elem).css('transform', 'rotate(' + parseInt(value, 10) + 'deg)');

     }
};
```

And here is how it is used:

```
$('#box').css('rotate', 20);
```

For your custom property to be recognized, you have to add it to the **$.cssHooks** array. The rotate property requires a lot of work, but you can probably think of more time savers that will be straightforward to write.

63. The jQuery $.proxy() method

Many beginners fall prey to JavaScript's **this** context. jQuery doesn't make it much easier as it sets the **this** of every event listener function to the originating element. With **$.proxy()**, you can work around this:

HTML

```
<button id="toggle"> Show/Hide Ball</button><br />
<img id="football" src="ball.png" style="display:none;" />
```

JS

```
// Listen for button clicks
$('#toggle').click( $.proxy(function(){

    // "this" would normally point to the button.
    // With $.proxy(), we are overriding it to point
    // to the football

    this.toggle();

}, $('#football') ));
```

The **$.proxy()** utility returns a new function which binds the **this** variable of the function passed as its first argument, to the variable passed as the second.

64. Pass callbacks to jQuery's methods

A number of jQuery methods can take a callback function instead of a regular string or number. Here are some examples:

HTML

```
<input id="input1" style="width:60px;" type="text" />
<input id="input2" style="width:60px;" type="text" />
<input id="input3" style="width:60px;" type="text" />

<p class="lipsum">Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
<p class="lipsum">Aenean dapibus turpis ut justo congue id sodales mi dignissim.</p>
<p class="lipsum">In sed lorem elit, sit amet mollis enim. Integer at feugiat</p>
```

```
<p class="lipsum">Integer dignissim, neque eu varius dignissim, risus lacus accum</p>
<p class="lipsum">Proin tempus, tortor a sagittis auctor, urna felis ullamcorpe</p>
```

JS

```
// Set the value of all input boxes to that of their ids.
// This will implicitly loop through all of the elements.

$('input[type=text]').val(function(){
    return this.id;
});

// Cache the paragraphs
var p = $('.lipsum');

// Hide the paragraphs that contain the string 'lorem'
p.filter(function(){
    return $(this).text().match(/lorem/i) != null;
}).hide();

p = p.filter(':visible');

// Add a number before each of the visible paragraphs
p.prepend(function(index){
    return (index+1) + ') ';
});

// Set odd/even paragraphs to different colors
p.css('color', function(index){
    return index%2 == 1 ? 'blue': 'green';
});

// Limit the length of the paragraphs
p.html(function(){
    var content = $(this).text();

    if( content.length > 80 ){
        return content.slice(0, 80) + '...';
    }

    return content;
});

// Is the first paragraph green?
console.log('Is it green:', p.first().is(function(){
    // Your browser might report green in different ways:
```

```
    return $(this).css('color').match(/green|rgb\(0, 128, 0\)|#00FF00/i);
}));
```

Passing a callback saves us from having to do complex iterations through the elements manually with the **$.each()** method.

VI. Plugins

Another factor that made jQuery so popular is the community of programmers that create and share awesome stuff. As a result, searching for high-quality plugins becomes an important first step when starting work on a new project. This chapter is dedicated to presenting a collection of cool libraries and plugins that will enhance the way you work with jQuery.

65. Test HTML5 support with Modernizr

Although not part of jQuery, the Modernizr library is an indispensable tool in the toolkit of the HTML5 developer. It let's you determine which features of the new standards does the visitor's browser support. This will help you decide whether to show the full experience or a fallback.

First, you need to include the Modernizr library in your page:

```
<script src="modernizr.custom.js"></script>
```

Then, you can test whether specific features are supported:

```
console.log('Custom fonts:', Modernizr.fontface);
console.log('Border radius:', Modernizr.borderradius);
console.log('Multiple Background Images:', Modernizr.multiplebgs);
console.log('CSS Animations:', Modernizr.cssanimations);
console.log('CSS Reflections:', Modernizr.cssreflections);
console.log('CSS 3D Transformations:', Modernizr.csstransforms3d);
console.log('Canvas:', Modernizr.canvas);
console.log('Drag&Drop:', Modernizr.draganddrop);
console.log('Audio:', Modernizr.audio);
console.log('IndexedDB:', Modernizr.indexeddb);
console.log('Web Workers:', Modernizr.webworkers);
console.log('Geolocation:', Modernizr.geolocation);
console.log('Touch Events:', Modernizr.touch);
console.log('Webgl:', Modernizr.webgl);
```

Result

```
Custom fonts: true
Border radius: true
Multiple Background Images: true
CSS Animations: true
CSS Reflections: true
CSS 3D Transformations: true
Canvas: true
Drag&Drop: true
Audio: true
IndexedDB: true
Web Workers: true
Geolocation: true
```

```
Touch Events: false
Webgl: true
```

When it loads, the library runs some fast feature detection tests and makes the results available as properties of the global **Modernizr** object. Note that it doesn't enable support for missing features. For this purpose, you can use a script loader and include appropriate shims in the page.

66. Bring media queries to jQuery

There is one additional feature of Modernizr that is worth mentioning - its ability to bring CSS3-like media queries to JavaScript / jQuery. This will let you write layouts that respond to the screen size and orientation of the device.

First, include the library:

```
<script src="modernizr.custom.js"></script>
```

You can now start parsing media queries:

```
var w = $(window);

w.on('resize',function(){

    if(Modernizr.mq('(max-width: 640px) and (orientation:portrait)')){
          console.log('iPhone portrait layout');
    }
    else if(Modernizr.mq('(max-width: 960px)')){
          console.log('Narrow layout');
    }
    else{
          console.log('Regular layout');
    }

});
```

Modernizr has other neat features that you can learn more about by visiting their site.

67. Speed up your site with a script loader

The more scripts you include in your page, the slower is your website to load. This is because scripts block the rendering of the page until they are downloaded, parsed and executed. This can be remedied by using a script loader like head.js.

First include head.js in your page. I am using the free Cloudflare CDN for extra speed:

```
<script src="http://cdnjs.cloudflare.com/ajax/libs/headjs/0.99/head.load.min.js">
</script>
```

This is the only script you will need to include as script tag, the rest will be requested through the library like this:

```
head.js(
     'http://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.9.2/jquery-ui.min.js',
     'http://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/2.2.2/bootstrap.min.js',
     'include.js',
     function(){
            // Everything is loaded! Let's print the message defined in include.js:
            console.log(message);
     }
);
```

The callback that we passed as the last parameter of the function will be executed once all the scripts are loaded. What is even better, is that scripts are loaded in a non-blocking way, asynchronously, which eliminates slow downs.

68. Render HTML with templates

Writing HTML by hand is cumbersome. Generating it through JavaScript is even more so. This is where template libraries like Mustache.js come into play.

Include the library in your page (or via a script loader, like the previous tip suggested):

```
<script src="mustache.js"></script>
```

You can then define a template. You can write your templates as regular JS strings and assign them to variables, but I will demonstrate a different approach:

```
<div id="people"></div>

<!-- Store the template inline -->
<script type="text/html" id="template">
    <ul>
        <li>Name: {{name}}</li>
        <li>Email: {{email}}</li>
        <li>Age {{age}}</li>
    </ul>
</script>
```

The template is defined inside a **text/html** script block. There is no such thing as a text/html scripting language, so browsers won't attempt to execute it, but at the same time won't render it as HTML and won't display it on the page. This makes it perfect for our needs – we only need to get the contents of this tag and pass it to the Mustache library:

```
// Get the template string
var template = $('#template').text();

// Define a list of people
var list = [{
    "name" : "Peter Parker",
    "email": "pp@gmail.com",
    "age"  : 23
},{
    "name" : "The Lizard",
    "email": "lizrd@flickr.com",
    "age"  : 44
},{
    "name" : "Mary Jane",
    "email": "mjay@gmail.com",
    "age"  : 22
}];

var people = $('#people');
```

```
// Compile the template for better performance.
// This turns it into a JavaScript function.

var compiled = Mustache.compile(template);

$.each(list, function(){
    people.append(compiled(this));
});
```

Result

```
* Name: Peter Parker
* Email: pp@gmail.com
* Age: 23

* Name: The Lizard
* Email: lizrd@flickr.com
* Age: 44

* Name: Mary Jane
* Email: mjay@gmail.com
* Age: 22
```

To render the templates, we simply pass an object to the compiled function. The result is a series of unordered lists with the objects' properties as list items. You can read more about the features of the library in the docs.

69. Give jQuery a utility belt

jQuery is the best choice to manipulate the dom and listen for events. But it falls short when it comes to manipulating data. For this, we have helper libraries like underscore.

Include the library into your document:

```
<script src="underscore-min.js"></script>
```

And start manipulating data:

```
// The variables we will be working on
var arr = [53, 2, 4, 6, 7, 13, 77, -23, 6, 11, 401];

var obj = {
    name: "Peter Griffin",
    age: 42,
    weight: "won't tell"
};

var func = function(name, message){
    console.log(name, 'says', message || this);
};

console.log('Odd elements:', _.filter( arr, function(num){ return num % 2 == 0; }) );
console.log('Does it contain 401?', _.contains( arr, 401) );
console.log('Max number:', _.max(arr) );
console.log('Sorted:', arr.sort(function(a,b){ return a-b}) );
console.log('Randomized:', _.shuffle(arr) );
console.log('Unique values:', _.unique(arr) );

console.log('---');

console.log('Object keys:', _.keys(obj) );
console.log('Values:', _.values(obj) );
console.log('Pick:', _.pick(obj, 'name', 'age') );
console.log('Size:', _.size(obj) );

console.log('---');

// Bind the function to a specific value of this

var binded = _.bind(func, 'hi!');

binded('Brian');
binded('Stewie');

// Allow the function to run only once
var throttled = _.once(func, 10);

var dwarfs = 'Bashful Doc Dopey Grumpy Happy Sleepy Sneezy'.split(' ');

for(var i=0; i<7; i++){
    throttled(dwarfs[i], 'he is hungry');
}
```

```
console.log('---');

// You can combine methods through chaining

console.log('Results:');

_.chain(arr).sortBy(function(val){return val})
                 .filter(function(val){ return val%2 != 0})
                 .each(function(val){ console.log(val) });
```

Result

```
Odd elements: [2,4,6,6]
Does it contain 401? true
Max number: 401
Sorted: [-23,2,4,6,6,7,11,13,53,77,401]
Randomized: [13,77,6,2,6,4,401,53,11,7,-23]
Unique values: [-23,2,4,6,7,11,13,53,77,401]
---
Object keys: ["name","age","weight"]
Values: ["Peter Griffin",42,"won't tell"]
Pick: {"name":"Peter Griffin","age":42}
Size: 3
---
Brian says hi!
Stewie says hi!
Bashful says he is hungry
---
Results:
-23
7
11
13
53
77
401
```

This only a limited overview of the things you can do with underscore. You should consult the documentation for more info.

70. Simplify JavaScript's date and time functions

Working with JavaScript's date and time functions is not the most enjoyable experience in the world. You can do pretty much anything with JavaScript's Date() object, but you will find yourself constantly searching for snippets or tutorials online even for the simplest things. But there is one neat library – moment.js – that makes things much better.

First you have to include it into your page:

```
<script src="moment.min.js"></script>
```

Then you can use the powerful **moment()** function:

```
// Create a new moment object
var now = moment();

// Create a moment in the past, using a string date
var m = moment("April 1st, 2005", "MMM-DD-YYYY");

// Create a new moment using an array
var m = moment([2005, 3, 1]);

console.log('What time is it?', moment().format('HH:mm:ss'));

var day = moment().day(); // 5
console.log('What day of the week is it?', moment.weekdays[day] );

console.log('What is the current month name?', moment.months[moment().month()] );
console.log('What time is it in London?', moment.utc().format('HH:mm:ss') );
console.log('What time is it in Japan?',
moment.utc().add('hours',9).format('HH:mm:ss') );
console.log('The next world cup will be ', moment('June 12th, 2014','MMM DD
YYYY').fromNow() );
console.log('What date will it be 7 days from now?',
moment().add('days',7).format('MMMM Do, YYYY') );

// Find the duration between two dates

var breakfast = moment('8:32','HH:mm');
var lunch = moment('12:52','HH:mm');
```

```
console.log( moment.duration(lunch - breakfast).humanize() + ' between meals' );
```

Result:

```
What time is it? 11:47:24
What day of the week is it? Monday
What is the current month name? February
What time is it in London? 09:47:24
What time is it in Japan? 18:47:24
The next world cup will be in a year
What date will it be 7 days from now? March 25th, 2013
4 hours between meals
```

The library can calculate time differences, format dates, parses time strings and more.

71. Use classical inheritance in JavaScript

JavaScript is the only major prototype-based language which makes it difficult for programmers with experience in other languages. But do you know that there are libraries that give you classes, constructors and inheritance much like with any regular object oriented language? Here is an example with the Class library, from the creators of jQuery.

After you include the library, you can proceed writing your classes as you normally would:

```
<script src="class.js"></script>
```

```
// Using the JavaScript inheritance class:
var Animal = Class.extend({
     init: function(name){
            this.name = name;
     },

     sound: function(snd){
            console.log(this.name + ' said ' + snd + '!');
     }
});
```

```
var Cat = Animal.extend({
    sound: function(){
        this._super('meow');
    }
});

var Dog = Animal.extend({
    sound: function(){
        this._super('bark');
    }
});

var kitty = new Cat('Snowball');
var doggy = new Dog('Frankenweenie');

kitty.sound();
doggy.sound();
```

Result:

```
Snowball said meow!
Frankenweenie said bark!
```

Classical inheritance can simplify your application design by giving you the building blocks you are already familiar with. You can take things one step further and even use a library like Backbone.js which gives you a full MVC implementation on the client side.

72. How to work with IndexedDB

Modern browsers have support for a new client-side database called IndexedDB (see which browsers provide support for it here). It is a NoSQL database with good performance that you can use to store large amounts of data in the browser and run queries against it. The database has a verbose API, which involves passing lots of callbacks around, but there is a small library called db.js that you can use to make things easier:

```
<script src="db.js"></script>
```

Let's say we have these two buttons, for listing and creating new records:

```
<button id="insertRecords">Insert Records</button>
<button id="listRecords">List Records</button>
```

We can then use the **db** object provided by **db.js**. It provides an interface similar to jQuery's deferred object:

```
// Define an array with people to be inserted
var arr = [{
     "email"      : "jack.sparrow@blackperl.com",
     "name"       : "Jack Sparrow",
     "phone"      : "555-123-567-333"
},{
     "email"      : "blackbeard@flyingdutchman.com",
     "name"       : "Black Beard",
     "phone"      : "555-423-567-441"
},{
     "email"      : "ljsilver@hispaniola.com",
     "name"       : "Long John Silver",
     "phone"      : "556-352-436-666"
}];

// Use the db.js library

db.open({
    name: 'test2',
    version: 1,
    schema: {
        people: {
            key: {
                keyPath: 'id',
                autoIncrement: true
            },
            indexes: {
                email: { unique: true } // emails should be unique
            }
        }
    }
}).done(function(server){

     // Listen for clicks on the buttons

     $('#listRecords').click(function(){
           server.people.query().filter().execute().done(function(results){
                   if(!results){
                        console.log('No records found!');
                   }

                   $.each(results, function(){
                        console.log(this);
                   });
           });

   });


   $('#insertRecords').click(function(){

           console.log('Inserted!');

           $.each(arr, function(){
                   server.people.add(this);
           });

   });

});
```

This saves us from having to listen for events, pass callback functions and open cursors. The IndexedDB standard was designed with the vision that a third party library will make it more user friendly, and I think that db.js does a great job at it.

73. jQuery and cookies

Cookies are the old and tested method of storing small amounts of data in the browser. They are sent along with the request headers, so they will be available in the server side. This is a clear advantage over the other methods of client-side storage like localstorage, sessionstorage and IndexedDB, which means that you will have to deal with them for the foreseeable future.

The only problem is that reading and writing cookies is cumbersome when done with JavaScript. However, there are jQuery plugins that make it easier. One of them is jQuery.cookie:

Include the plugin file after including jQuery into the page:

```
<script src="jquery.cookie.js"></script>
```

You then have access to a sane way of reading and setting cookies:

```
$.cookie('password', '12345', { expires: 7 });
```

This will keep the cookie for seven days. On every page load during this time, you can read the value of the cookie:

```
console.log('My password is:', $.cookie('password'));
```

You can also read all cookies at once, and delete cookies as is explained in this tutorial.

Note that you shouldn't under any circumstances store passwords in cookies! This is only for demonstration purposes.

74. Parse and print markdown

Markdown is a lightweight format for writing rich text on the web. It is used on sites like Github and StackOverflow for writing comments and in various third party applicaitons. Here is how to convert it to HTML with markdown.js:

```
<script src="markdown.js"></script>
```

Let's say that you have a commenting form on your site that accepts markdown:

```
<input type="text" id="md" value="This is a some **Markdown** text. You can give
_emphasis_, add [links](http://www.google.com/) and much more!" />
<button id="convert">Convert</button>
<p id="convertedHTML"></p>
```

Now, let's convert the contents of the text area into HTML:

```
// Cache some selectors
var convert = $('#convert'),
     convertedHTML = $('#convertedHTML'),
     md = $('#md');

// Listen for clicks on the button and parse the contents of the input

convert.click(function(){
     convertedHTML.html( markdown.toHTML( md.val() ) );
});
```

This will produce HTML that is inserted into the paragraph. It will turn the word *Markdown* bold, *emphasis* italic, and make *link* a hyperlink pointing to google.

75. Animate and work with colors

Out of the box, jQuery can't animate or manipulate colors. This is a useful feature to have if you need to create eye-catching animations. Luckily, the powerful Color plugin brings this and much more to jQuery:

```
<script src="jquery.color.js"></script>
```

HTML

```
<button id="animate">Animate</button>
<div id="rainbow" style="width:80px; height:80px;float:right;"></div>
```

JS

```
// Create a new color object
var c = $.Color("#12beff");

// Print the color components

console.log('Red:', c.red() );
console.log('Green:', c.green() );
console.log('Blue:', c.blue() );
```

```
$('#animate').click(function(){

    // Since we have included the jQuery.Color plugin,
    // we can animate the color properties

    $('#rainbow').css('background-color','white')
                 .animate({'background-color': 'blue'})
                 .animate({'background-color': 'green'})
                 .animate({'background-color': 'orange'})
                 .animate({'background-color': 'purple'});
});
```

In addition, the plugin supports blending colors, converting them into different formats and modifying the hue, saturation, lightness and more.

76. Upgrade jQuery with CSS3 animations

CSS3 transitions and animations are smoother and faster than the ones possible with JavaScript alone, but are not available in every browser. Use the jQuery animate enhanced plugin to upgrade the library to use CSS3 animations when possible.

```
<script src="jquery.animate-enhanced.js"></script>
```

HTML

```
<button id="animate">Animate</button>
<div id="square" style="width:80px;height:80px;
position:absolute;background-color:white;"></div>
```

JS

We have included the plugin in the page, so now the animate method will automatically make use of the CSS3 capabilities of this browser:

```
$('#animate').click(function(){
    $('#square').animate({
        width:200,
        height:200,
       left:40,
       top:20
    });

});
```

No changes need to be done to your end – you only need to include the library in your site and the rest will be done automatically for you.

77. Show full screen YouTube video backgrounds

Do you know there is a way to show full screen background videos hosted by YouTube? Here is how to do it with the help of a jQuery plugin called OKVideo:

```
<script src="okvideo.js"></script>
```

JS

```
// Use the okvideo plugin

$.okvideo({ video: 'http://www.youtube.com/watch?v=vp1Mb7n3cAM' });
```

Additionally, it supports videos from Vimeo, playlists, controlling the volume and choosing a quality.

78. Enhance your web app with better dialogs

The default browser dialogs like **alert()** and **confirm()** are ugly and horribly non-customizable. You can fix this with Alertify.js:

```
<link rel="stylesheet" href="alertify.core.css" />
<link rel="stylesheet" href="alertify.default.css" />
```

```
<script src="alertify.js"></script>
```

HTML

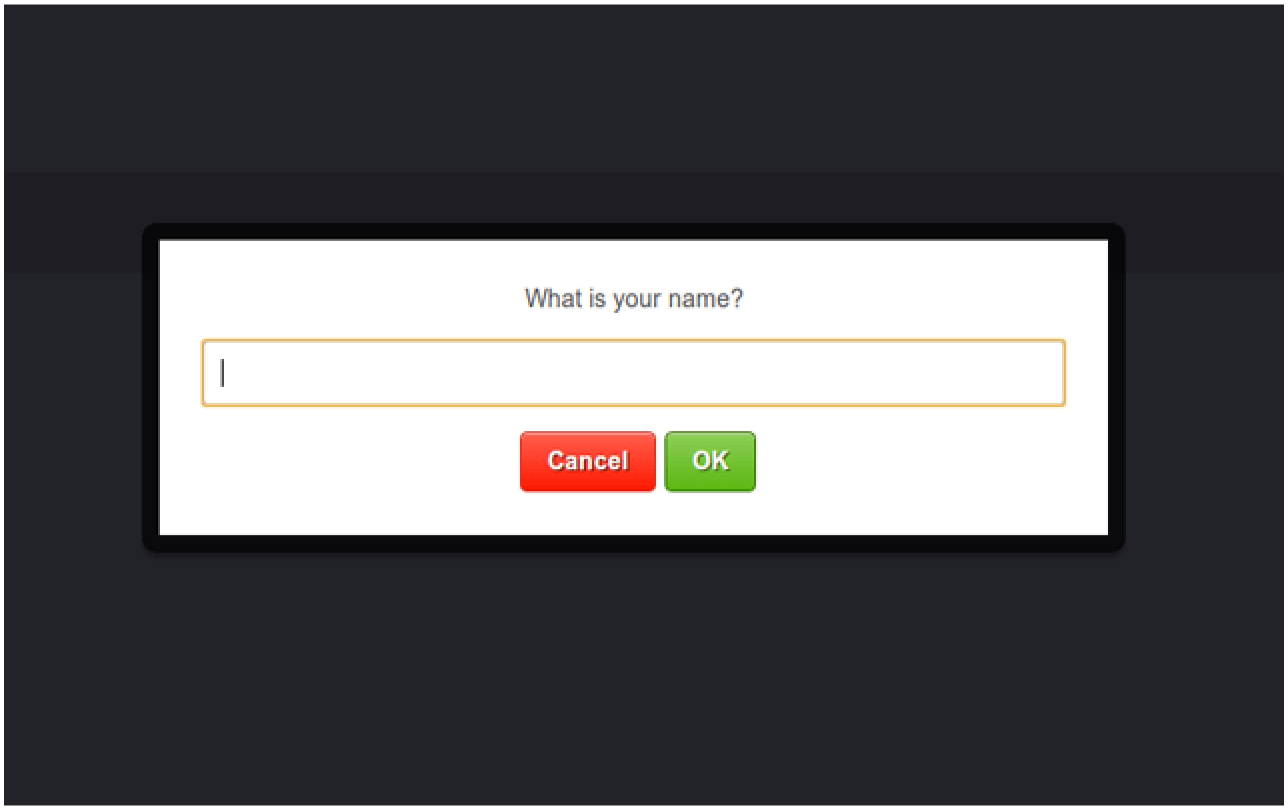
```
<button id="alert">Alert</button>
<button id="confirm">Confirm</button>
<button id="prompt">Prompt</button>
<button id="log">Log</button>
```

JS

```
// Listen for clicks on the buttons

$('#alert').click(function(){

     // Output a message
     alertify.alert("Hello!");

});


$('#confirm').click(function(){

     alertify.confirm("Make your choice",function(choice){
         console.log('You chose:', choice);
     });

});


$('#prompt').click(function(){

     alertify.prompt("What is your name?",function(choice, name){

         if(choice){
            console.log('Hello ', name, '!');
         }
         else {
            console.log("You are shy aren't you!");
         }

     });

});
```

```
$('#log').click(function(){

    alertify.log("This is a log message!");

});
```

Alertify provides pretty and customizable alternatives to all of the browsers' built in dialogs. It also lets you show log messages to the screen which can come in handy when you want to show unobtrusive notifications.



79. Notifications in your favicon

Another neat plugin is [Tinycon.js](). It can do something that you might have not thought was possible – it can display a number in your favicon:

```
<link rel="icon" href="favicon.ico" />
<script src="tinycon.js"></script>
```

JS

```
// Simple as this:

Tinycon.setBubble(17);
```

This would be a great addition to chat systems or weather apps.

80. Supercharge your select dropdowns

The Chosen plugin takes an ordinary select element, and adds a number of enhancements:

```
<link rel="stylesheet" href="chosen.css" />
<script src="chosen.jquery.min.js"></script>
```

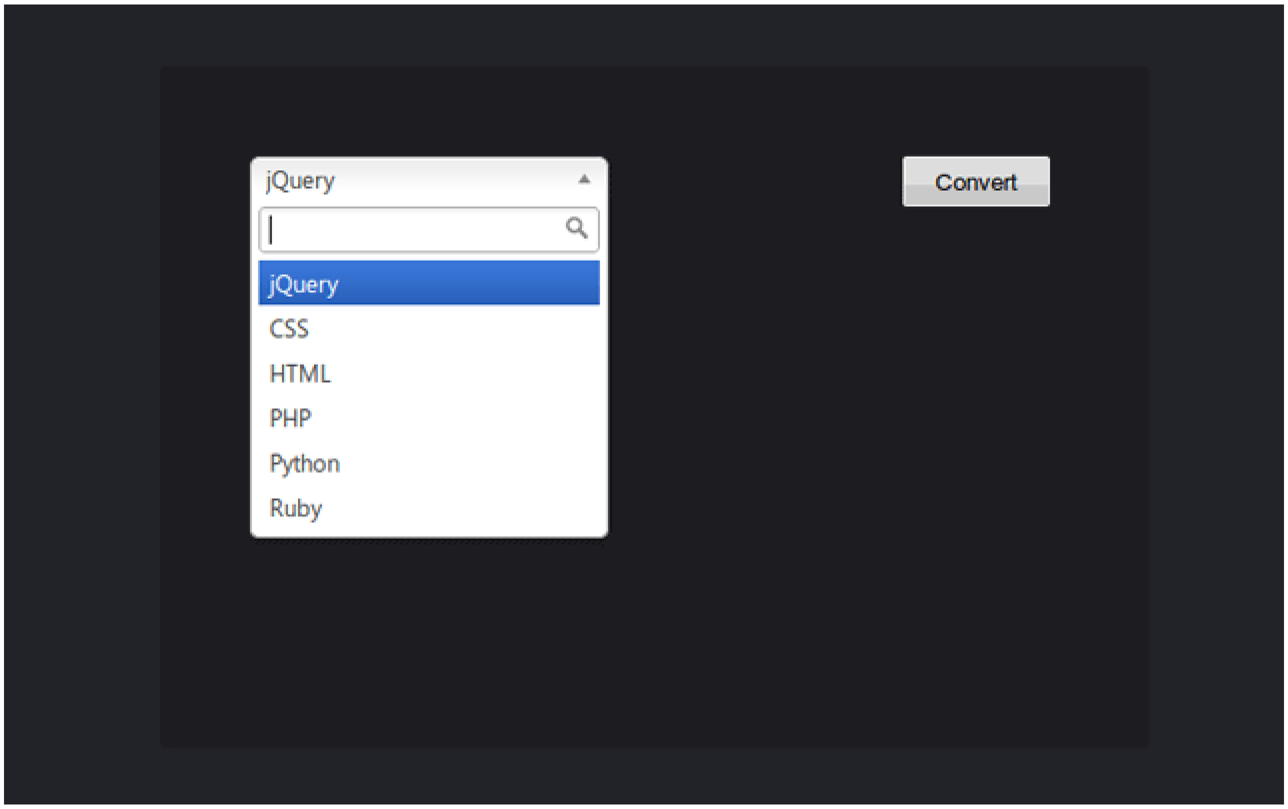
HTML

```
<select id="sel" style="width:200px;">
    <option value="jquery">jQuery</option>
    <option value="css">CSS</option>
    <option value="html">HTML</option>
    <option value="php">PHP</option>
    <option value="python">Python</option>
    <option value="ruby">Ruby</option>
</select>

<button id="convert" style="float:right">Convert</button>
```

JS

```
// When the button is clicked, supercharge the select item

$('#convert').click(function(){
    $('#sel').chosen();
});
```

This enhances the ordinary select dropdown with a beautiful design and real time option search . The original element is still present on the page, so forms and AJAX requests that depend on it will still work.



81. *Blur page elements with blur.js*

Here is another thing you wouldn't think is possible with jQuery. The Blur.js library applies a set of CSS/SVG filters so that it can blur elements of your page. What is more remarkable is that it even works in IE!

```
<script src="blur.js"></script>
```

JS

```
// Blur the entire page for 5 seconds

$('body').blurjs();

setTimeout(function(){
    $.blurjs('reset');
}, 5000);
```

You still can interact with the page as you would normally do, but the difference is that everything is blurred.

82. Generate QR Codes with jQuery

QR codes encode textual information in an image which you can scan with your smartphone. There is also a plugin for generating them with jQuery.

```
<script src="jquery.qrcode-0.2.min.js"></script>
```

HTML

```
<div id="qr" style="height:220px;padding:30px;background-color:#fff;
text-align:center"></div>
```

JS

```
// Simply call the plugin and pass some options:
$('#qr').qrcode({

    // Width an height of the code
    width: 200,
    height: 200,

    // QR code color
    color: '#000',

    // Background color, null for transparent background
    bgColor: null,

    // The encoded text
    text: 'http://www.google.com/'
});
```

And the result is a 200 by 200 pixel QR code generated on the fly.

83. Face detection with jQuery

JavaScript is a powerful and fast language. Here is one example in a friendly jQuery wrapper: detecting faces in a picture with the Face Detection jQuery plugin.

The first step is to include the relevant libraries:

```
<script src="facedetection/ccv.js"></script>
<script src="facedetection/face.js"></script>
<script src="jquery.facedetection.js"></script>
```

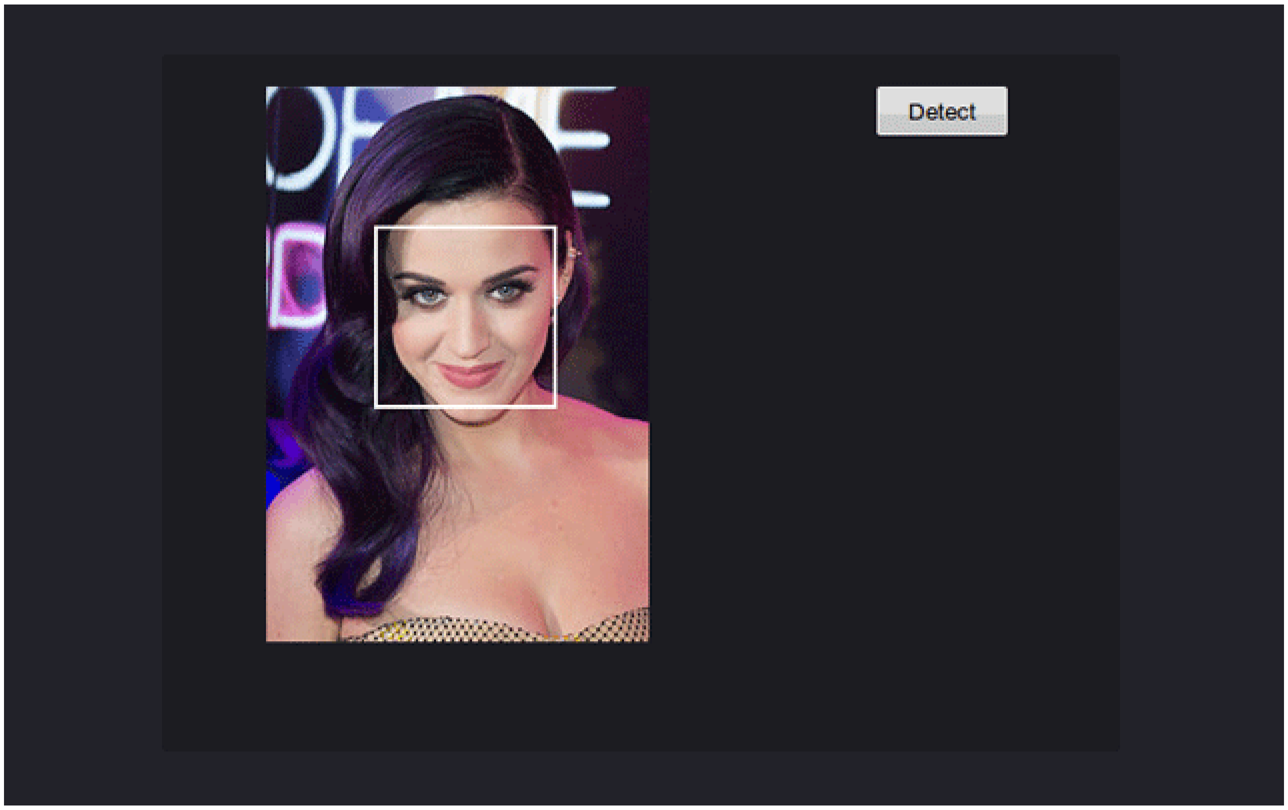
Then, we should define some markup and include the photo:

```
<button id="detect" style="float:right">Detect</button>
<div id="holder" style="position:relative;width:300px;">
    <img id="katy" src="Katy_Perry.jpg" />
</div>
```

When the button is clicked, we are going to try and detect faces in the picture with the plugin. We will then loop through the results and will draw a frame around each face.

```
// Listen for clicks on the button

$('#detect').click(function(){
    var coords = $('#katy').faceDetection();

    $.each(coords,function(){

            var frame = $('<div>').css({
                width:this.width,
                height:this.height,
                position:'absolute',
                top:this.y,
                left: this.x,
                border:'2px solid #fff'
            });

            $('#holder').append( frame );
    });
});
```

It is nice to see that something so advanced is possible with JavaScript alone.



84. Trigger full screen mode

Newer browsers have an API that allows your page to request to be shown in the full screen mode of the browser. Bigscreen.js is a small library that wraps around that API and makes it easier to use cross-browser:

```
<script src="bigscreen.min.js"></script>
```

One function call is all there is to it:

```
// Trigger the big screen plugin
BigScreen.toggle();
```

This will make your site take the full width and height of the screen. Keep in mind that browsers limit keyboard input and show nagging messages as a protective measure.

85. Manipulate the browser history

Another new feature provided by browsers is the history api, which allows your web app to modify the URL and record application state. Here is how to use it with the History.js library:

```
<script src="jquery.history.js"></script>
```

HTML

```
<p> Click the links below. After this,
use your browser's back/forward buttons to navigate
through the history. See how the URL changes and the
message paragraph gets updated.</p>

<a href="?page1" class="page">Page1</a>
<a href="?page2" class="page">Page2</a>
<a href="?page3" class="page">Page3</a>

<p id="message" style="padding:10px;"></div>
```

JS

```
// Cache some variables
var win = $(window),
    m = $('#message'),
    links = $('.page');

// Bind to StateChange Event
History.Adapter.bind(window,'statechange',function(){

    var state = History.getState();
    m.text(state.data.text);

});

var text = [
    'This is the first page',
    'This is the second page',
    'This is the third page'
];

links.click(function(e){
    var index = links.index(this);

    History.pushState(
        { text: text[index] },    // Data that will be available
        'Page ' + (index+1),      // Title of the page in the browser history
        this.href                 // The URL of the page to navigate to
    );

    e.preventDefault();
});
```

This way you can deliver a faster browser experience by fetching relevant pieces of data using AJAX, updating the page, and showing the new URL in the address bar. If this is something you would like to try, you might also want to look at the Pjax library that can do it for you.

86. Turn textboxes into rotating knobs

With the jQuery Knob plugin you can emulate a real rotating knob control. It supports touch which makes it a good fit for mobile web apps.

Include the library in your page:

```
<script src="jquery.knob.js"></script>
```

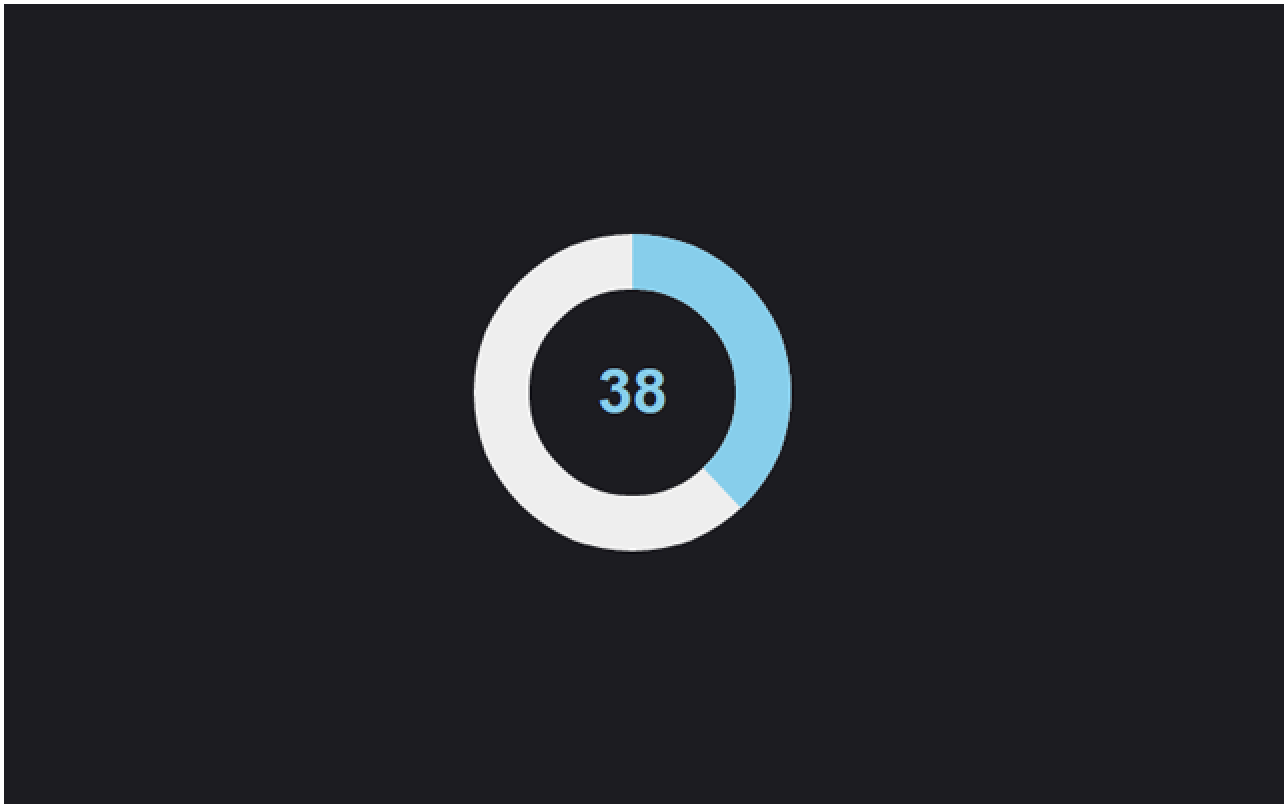
Then create a new text input element. You can provide min and max values for the knob:

```
<input type="text" id="dial" value="23" data-min="0" data-max="100" />
```

Then, simply call the plugin:

```
// Trigger the plugin with an input text field

$('#dial').knob();
```

You can fully customize the knob's colors, size and behavior.



87. Work with money and currencies

Many web apps need to work with and convert between multiple currencies with up-to-date exchange rates. There is an easy micro-library that makes this much easier – money.js.

```
<script src="money.js"></script>
```

To use the library, you will have to provide an up-to-date list of exchange rates, which you can obtain from an API like Open Exchange Rates. Here though I have cached the exchange rates in a JSON file:

```
// First, fetch the exchange rates

$.getJSON('rates.json',function(data){

     fx.rates = data.rates;
     fx.base = data.base;

     // Then, start converting
     console.log("100 USD in EUR:", fx(100).from("USD").to("EUR").toFixed(2) );
     console.log("2 GBP to USD:", fx(2).from("GBP").to("USD").toFixed(2) );
     console.log("123 EUR to RUB:", fx(123).from("EUR").to("RUB").toFixed(2) );

});
```

You need to pass an amount, source and target currencies and the library will handle the rest.

88. Easily display Google Maps

If you've worked with the Google Maps JavaScript API before, you know that it requires writing a lot of code even for the simplest things. There is a micro library that makes things much easier – gmaps.js.

Gmaps.js doesn't replace the Google Maps JS API, but builds upon it, which is why you need to include both in the page:

```
<script src="http://maps.google.com/maps/api/js?sensor=true"></script>
<script src="gmaps.min.js"></script>
```

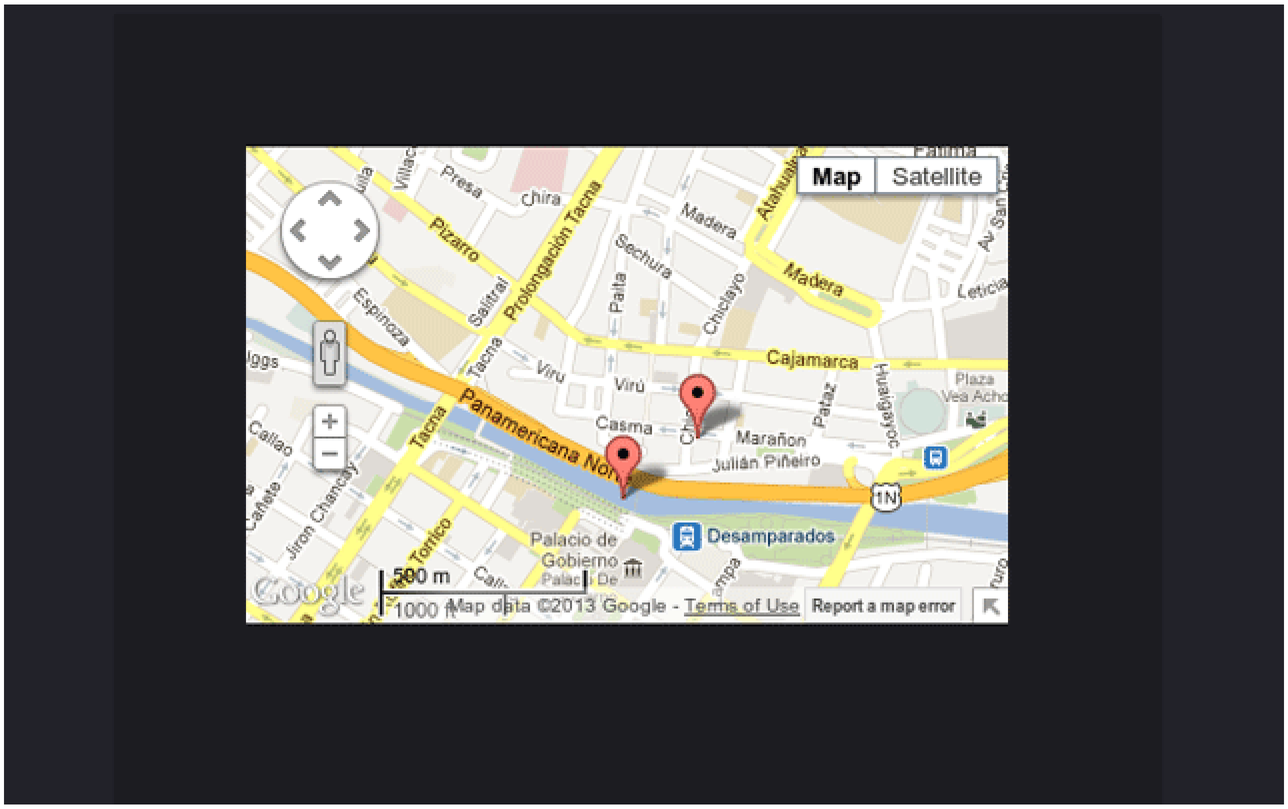
This is your map's placeholder:

```html
<div id="map" style="width:400px;height:250px;"></div>
```

Using gmaps.js is straightforward:

```javascript
// Create a map instance, and add two markers
var map = new GMaps({
     div: '#map',
     lat: -12.043333,
     lng: -77.028333
});

map.addMarker({
     lat: -12.043333,
     lng: -77.03,
     title: 'Lima',

     click: function(e){
            alert('You clicked in this marker');
     }
});

map.addMarker({
     lat: -12.042,
     lng: -77.028333,
     title: 'Marker with InfoWindow',
     infoWindow: {
            content: '<p>HTML Content</p>'
     }
});
```

This will show a Google map in the container with two markers at specific coordinates.



89. jQuery and Web Workers

Web worker threads are separate processes from the page which you can do computationally intensive operations in. There is a small wrapper library that integrates them with jQuery as a plugin:

```
<script src="jquery.worker.js"></script>
```

To initialize a worker, pass the name of a JavaScript file. It will be started in a separate process and you will be able to communicate with it using jQuery's deferred mechanism:

```
var worker = $.worker({ file:'worker.js' });

worker.progress(function(){
    // Will output the same message that we sent it
    console.log('Received message:', this.data);
});

worker.postMessage('Hello there!');
```

The **postMessage** is an additional method that you can use to send messages to the worker. In this case worker.js is a simple echo script, which prints whatever it receives:

```
self.onmessage = function(e) {
   self.postMessage(e.data);
   self.close();
};
```

Result

```
Received message: Hello there!
```

Workers are a solution to a very specific problem – doing computationally expensive operations which block the user interface of your app.

90. Mask input fields

You may wish to allow only specific groups of characters to be entered in a text field. Here is how to do it with the jQuery input mask plugin:

```
<script src="jquery.maskedinput.js"></script>
```

HTML

```
<input type="text" id="date" style="margin-bottom:8px" /> <label>Date</label>
<input type="text" id="phone" style="margin-bottom:8px" /> <label>Phone</label>
<input type="text" id="tin" style="margin-bottom:8px" /> <label>TIN</label>
<input type="text" id="ssn" style="margin-bottom:8px" /> <label>SSN</label>
<input type="text" id="color" /><label>Color</label>
```

JS

```
// Set up some input masks

$("#date").mask("99/99/9999");
$("#phone").mask("(999) 999-9999");
$("#tin").mask("99-9999999");
$("#ssn").mask("999-99-9999");

// Custom definition
$.mask.definitions['h'] = "[A-Fa-f0-9]";

// allow only hash values to be entered
$("#color").mask("#hhhhhh");
```

This will prevent users from entering values that do not conform to the pattern you've chosen.

91. Lightweight form validation

There are many form validation plugins and libraries, but most of them are cumbersome and difficult to use. This is not the case with validate.js:

```
<script src="validate.js"></script>
```

HTML

```
<form id="clientInfo">
    <label for="fullname">Full Name * :</label>
    <input type="text" id="fullname" name="fullname" />

    <label for="email">Email * :</label>
    <input type="text" id="email" name="email" />

    <label for="message">Message (20 chars min, 200 max) :</label>
    <textarea id="message" name="message"></textarea>

    <button type="submit">Submit form</button>
</form>
```

JS

```
// Declare how the fields should be validated

var validator = new FormValidator('clientInfo', [{
    'name': 'fullname',
    'display': 'name',
    'rules': 'required'
}, {
    'name': 'email',
    'rules': 'valid_email|required'
}, {
    'name': 'message',
    'rules': 'min_length[20]|max_length[200]'
}], function(errors) {

    if (errors.length > 0) {

        var messages = '';

        $.each(errors,function(){

               messages += this.message + '\n';

         });

        // Normally, you would print these errors next to the
        // form inputs, but we are simply alerting them here:

        alert(messages);

    }

});
```

You also get the id and name attributes of the offending field, which you can use to display an inline error message. Try it by doing a **console.dir(this)** before the alert.

92. Work with retina graphics

To make your websites presentable on devices with high dpi screens, you need to use higher resolution imagery. One technique is to swap the regular image with a higher resolution one with a jQuery plugin like jQuery Retina:

```
<script src="jquery.retina.js"></script>
```

Here is the HTML of the image. The smaller image is set as a source. The higher-res version is set as the value of the data-retina attribute:

```
<img id="nature" src="http://farm5.staticflickr.com/4032/5077495230_7d9f6b1121_n.jpg"
data-retina="http://farm5.staticflickr.com/4032/5077495230_7d9f6b1121_z.jpg" />
```

Converting the image is as simple as this:

```
$('#nature').retina();
```

The image that the data-retina attribute points to is twice the width and height of the regular one (for a total of 4 times the number of pixels). If this is a retina device, the larger image will be loaded instead of the smaller.

93. Instagram-like filters with JavaScript

Another testament to the power of JavaScript and HTML5 is the Caman.js library. It is a powerful collection of canvas manipulation effects that you can use to create Instagram-like filters.

The first step is to include the library in your HTML. It doesn't depend on other third party libraries, so you could use it without jQuery:

```
<script src="caman.full.min.js"></script>
```

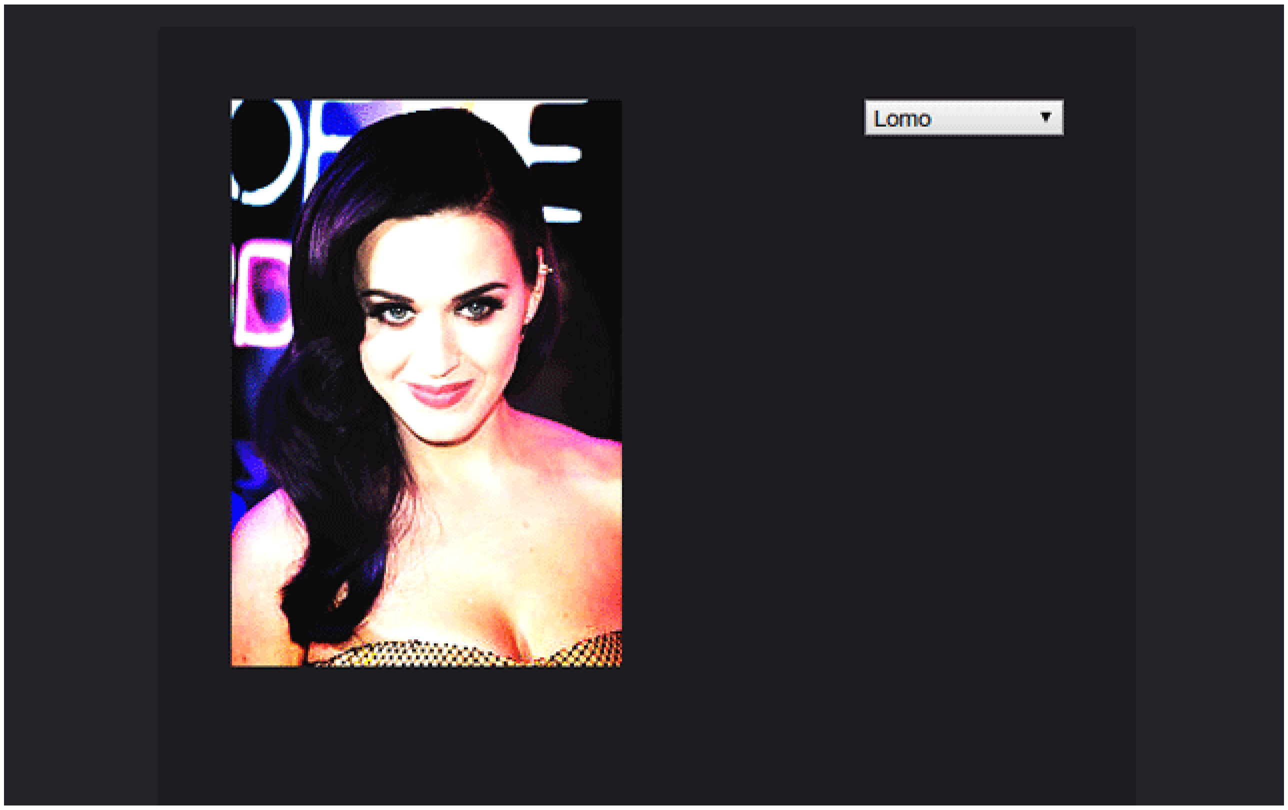
Next, we need a list of all the filters that caman has built-in:

```
<select id="filters" style="float:right">
    <option>Choose a filter</option>
    <option value="vintage">Vintage</option>
    <option value="lomo">Lomo</option>
    <option value="clarity">Clarity</option>
    <option value="sinCity">Sin City</option>
    <option value="sunrise">Sunrise</option>
    <option value="crossProcess">Cross Process</option>
    <option value="orangePeel">Orange Peel</option>
    <option value="love">Love</option>
    <option value="grungy">Grungy</option>
    <option value="jarques">Jarques</option>
    <option value="pinhole">Pinhole</option>
    <option value="oldBoot">Old Boot</option>
    <option value="glowingSun">Glowing Sun</option>
    <option value="hazyDays">Hazy Days</option>
    <option value="herMajesty">Her Majesty</option>
    <option value="nostalgia">Nostalgia</option>
    <option value="hemingway">Hemingway</option>
    <option value="concentrate">Concentrate</option>
</select>
```

All that is left now is to listen for changes on the dropdown, create a fresh canvas element and apply the Caman filter on it.

```
// Initialize some variables

var dropdown = $('#filters'),
    canvas = $();

dropdown.change(function(){

    // Create a canvas element and insert it into the page
    var tmp = $('<canvas>').insertAfter(dropdown).hide();

    // Get the current value of the dropdown
    var filter = this.value;

    Caman(tmp[0], 'Katy_Perry.jpg', function(){

        // Remove the canvas element from the previous operation
        canvas.remove();
        canvas = tmp.show();

        // If such a filter exists, use it
        if(filter in this){
            this[filter]().render();
            }
        });

});

// Show the regular image on load
dropdown.trigger('change');
```

The Caman library can do all sorts of adjustments on an image. You can even extend the library with your own filters. For an even cooler example, you can check out this post from Tutorialzine.



94. jQuery keyboard shortcuts

Every modern web app worth its salt has to support keyboard shortcuts. One neat library that will help you with that is keymaster.js:

```
<script src="keymaster.js"></script>
```

After you include the library in your page, you can bind event listeners for key combinations:

```
// Set up your key bindings

key('ctrl+s', function(){
     console.log('Saving!');

     // Prevent the default browser event
     return false;
});

key('ctrl+r', function(){

     console.log('No reloads for you!');
     return false;

});

key('ctrl+o', function(){
     console.log('Showing an open dialog');
});

key('d', function(){
     console.log('Deleting messages!');
});

console.log('Try presing some keys!');
```

Returning false from within a listener will prevent the default browser event. In this case, I am disabling the **Save as** popup dialog as well as the page reload shortcut.

95. Inflate web type

Fittext.js is a jQuery plugin that resizes the font-size of an element depending on the width of its container. This can come handy when making responsive designs.

Include the library in your page:

```
<script src="fittext.js"></script>
```

The element you want to expand:

```
<p id="type">Enjoy the jQuery Trickshots book!</p>
```

Trigger the plugin:

```
$('#type').fitText();
```

Resize your browser window to test it. The text will fill the element's entire width. Combine this with media queries for a fully responsive effect.

96. JavaScript and web cameras

One of the most exciting new browser features is WebRTC – a protocol for real time conferencing entirely in the browser. This is made possible with a number of new JavaScript APIs amongst which is getUserMedia. This API gives you access to devices like cameras and microphones attached to your computer.

Of course, this won't work in older browsers. For this purpose we have the getUserMedia shim. In addition to squashing browser differences, it also provides a fallback using Flash and gives you access to the computer's webcamera on nearly any browser out there.

To use it, include the shim into your page:

```
<script src="getUserMedia.js"></script>
```

Then create a holder for the video stream from the camera:

```
<div id="webcam"></div>
```

Lastly, write this code:

```
// The camera feed options
var options = {
     "audio": false,
     "video": true,
     el: "webcam",

     // The plugin supports a flash fallback for IE. We won't be using it
     noFallback:true,

     width: 320,
     height: 240,

     mode: "callback"
};

getUserMedia(options, success, error);

function success(stream) {
     var video = options.videoEl;

     // Firefox
     if (( typeof MediaStream !== "undefined" && MediaStream !== null) && stream
instanceof MediaStream) {

            if (video.mozSrcObject !== undefined) { //FF18a
                   video.mozSrcObject = stream;
            } else { //FF16a, 17a
                   video.src = stream;
            }

            return video.play();

     } else { // The rest
            var vendorURL = window.URL || window.webkitURL;
            video.src = vendorURL ? vendorURL.createObjectURL(stream) : stream;
        }

        video.onerror = function () {
            stream.stop();
            streamError();
        };

    }

    function error(){
        console.log('No web cam found!');
    }
```

If you try this code on a computer with a web cam, you will see a video feed in the #webcam div.

97. Responsive date picker

With the proliferation of mobile devices, it is getting more and more important to make your interfaces touch-friendly and usable on any kind of screen size. Here is one plugin that can will help you create responsive datepickers – Pickadate.js.

```
<script src="pickadate.min.js"></script>
```

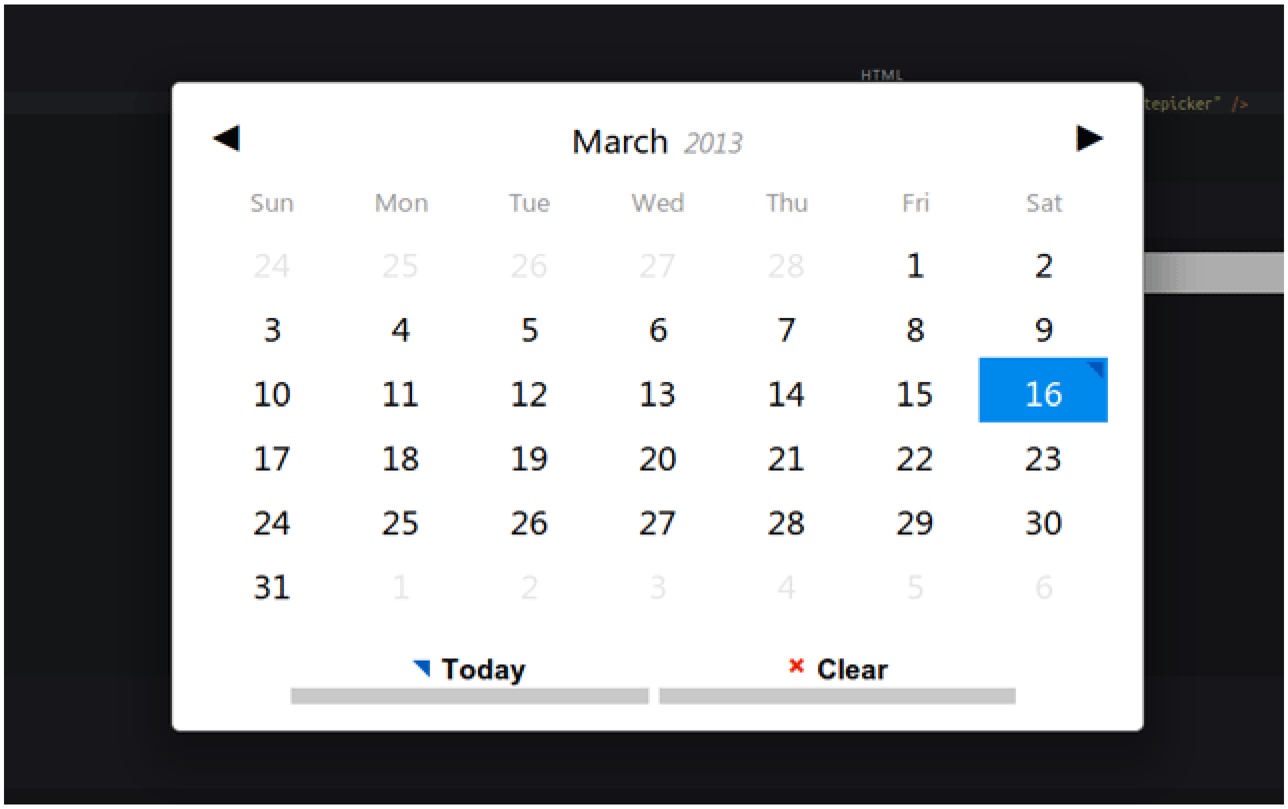
HTML

```
<input type="text" id="datepicker" />
```

JS

```
// Trigger the plugin.

$('#datepicker').pickadate();
```

Focusing the text input will show a pretty date picker on your screen.



98. *Handling file drag&drop in the browser*

Another cool feature that modern browsers provide is the ability to drag and drop files from your desktop onto the browser. The API itself involves listening for a number of events and things can get messy if you want to allow only specific types of files. However there is a library named Filereader.js, that makes things easier.

Let's say that we want to allow users to drop a photo from their computer into the browser, and display it directly without having to upload it. The first step is to include the library in the page:

```
<script src="filereader.js"></script>
```

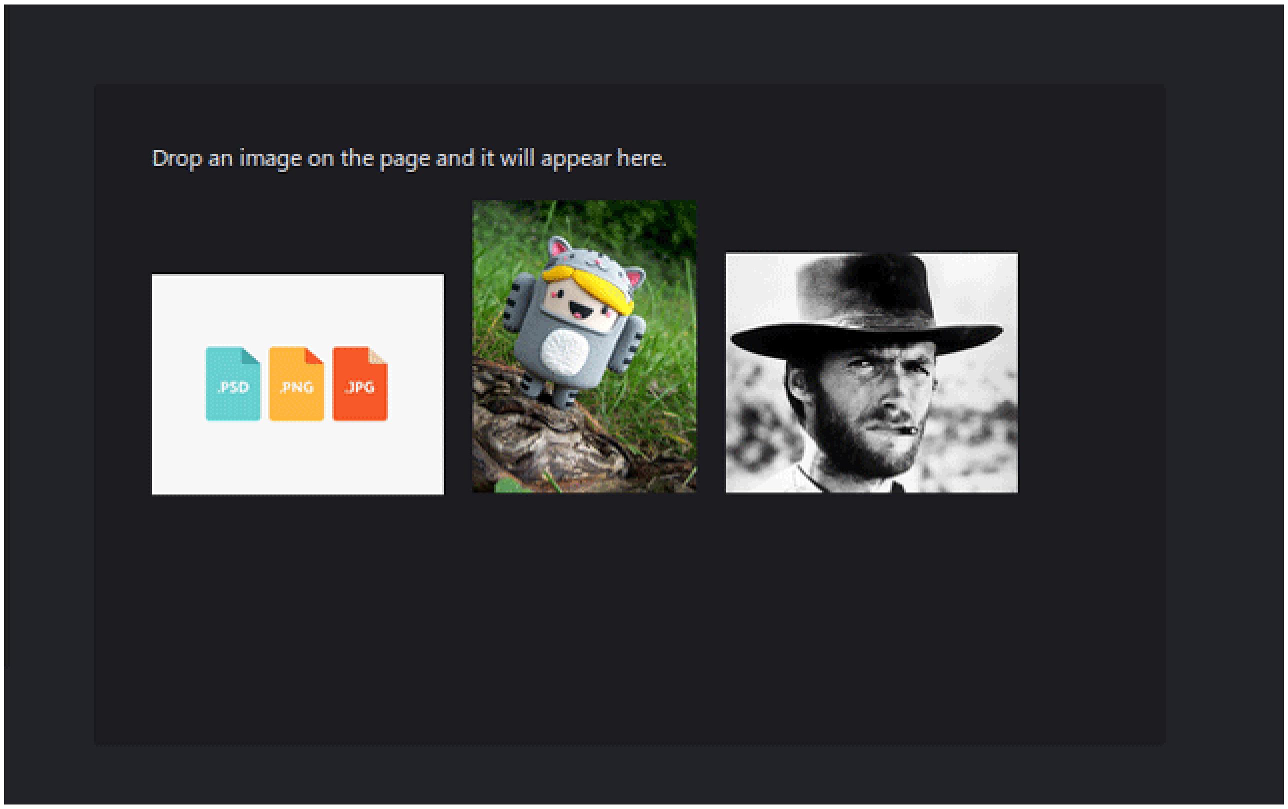
Then we should define a div that will hold the images after they are dropped:

```
<div id="pics"></div>
```

We can then use the filereader library to choose which element will accept the file drop event, and to display the photo:

```
// Call and configure the plugin

var pics = $('#pics');

$('body').fileReaderJS({
    on:{

        load: function(e, file){

            var img = $('<img>');
            img.load(function() {

                img.css({
                    maxWidth:150,
                    maxHeight:150
                });

                pics.append(img);
            });

            img.attr('src', e.target.result);
        },

        beforestart: function(file){
            // Allow only images
            return /^image/.test(file.type);
        }
    }
});
```

The beforestart function tests the type of the file and discards anything that is not an image by returning false. The result is that we are displaying every image dropped onto the page in the #pics div, resized to a maximum of 150x150 pixels.


Drop an image on the page and it will appear here.

99. Rumble elements with jRumble

jRumble is a small jQuery plugin that might not be the most useful thing in the world, but it sure is fun. Include it into your page:

```
<script src="jquery.jrumble.1.3.min.js"></script>
```

Then create the element that you want to “rumble”:

```
<div id="rble" style="width:200px;height:200px;background:#fff"></div>
```

Now get to work:

```
var element = $('#rble').jrumble({
     speed: 20,
     x:1,
     y:2,
     rotation: 1
});

element.trigger('startRumble');
```

To stop the effect:

```
element.trigger('stopRumble');
```

100. Dynamic charts

The xCharts library is a powerful tool for creating beautiful, animated, AJAX- enabled charts. To use it, include the stylesheet and relevant JS files in your page:

```
<link rel="stylesheet" href="xcharts.css" />

<script src="d3.v3.min.js"></script>
<script src="xcharts.min.js"></script>
```

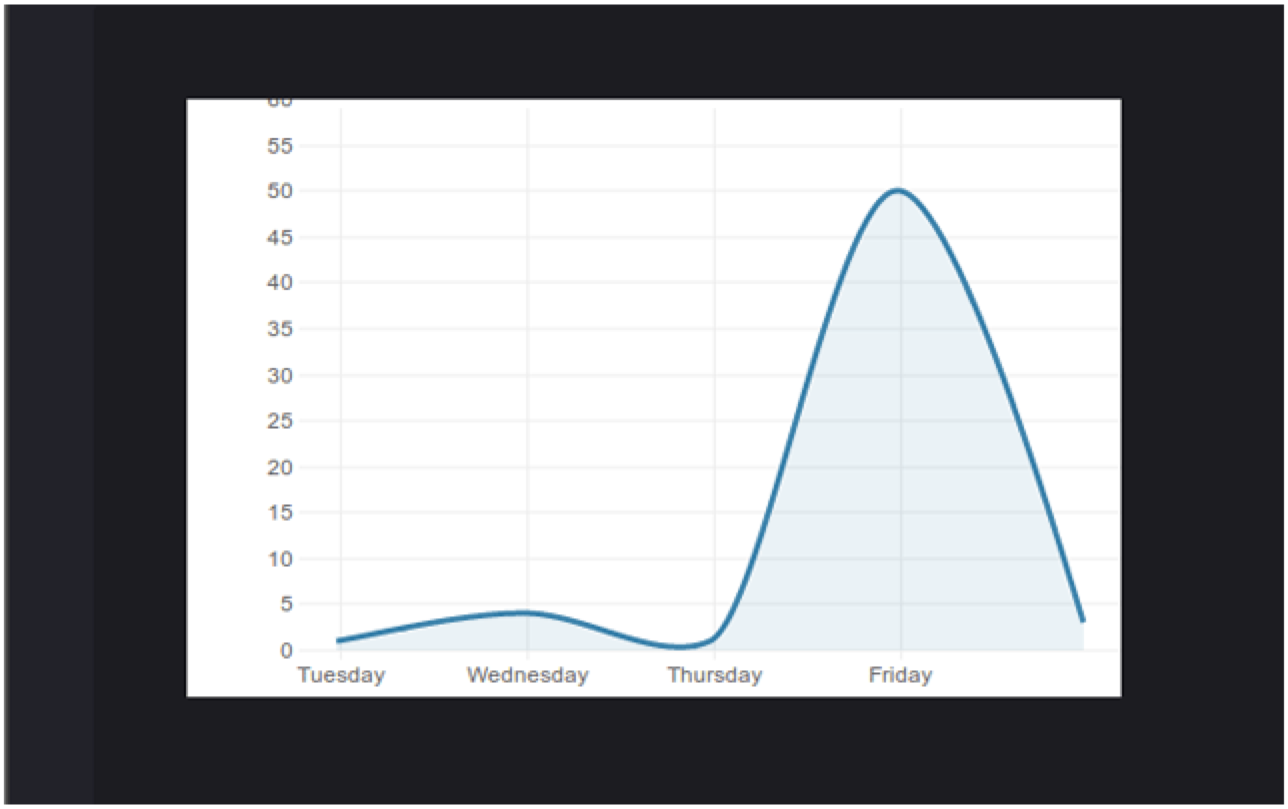
Then, create the element that will host the chart. Be sure to specify a width and a height:

```
<div id="chart" style="width:450px;height:250px;background:#fff"></div>
```

Then, create the chart:

```
// Generate the data set

var set = [];

for(var i = 0; i<5;i++){
    set[i] = {
        x: new Date( (new Date()).getTime() - i*24*60*60*1000 ),
        y: Math.round(Math.random() * 50)
    };
}

var data = {
  xScale: "time",
  yScale: "linear",
  type: "line",
  main: [
    {
      className: ".chart-identifier",
      data: set
    }
  ]
};

var opts = {
  tickFormatX: function (x) { return d3.time.format('%A')(x); }
};

// Create the chart
var myChart = new xChart('line', data, '#chart', opts);
```

We are generating random data in this case, but you can easily hook it up with an AJAX call (like we did in this Tutorialzine article). You can use the resulting chart to enhance your web applications and show pretty visualizations of your data.


55
50
45
40
35
30
25
20
15
10
5
0
Tuesday
Wednesday
Thursday
Friday

Summary

And this, dear reader, concludes our awesome trip. There are no doubt thousands more cool things one can do with our favorite library and we would love to hear them. We hope that what you read here will help enhance your understanding of jQuery, improve your day-to-day work and impress your friends.

For more awesome tips, tricks and tutorials, be sure to visit our site – Tutorialzine.com.

- Team Tutorialzine