# *jQuery*

## 1. Overview

### 1.1 What is JQuery

- jQuery is a freely available Open Source JavaScript Library.

- Its purpose is essentially to simplify the task of creating modern web pages that are highly interactive and responsive, essentially full of all the Web 2.0 goodness that you see all over the internet today.

- It works across all the modern browsers. One of the great things about jQuery is that it abstracts away a lot of the browser-specific features, which allows you to concentrate on the design and finished result that you are trying to achieve rather than spending all your time trying to figure out how to achieve various effects and features within individuals browsers.

- jQuery focuses on doing a few things very well. For example:

1. jQuery makes it really easy to get and manipulate page content. Normally you do this using the Document Object Model or the DOM.

2. jQuery also makes it really simple to work with the modern browser event model, which is a key part of building modern Web 2.0 applications.

3. jQuery adds a pretty sophisticated effects library that you can use to achieve a lot of sophisticated transitions and effects that you see in modern websites today, things like image fades and animations and events … etc.

- If you take a look at how most modern web development is done today, there are a couple of scenarios that seems to pop up over and over again. They typically follow a couple of very common patterns.

1. The first pattern is usually when the page loads in the browser, there is a bunch of setup that you have to do, and this is usually in response to the window load event, and jQuery provides a really efficient way of handling this.

2. The second scenario that pops up a lot looks something like this. There is an ***event*** that happens on the browser. The user clicks on something or they type a key or they move the mouse over something, and that sets off a reaction, something along the lines of, 1 you retrieve the content from the page, 2 then you do some kind of manipulation on it. Maybe you animate it somehow and 3 then you put the content back in the page.

Now this may not be followed exactly, but the pattern is essentially along these lines where the user does something, and in response to that event, you have to somehow get content and manipulate it.

- jQuery leverages your existing knowledge of CSS. You'll find yourself writing syntax that looks a lot like CSS code.

- jQuery is also built from the ground up to work with sets of elements. This is something that Document Object Model in JavaScript is not designed to do very well. jQuery makes it second nature. In fact, many times you won't even realize that you are writing code that just by default works across sets of elements.

- jQuery also allows you to perform multiple operations on a set of elements with just one line of code, and this is a very powerful feature known as ***Statement Chaining***. It really simplifies the way that you work with JavaScript and elements in the page.

- jQuery hides away a lot of the various browser quirks and this allows you to concentrate on your end result. So you don't spend a lot of time trying to figure out how to achieve each effect within each individual browser. You simply write to the jQuery interface and jQuery handles all of the individual browser features behind the scenes.

- jQuery is really extensible. There are a lot of plug-ins available that perform all kinds of web development tasks. In fact, it's pretty easy to learn how to write your own, but in many cases you won't have to. The jQuery website contains hundreds of plug-ins that have written by other people, that you can just freely download and use in your own pages.

- jQuery is currently compatible with modern versions of all the main browsers that are in use today. So you can use jQuery across Internet Explorer, Safari, Chrome, Firefox and Opera.

## 1.2 Downloading and Installing jQuery

There are two different versions of jQuery. There is the compressed production version, and then there is the uncompressed development version. Essentially, what you do is you usually download both of these and you use the Development version while you're doing your development, and then when you're ready to upload your finished project to your web server, you use compressed production version.

The reason you do that is because the compressed, production version is minified and compressed, and it's a lot faster to download to your user's computer than the uncompressed development version. The uncompressed development version is also a lot more readable, so if you're debugging, you can step through the code a lot easier.

# 1.3 Creating a Simple jQuery-Enabled Page

**Example:** create a simple jQuery enabled page that going to do three things:

1. Include a reference to the jQuery library.

2. Event handler is going to respond to the page loading up in the browser window.

3. Display an alert that indicates the event handler was in fact called and jQuery is therefore working.

So, before we do that, let's take a look at how you do this without jQuery.

- Typically, code that you want to have executed when the page is loaded is written something like write a ***runOnLoad*** function and that displays an alert and you can see I am triggering that off the ***window.onload*** event.

- the problem is that the window's ***onload*** event only fires after all of the page content has been downloaded including any images that you have in the page.

- It's also a lot harder to add multiple independent ***onload*** functions because if you want to have more than one function run on the window ***onload*** event, you have to cram them all into one function or you have to use the DOM event model to do that, which is different among browsers.

```
function runOnLoad() {
   alert("Hello World");
}
window.onload = runOnLoad;
```

- jQuery provides a way to run the code when the DOM of the page is actually ready, regardless of whether all of the content such as images has been downloaded, and that's called the ***document.ready*** event.

- It also makes it a lot easier to write modular independent ***on-load*** functions, and jQuery will just chain them together.

So, let's see how to do this the jQuery way.

- You write a ***document.ready*** event handler. The little dollar sign character, that's the jQuery object, and that's where all of jQuery's functionality is accessed from.

So what I am doing here is writing the jQuery object and calling the function with a parameter of ***document*** and then ***.ready***, and I am passing it an anonymous function that will display an alert when the page is loaded. However, you can also just do this the old-fashioned way. You can write an independent function, give it a name, and then pass the name of that function. I am just doing it this way because it's a bit more succinct and concise.

```
<head>
   <title>First jQuery-Enabled Page</title>
   <script type="text/javascript" src="../jquery-1.9.1.js"></script>
   <script type="text/javascript">
      $("document").ready(function() {
         alert("The page just loaded!");
      });
   </script>
</head>
```

- The code will now execute when the DOM of the page has loaded and is ready for use rather than just waiting for all the page content to finish downloading.

- another advantage of this approach is that you can call the ***document.ready*** function multiple times with individual functions that you want executed and jQuery will just chain each one together to be called in succession. So that allows you to write much more modular code to run when the page gets loaded and they can be contained within separate JavaScript files and so on.

# 1.4 Overview of Features in jQuery

The features of jQuery break down across 8 major categories:

1. The core functionality category, implements the core jQuery functions as well as some commonly used utility methods.

2. The Selection and Traversal category includes functions for extracting content from documents and navigating among the contents of the document. This is a lot of what the DOM is usually used for in modern web development. And it's a very key piece of jQuery's power.

3. The manipulation and CSS category includes functions for editing and changing document content and working with CSS data, such as positioning information and so on.

4. The Events category simplifies working with the modern DOM events and provides a lot of common event helper functions that cover some common scenarios that pop up in interaction design in web pages.

5. The Effects category contains functions for creating some basic animations and effects such as hiding and showing elements, moving things around, fading things in and out, that kind of stuff.

6. the AJAX category include utilities for working with AJAX, such as loading content from remote pages, dealing with JavaScript Object Notation data and so on.

7. The UI category is actually an official plug-in from jQuery that provides some common interface widgets, like slider controls, accordion, progress bars, dialogues… etc.

8. Extensibility category is what enables the construction of jQuery plugins that add on to the functionality of the base library.

# 2. Retrieving Page Content

## 2.1 Overview of Selectors and Filters

Retrieving content from the page is one of jQuery's best features. This is something that you would usually use the DOM for in the past and it's one of the things that jQuery makes much easier.

- The selectors and filters feature of jQuery select content from the document, so that it can be manipulated using other functions, either jQuery functions or native built-in JavaScript functions. And you can think of jQuery selectors and filters as the query part of the jQuery. The way these work:

1. The selectors will return an array of objects that match the selection criteria that you give them.

2. The filters will operate on a selector to further refine the results array that the selector returns.

- The array that comes back is not a set of DOM elements.

- It's a collection of jQuery objects that are wrapped around the DOM elements and these jQuery objects provide a large number of predefined functions and properties for further operating on the objects. You can get access to the underlying DOM element for each one of these objects, if you want to, but the whole purpose of doing this is that you are wrapping them in jQuery objects, so that you have a whole bunch of convenience functions and high-level ways of operating on them without having to resort to the DOM in order to do that.

## 2.2 Using Basic jQuery Selectors

- Basic jQuery selectors are based on the familiar CSS syntax that you are probably already familiar with, and they work pretty much the same way that CSS does.

- List of the basic CSS selectors that correspond directly to their CSS counterparts.

| | |
|---|---|
| tagname | Finds all elements that are named ***tagname***. |
| #identifier | Finds all elements that has that ***id*** of ***identifier***. |
| .className | Finds all the elements that have a class attribute with the value of ***className***. |
| tag.className | Gets elements of type ***tag*** that have a class attribute with the value of ***className***. |
| tag#id.className | Retrieves the ***tag*** element that has an ***ID*** of ***id*** and a class attribute with the value of ***className*** . |
| * | The asterisk, finds all of the elements on the page. |

- How you would use jQuery versus using the plain browser DOM to get at information in a document.

Get all the paragraph <***p***> ***tags*** in the document, using the DOM vs jQuery

```
document.getElementsByTagName("p");
```

```
$("p");
```

Get the tag with the ***id="list1"*** , using the DOM vs jQuery

```
getElementById("list1");
```

```
$("#list1");
```

Get all the ***list item tags*** that have class ***a*** on them.

Using the DOM, it's a little bit more complicated. You would have to first use the ***getElementsByTagName*** function to get all the ***li*** tags, then you

would have to write ***for*** loop that would loop through all the results in that array and get the class attribute and compare it to this name ***a***, to see if it should be included in the result set.

Using jQuery:

```
$("li.a");
```

Get all the tags regardless of what kind of tag they were, that have a class ***b*** on them but only if they are inside a ***ul*** tag.

Using DOM this would be pretty complicated but jQuery makes it really easy.

```
$("ul .b");
```

- In addition to the basic selectors, there are the hierarchy and combination selectors. So, these allow you to get a little bit more advanced in selecting page content.

- It allows you to select elements based on hierarchical relationships or a combination of criteria.

| | |
|---|---|
| selector, selector, … | Finds all of the specified selectors. |
| .class1.class2 | Finds all elements with both ***.class1*** and ***.class2*** applied. |
| parent>child | Finds all ***child*** elements that are direct children of elements of type ***parent***. |
| ancestor descendent | Finds all ***descendent*** elements that are contained within elements of type ***ancestor***. |
| prev+next | Finds *every* ***next*** elements that are immediately next to a ***prev*** element. |
| prev~siblings | Finds all ***sibling*** elements that come after ***prev*** and match the siblings selector. |

# 2.3 Using Basic jQuery Filters

Filters are the feature of jQuery that make selectors even more powerful while retaining the CSS style simplicity.

- Filters work in conjunction with selectors and they provide even more fine-grained control over how elements are selected in the document.

- jQuery filters fall into six different categories:

1. The Basic category provide basic filtering like getting the first or the last or the even or odd numbered items in the returned set from a selector.

2. The Content filters take a set of elements and they filter out elements based on their content. Like for example, whether an element contains a particular string.

3. Visibility filters act on elements using their visibility setting. And they will filter out elements that are either hidden or visible.

4. The Attribute filters will examine a given attribute on an element and they will use that attributes' value to determine if it should be filtered out or included in the selector's result set.

5. The Child filters select elements based upon their relationship with their parent element.

6. The special set of filters that work on form elements and you can use these as really convenient powerful ways of processing elements in forms based upon what kind of form fields they are, whether they are enabled or not, whether they are checked or not, etc.

- Basic filters allow you to refine the results of a jQuery selector by only including elements that match certain conditions.

```
$(“p:not(p:eq(2))”)
```

Selects all the paragraphs except the one with the index no. 2 (third paragraph).

| | |
|---|---|
| :first | Selects only the first instance of the selector's returned set. |
| :last | Selects only the last instance of the selector's returned set. |
| :even | Selects only even-numbered elements in the selector's returned set. |
| :odd | Selects only odd-numbered elements in the selector's returned set. |
| :eq(n) | Filters out elements that are not positioned at the given index. |
| :gt(n) | Includes elements that are after the given index (greater than). |
| :lt(n) | Includes elements that are before the given index (less than). |
| :header | Selects all header elements (H1, H2, H3, etc). |
| :animated | Selects all elements that are currently being animated in some way. |
| :not(selector) | Includes elements that do not match the given selector. |

## 2.4 Using jQuery Attribute Filters

Attribute filters provide the ability to further filter out the results of a selector statement based upon attributes that are on the elements being selected.

**Examples:**

```
$("p[class]").css("border", "3px solid red");

$("p[id=para1]").css("border", "3px solid red");

$("p[id^=para]").css("border", "3px solid red");

$("p[id^=para][lang*=en-]").css("border", "3px solid red");
```

Table of the available filter attributes:

| | |
|---|---|
| [attribute] | Includes elements in the result set if they have the specified ***attribute***. |
| [attribute=value] | Includes elements in the result set if they have the specified ***attribute*** and it has the given ***value***. |
| [attribute!=value] | Includes elements in the result set only if they have the specified ***attribute*** and it doesn't have the given ***value***. |
| [attribute^=value] | Includes elements that have the specified ***attribute*** and it starts with the specified ***value***. |
| [attribute$=value] | Includes elements that have the specified ***attribute*** and it ends with the specified ***value***. |
| [attribute*=value] | Includes elements that have the specified ***attribute*** and it contains the specified ***value***. |
| [attrFilter1][attrFilterN] | Includes elements that match all of the specified attribute filters. |

## 2.5 Child, Visibility, and Content Filters

Content and Visibility filters examine the content of the elements that are returned by the selector expression or their visibility property to determine if they should be included or excluded from the result set.

```
$("p:contains(3)").css("border", "3px solid red");

$("p:parent").css("border", "3px solid red");

$("ul:has(li[class=a])").css("border", "3px solid red");

$("ul li:nth-child(2n)").css("border", "3px solid red");
```

The Content filter:

| | |
|---|---|
| :contains(text) | Filters the selection to only include elements that contain the ***text*** string anywhere within their content, in any of their child elements. |
| :empty | Filters the selection to only include empty elements (have no child elements inside them). |
| :has(selector) | Matches elements that contain at least one element that has the specified selector. |
| :parent | Matches all elements that are parents (i.e. they contain at least one other element, including ***text***). |

The Visibility filter:

| | |
|---|---|
| :invisible | Filters the selection to only include visible elements. |
| :hidden | Filters the selection to only include hidden elements. |

The Child filters allow you to refine the selection by looking at the relationship that the element has with its parent elements.

| | | |
|---|---|---|
| :nth-child(equation) | :nth-child(index) | :nth-child(even) |
| :nth-child(odd) | Matches elements at index, or even or odd increments, who match an equation of the form Xn+M (for example, 2n or 3n+1). | |
| :first-child | Matches elements who are the first child of their parent. | |
| :last-child | Matches elements who are the last child of their parent. | |
| :only-child | Matches elements who are the only child of their parent. | |

***Note:*** When you are using the ***nth-child*** operator and you are using the equation, the n variable starts off counting at 1. It doesn't start off counting at 0. Although jQuery operates on elements that are 0 index based, this is an exception here.

## 2.6 Form Selectors and Filters

jQuery have a set of selectors and filters that are just used for dealing with form elements. They work pretty much like any other selector, but they happen to start with a colon character, like filters do. These kinds of selectors and filters make it easy to build form processing logic using jQuery.

The available selectors for form elements:

| | |
|---|---|
| :input | Finds all input, select, textarea, and button elements. |
| :text | Finds all text elements. |
| :password | Finds all password elements. |
| :radio | Finds all radio button elements. |
| :checkbox | Finds all checkbox elements. |
| :submit | Finds all submit elements. |
| :reset | Finds all reset elements. |
| :image | Finds all image elements. |
| :button | Finds all button elements. |
| :file | Finds all file upload elements. |

The form filters are essentially convenience to help you select form elements that are in certain state. For example, we have a filter for enabled and disabled, and we have got a filter for check and selected.

| | |
|---|---|
| :enabled | Matches all form elements that are enabled. |
| :disabled | Matches all form elements that are disabled. |
| :selected | Finds all form elements that are selected. |
| :checked | Matches all form elements that are checked (radiobuttons and checkboxes). |

**Examples:**

```
$("form :input").css("border", "3px solid red");

$("form :text").css("border", "3px solid red");

$("form :text:enabled").css("border", "3px solid red");

$("form :checked").css("border", "3px solid red");

$("form :checkbox:checked").css("border", "3px solid red");
```

# 2.6 Traversing Documents

Traversing across the information returned from a document is something that's a very common operation in web development and jQuery provides a lot of functions that make this process easier.

| | |
|---|---|
| length | The number of elements in the jQuery result set. |

```
// Inspect the length and of a result set
alert("There are " + $("p").length + " <p> elements");
//Returns the number of paragraph elements.
```

| | |
|---|---|
| get() | Returns an array of all matched DOM elements. Useful if you need to operate on the DOM elements themselves instead of using built-in jQuery functions. |
| get(index) | Access a single matched DOM element at a specified index in the matched set. |

**Note:** In jQuery, the result set that come back from a jQuery selection is not an array of DOM elements. Those DOM elements have been wrapped inside jQuery objects, which have a lot of functionality built on top of them. Therefore, these functions are useful if you need to operate on the DOM elements themselves.

```
// use the get() and get(index) to retrieve DOM elements
var elems = $("li").get();
//Returns an array of all the list items as DOM elements.
alert("There are " + elems.length + " <li> tags");
//This is the built-in JavaScript array length operator (not the jQuery operator).
```

```
alert($("li").get(0));
//Returns [object HTMLLIElement] showing the type of object is at index 0 to work on that li tag using the DOM if we want.
```

| | |
|---|---|
| find(expression) | Searches for descendent elements that match the specified expression. Searching inside a ***div*** for images, for example. |

```
// use the find function
$("ul").find("li.b").css("border", "3px solid red");
```
//Find list items that have a class ***b*** inside of the ***ul*** tag.

**Note:** You can accomplish this using selector expressions, but this is just for the sake of illustration. There are much more powerful ways to use the ***find***.

| | |
|---|---|
| each(fn) | Loop over the contents of every element in the result set of a jQuery selection and execute a function against that element. |

```
// Use the each function
var leftmargin = 0;
var border = 3;
$("p").each(function() {
   $(this).css("border", border+"px solid red");
   $(this).css("margin-left", leftmargin);
   border += 2;
   leftmargin += 10;
});
```
//Apply progressively increasing values to the style applied to the paragraph elements.

**Note:** The ***this*** keyword is going to refer to each one of the paragraphs as it comes through.

## 2.7 Understanding jQuery Statement Chaining

One of jQuery's most powerful features is its ability to chain multiple functions together (statements chaining) and this basically allows you to perform several operations in just one line of code. It allows you to perform multiple operations on one result set without having to get that result set repeatedly, as if you might have to do in other scripting conventions.

**Example:** `$(selector).fn1().fn2().fn3();`

A series of functions will get executed in order, starting on the left and going across to the right. The selector will be executed. It will come back with a result set of elements in the web page and then we can do multiple operations on them.

## 2.8 Practical Example 1: Annotating Page Links

There are different kinds of links in the page. Some goes to PDF, a couple of more HTML files, another goes to a mailto link. If you look at the design, you can't really tell that visually. You really can't tell that Link #3 is a PDF file versus an HTML file. So, we are going to write a one line script in jQuery that automatically decorates all the links in a page with little icons that indicate that they go to PDF files rather than HTML pages.

That query should get all of the links that have attributes named ***href*** and if the ***href*** attribute ends with a ***.pdf*** string, then it will include a little icon.

```
<script type="text/javascript">
   $("document").ready(function() {
      $("li a[href$='.pdf']").after("<img src='images/small_pdf_icon.gif'
align='absbottom' />");
   });
</script>
```

# 3. Manipulating Page Content

## 3.1 Creating, Getting, and Setting Content

- Once you've used the selectors and filters to get the content out of the webpage, typically you're going to want to do something with it.

- Sometimes you want to create new content to add it into the page dynamically.

- jQuery's content manipulation functions are functions for creating, copying, deleting and moving content around as well as wrapping page content inside other content.

- jQuery provides some really good cross-browser support for working with CSS information including positioning, and sizing information.

- To create new HTML content, you basically just pass a ***string*** containing the new ***HTML*** to the jQuery function.

```
var newHeader = $("<h1>My New Header</h1>");
```

Or

```
var myStr = "<h1>My New Header</h1>";

var newHeader = $(myStr);
```

The result of this statement will be a new jQuery object, a new wrapped element, a result set if you will, that contains the HTML that you passed in as the string there. And since it's just a string, it also works really well if you want to pass in a variable as well.

- You can use jQuery's ***html()*** and ***text()*** methods to get and set content on elements.

| | |
|---|---|
| html() | Returns the HTML content of the first matched element that comes back from the selector's result set that you are calling this function on. |
| html(newcontent) | Sets the HTML content of every matched element. |
| text() | Returns the text content of the first matched element. |
| text(newtext) | Sets the text content for all matched elements. |

***Note:*** The main difference between text and HTML is that if you pass HTML code to the ***html()*** function, it will actually create real HTML for you. If you try to pass a string into the ***text()*** function that contains HTML, the ***text()*** function will automatically escape it for you. It will change angle brackets into text representations. It's not going to actually create HTML code.

```
alert($("#list1").html());
```

Returns the HTML content that is inside of the ***UL*** (all the ***li*** tags).

Returns only the ***text*** if it's a ***paragraph*** element.

```
//change the html of the UL
$("#list1").html("<li>This is a new item</li>");
```

Set the HTML to the new list item. The new one replaced all the old list items.

```
// create a new p set the content of para2 to the new p
var newItem = $("<p>This is a new paragraph</p>");
$("#para2").html(newItem.html());

// set the text content of the last paragraph
$("p:last").text("this is the last paragraph");
```

# 3.2 Manipulating Attributes

To inspect or change the value of attribute on elements, use ***attr*** functions.

### attr(name)

Accesses property on the first matched element. This method makes it easy to retrieve a property value from the first matched element. If the element does not have an attribute with such a name, undefined is returned.

**Ex:** .attr(src) get the value of the source attribute on an image.

### attr(properties)

Sets a series of attributes on all matched elements using an object notation syntax. This is the best used for setting large numbers of properties at once.

**Example:**

```
$(“img”).attr({ src: “/images/hat.gif”, title: “jQuery”, alt: “jQuery Logo” });
```

Setting attributes as a properties object.

### attr(key, value)

Sets a single property to a value on all matched elements.

**Example:**

```
$("a").attr("target", "_blank");
```

It will look through all the link tags on a page and set the target attribute to be blank, so that all links will open in new pages.

### attr(key, fn)

Sets a single property to a computed value, on all matched elements. Instead of supplying a string value, a function is provided that computes the value of the attribute.

**Ex:** when you're in a loop and you're using index values to calculate values of an attribute. For example, suppose you're looping through a series of images and you're numbering things in ascending order, then this function would be used as a callback. So as each attribute is set, you can use a function to calculate what its value should be.

removeAttr(name)

Removes the named attribute from all matched elements.

**Example:**

```
$("a").removeAttr("href");
```

Removed the ***href*** from the link, which means that as far as the browser is concerned, this is not a valid link anymore.

## 3.3 Inserting Content

jQuery provides several functions for inserting content into the document and for modifying the document object model's DOM tree. And you can insert content into the document before and after existing page elements. You can also insert content inside of elements.

append(content)

Appends content to the inside of every matched element.

prepend(content)

Prepends content to the inside of every matched element.

**Example:**

```
$("p").append(" with some content appended");

$("p").prepend("Hello! ");
```

The ***append*** function insert the content to the end of every one of matched elements that this function is being called on. Conversely, the ***prepend*** function will take content and put it at the beginning of the inside of every matched element.

| | |
|---|---|
| appendTo(selector) | Appends all of the matched elements to another, specified, set of elements. |
| prependTo(selector) | Prepends all the matched elements to another, specified, set of elements. |

**Example:**

| | |
|---|---|
| `$("li:first").appendTo("ul");` | //Append the first ***li*** element to the end of the ***ul*** tag. |
| `$("li:last").prependTo("ul");` | //Append the last ***li*** element to the beginning of the ***ul*** tag. |

| | |
|---|---|
| after(content) | Inserts ***content*** after each of the matched elements. |
| before(content) | Inserts ***content*** before each of the matched elements. |

| | |
|---|---|
| insertAfter(selector) | Inserts all of the matched elements after another, specified, set of elements. |
| insertBefore(selector) | Inserts all of the matched elements before another, specified, set of elements. |

# 3.4 Wrapping, Replacing, and Removing Content

jQuery has a bunch of functions for doing operations like wrapping content in the page, replacing it, copying it and removing it.

| | |
|---|---|
| .wraps(html) | Wraps each matched element with the specified HTML content. |
| .wraps(element) | Wraps each matched element with the specified element. |

The ***Wrap()*** function will take each one of the elements on the matched set that you are calling the ***Wrap()*** function on, and place them inside the HTML that you specify in the function argument.

So, for example if I was calling the ***Wrap()*** function on a whole bunch of images, and I passed a string, like ***DIV*** inside the parentheses, then each one of the images will be wrapped inside it's own separate ***DIV***.

Similarly, the ***Wrap Element*** works pretty much the same way. Each one of the elements inside the matched set that this is being called on, will be wrapped inside the element that I specify in the function argument.

**Example:** `$("p").wrap("<div style='color:red'/>");`

Wrap all the ***paragraphs*** with ***div*** tags.

| | |
|---|---|
| .wrapAll(html) | Wraps all the elements in the matched set with the specified HTML content. |
| .wrapAll(element) | Wraps all the elements in the matched set into a single wrapper element. |

The ***wrapAll()*** works pretty similarly although what it does is rather than wrapping each one of the individually matched elements into it's own wrapper. It will take all of the elements inside the matched set and place them in either the HTML that I am specifying, or the element.

**Example:** `$("p").wrapAll("<div style='border:3px solid red'/>");`

Wrap all the ***paragraphs*** with a ***div*** tag.

| | |
|---|---|
| .wrapInner(html) | Wraps the inner child contents of each matched element (including text nodes) with an HTML structure. |
| .wrapInner(element) | Wraps the inner child contents of each matched element (including text nodes) with a DOM structure. |

The ***wrapInner*** functions will wrap the inside trial contents of each matched element, including text within either HTML or the DOM element that specify in the function argument, assuming that the DOM element that I am passing is allowed to have children inside of it. So, for example Images can't have child elements inside them. So that wouldn't work. If I call ***wrapInner*** on an ***ul*** tag and I had a whole bunch of ***li***s, then all those ***li***s will be wrapped inside whatever I passed in as this HTML or this element.

| | |
|---|---|
| .replaceWith(content) | Replaces all matched elements with the specified HTML or DOM elements. |
| .replaceAll(selector) | Replaces the elements matched by the specified selector with the matched element. |

The ***replaceWith*** function, you give it some content, and it will replace all of the matched elements in the result set with whatever content you specify. And content either be HTML or a series of DOM elements. And the ***replaceAll*** basically replaces the elements that are matched by the selector with the matched elements. So, the selector is going to be used to find which should be replaced by the all the elements that are being called on.

| | |
|---|---|
| .empty() | Removes all child nodes from the set of matched elements. |

**Example:** `$("ul").empty();`

Get rid of all the ***li*** items inside the ***ul*** tag.

| | |
|---|---|
| .remove() | Removes all matched elements from the DOM. |

| | |
|---|---|
| .clone() | Clone matched DOM elements and then selects all the clones. |
| .clone(bool) | Clone matched DOM elements, and all their event handlers, and select the clones (if the bool is set to ***true***). |

## 3.5 Working with CSS

- jQuery's CSS functions provide easy cross-browser access for setting properties as well as working with positioning and sizing information for elements that are in your webpage.

- The ***css()*** function allows you to retrieve and set styles for a set of matched elements.

.css(name)

Returns the value for the named CSS property for the first matched element.

.css(properties)

Sets the CSS properties of every matched element in the result set using an object-notation syntax:

```
var cssObj = {
‘background-color’ : ‘#dda’,
‘font-weight’ : ‘ ’,
‘color’ : ‘rgb(0, 40, 244)’ }
$(this).css(cssObj);
```

.css(property, value)

Sets a single style property to a value on all matched elements. If a number is provided, it is automatically converted into a pixel value, with the following exceptions: z-index, font-weight, opacity, zoom, and line-height. Because they don't take pixel values.

- jQuery also provides a set of functions for working with CSS classes.

- jQuery provides functions for adding and removing, toggling, and detecting these classes.

| | |
|---|---|
| .addClass(class) | Adds the specified class(es) to each of the set of matched elements. |
| .hasClass(class) | Returns true if the specified class is present on at least one of the set of matched elements. |
| .removeClass(class) | Removes all the specified class(es) from the set of matched elements. |
| .toggleClass(class) | Adds the specified class if it is not present, removes the specified class if it is present. |
| .toggleClass(class, switch) | Adds the specified class if the switch is true, removes the specified class if the switch is false. |

- The CSS positioning functions provide a cross-browser way of figuring out the positions of elements.

| | |
|---|---|
| .offset() | Gets the current offset of the first matched element, in pixels, relative to the document. It comes back within object that has a top and a left. |
| .offsetParent() | Returns a jQuery collection with the positioned parent of the first matched element. |
| .position() | Gets the top and left position of an element relative to its offset parent. |
| .scrollTop() | Gets the scroll top offset of the first matched element. |
| .scrollTop(val) | Sets the scroll top offset to the given value on all matched elements. |
| .scrollLeft() | Gets the scroll left offset of the first matched element. |
| .scrollLeft(val) | Sets the scroll left offset to the given value on all matched elements. |

- jQuery gives you cross-browser sizing information for elements using size-related CSS functions.

| | |
|---|---|
| .height() | Gets the current computed, pixel, height of the first matched element. |
| .height(val) | Sets the CSS height of every matched element. |
| .width() | Gets the current computed, pixel, width of the first matched element. |
| .width(val) | Sets the CSS width of every matched element. |
| .innerHeight() | Gets the inner height (excluding the border and including the padding) for the first matched element. |
| .innerWidth() | Gets the inner width (excluding the border and including the padding) for the first matched element. |

### .outerHeight(margin)

Gets the outer height (includes the border and padding by default) for the first matched element. If the margin argument is true, then the margin values are also included.

### .outerWidth(margin)

Gets the outer width (includes the border and padding by default) for the first matched element. If the margin argument is true, then the margin values are also included.

## 3.6 Associating Data with Page Elements

jQuery provides a very useful methods for associating arbitrary data with elements on the page. There are several scenarios where this is pretty useful. For example, you might have an image on a page, and you might want to associate some data with that image such as who the photographer is, and when and where the photo was taken. Or maybe you have a series of ***div*** elements on a page, and each one of those ***div***s contain some information about, say, an employee record, and you want to associate some data at runtime such as the employee name and their position and so on.

jQuery makes that really easy, by providing a couple of methods that are used for working with data.

The ***.data()*** method is used to store and retrieve data associated with elements on the page, and there's two ways of calling it.

The way data is stored is by keys and values.

.data( key, value )

.data( obj )

In the first way of calling it, you give it a ***string***, which is the ***key*** that you want to associate the data with, and then the ***value*** is any JavaScript ***object***. It can be a number, it can be a string, it can be an array, it can be a complex JavaScript object. Or you can just give it an Object which contains a whole bunch of key-value pairs to put the data into the page element.

.data( key )

.data()

Return the value at the named data store for the first element in the jQuery collection, as set by ***data(name, value)*** or by an HTML5 ***data-**** attribute.

.removeData()

.removeData( [name ] )

Remove data from an element or set of elements on the page. You can give it the ***element*** to remove the data from, and if you want, you can give it an optional ***string***, which is the individual ***key*** for the data you want removed. Or if you just don't provide any key name at all, then all of the data will be removed.

```
<script>
  $("document").ready(function() {
    $("#store").click(function () {
        //When the user clicks on the store button it will store some arbitrary
        data on the DIV object.
      $("#div1").data("key1", 1234);
      $("#div1").data("key2", "Joe Marini");
    });

    $("#remove").click(function () {
      //Clear the data from the DIV, when the user clicks the remove button.
      $("#div1").removeData();
    });

    $("#show").click(function () {
       //If there is any data, display it (All the 3 keys).
      //alert($("#div1").data("key1"));
      //alert($("#div1").data("key2"));
      console.log($("#div1").data());
      //alert(JSON.stringify($("#div1").data(), null, "  "));
    });
  });
</script>
```

```
<div id="div1" data-key3="3rd  data attribute">Sample DIV</div>
```

**Returns:**

Object {key3: "data attribute", key1: 1234, key2: "Joe Marini"}

**Note:** The data- part of the key name is stripped off. That's not part of the actual attribute name; that's just what the browser uses to indicate that something is a data attribute. So everything following the ***data-*** is what the actual attribute of the data is when it's put into the data handling method of jQuery.

# 3.7 Practical ex. 2: Automatic TOC Generator

The document got a bunch of Lorem ipsum text in it and a bunch of headings. The Table of Contents, contains links that jump to the various places in the document. So for example, if I click on a link, it will jump down to that heading. The Table of Contents is being built up automatically by jQuery code.

So we are going to see how to build that automatic TOC Generator, which will look through the document for various kinds of headers and then use those to build named anchors inside the document where we want to jump to.

```
buildBookmarks('h3', 'header');
```

When the ***document.ready*** event fires, the function will be called with two arguments. The first argument is the type of header tag that we want to serve as the TOC locations and the second argument is the ***id*** of the ***div*** that we want to append the resulting Table of Contents HTML into.

```
<div id="header">
```

The other thing you should realize also is that each one of these sections in the document is enclosed in a ***div*** and has an ***id*** and there is ***h3*** tags that are at the top of each one of the sections. These are the headers for each section in the document.

```
var oList = $("<ul id='bookmarksList'>");
```

Make an unordered list whose ***id*** is equal to ***bookmarksList*** which will hold the bookmark links.

So for each one of the header tags, there is two things we need to do. First, we need to create a named anchor and insert it into the header tag because that's going to serve as the jump location for the links in the TOC.

Then we need to add that link into the unordered list that will point to the named anchor.

```
$("div:not([id=header]) " + strWhichTag).each(function() {
```

Get all the ***div***s, because the sections are all divided into ***divs***. But not the ***div*** where id=”***header”***, because this is where the TOC is going to be. Use ***strWhichTag*** to look inside the ***divs***, because we need the ***h3*** tags.

Use the ***each*** function to iterate over each one of the contents that we find, because there is two things that we have to do.

```
$(this).html("<a name='bookmark" + cAnchorCount + "'></a>" +
$(this).html());
```

The first thing is to create the named anchor and since we need named anchors to be unique in the document, we have to use a variable ***cAnchorCount*** .

Inside the ***each*** function, set the html for the ***h3*** tag to be a named anchor using the this ***this*** keyword because it refer to the ***h3*** tag. Then re-add the existing HTML for the ***h3*** because we don't want the ***h3*** to be blown away.

**Note:** (my experiment) both of them return the content (text) of the ***h3*** tag.

```
console.dir($(this).html());
console.dir($(this).text());
```

```
oList.append($("<li><a href='#bookmark" + cAnchorCount++ + "'> " +
$(this).text() + "</a></li>"));
```

The second thing is we need to add our link to the list that we are creating.

That's going to be a list item, and inside the list item, there is going to be an anchor or a link and the ***href*** is going to go to the named anchor for the bookmark plus the text that's inside the ***h3***.

```
$("#" + sBookMarkNode).append(oList);
```

After we have finished creating the content of the list and the named anchor. When the loop terminates, put the list of links we created inside the ***div*** of the ***id=”header”***.

**Summary:**

We have a variable that keeps track of a counter, so that all of our named anchors are unique. We create an unordered list. That's going to be the list of TOC. For every ***div*** that's not the header, plus the ***h3s***, we are going to loop over each one of those guys and we are going to create a named anchor that goes in front of the ***h3*** tag. That's the link destination. Then we create a list item that has the link into it that jumps to that named anchor and so we do that for each one of the ***h3s*** and when we have done that, we put the entire list into the ***div***.

```
<script type="text/javascript">
   $("document").ready(function() {
      buildBookmarks('h3', 'header');
   });

   function buildBookmarks(strWhichTag, sBookMarkNode) {
      var cAnchorCount = 0;

      // create the list that will hold the bookmark links.
      var oList = $("<ul id='bookmarksList'>");

         //For each one of the header tags,
         //Create a new named anchor and insert it into the header tag.
         // Then add a new link to the list that points to the named anchor.
      $("div:not([id=header]) " + strWhichTag).each(function() {
         $(this).html("<a name='bookmark" + cAnchorCount + "'></a>" +
$(this).html());
         oList.append($("<li><a href='#bookmark" + cAnchorCount++ + "'> " +
$(this).text() + "</a></li>"));
      });

      // now find the ID of the bookmark container and append it.
      $("#" + sBookMarkNode).append(oList);
   }
</script>
```

# 4. Working with Events

## 4.1 Understanding the jQuery Event Handling Features

- jQuery provides a mechanism for working with events that is simpler than relying on the Document Object Model.

- Abstracts away the differences between browser implementations.

- jQuery works with sets of elements by default, so it's very easy to write code that assigns event handlers to groups of objects, just by using the results of the selectors and filters.

- jQuery events basically break down into the following categories:

1. The Binding and Unbinding category allow you to wire events up, and take them off of elements in a simple cross-browser way.

2. The Unified Event Object provides an event object that exposes the properties and methods that you use most commonly in a cross-browser way.

3. The convenience features provides functions that encapsulate a lot of common event features and helper routines that work cross-browsers.

## 4.2 Binding and Unbinding Events

***.on()*** function is used to attach an event handler for one or more events to the selected elements in the page, and then the ***.off()*** function removes an event handler.

You might see older jQuery code using functions like ***.live()*** and ***.die()*** which are already been deprecated, or function like ***.bind()*** and ***.unbind()***, which still work, but at some point, they'll be deprecated too.

```
.on( events [, selector ] [, data ], handler )
```

events argument is a space-separated string for the events that you want to attach, for example click or mouse down or mouse up.

selector argument is optional, and it's used to filter out descendants of the selected elements that are going to trigger the event.

data argument can be any arbitrary data that you want passed to the event handler when the event is triggered. So, if there's some data that you want to use inside your event handler, you can do that.

handler argument essentially is the function that's going to be triggered when the event happens. That's where your event handling code will be.

**Example:**

We have 2 style classes, ***.normal*** and ***.highlighted*** . And a ***div*** tag that has an ***ID*** of ***#evtTarget***, and the current class is the ***.normal*** class.

And we're going to wire up some event handlers so that when the mouse is over the element, the highlighted CSS class gets added, and then when I mouse out, it gets removed.

```
$("#evtTarget").on("mouseover mouseleave", highlight);
```

Start off by getting a reference to the event target element ***#evtTarget*** . And inside ***on***, pass ***mouseover mouseleave*** which are the two events that I want to attach a handler for. ***highlight*** is a function that handles the event.

```
function highlight(evt) {
   $("#evtTarget").toggleClass("highlighted");
}
```

It takes an event argument ***evt*** because it's an event handler function. Use the CSS helper function, ***toggleClass()*** which will either remove or add the ***highlighted*** CSS style sheet.

```
$(function() {

   $("#evtTarget").on("mouseover mouseleave", highlight);

   $("#evtTarget").on("click", function(evt) {
         //A handler for the mouse-clicking event which will turn off
         the highlighting behavior of the hover and output some HTML.

      $("#evtTarget").off("mouseover mouseleave", highlight);
      $("#evtTarget").html("<p>You shut off the hover effect!</p>");
   });

});

function highlight(evt) {
   $("#evtTarget").toggleClass("highlighted");
}
```

# 4.3 Convenient Event Helper Methods

noVid

There are also a bunch of higher-level event handling functions like click, double-click, hover, mousedown, and mouseup that you can use to handle common situations with events.

**Example:**

This is pretty much the same example we used in the previous lesson. It has the ***div***. It's got the style sheets ***.normal*** and ***.highlighted***, only this time we're going to write the code using the helper functions.

In the previous example we used the ***on*** function to bind events for ***mouseover*** and ***mouseleave***. However, jQuery give us a function that already does that.

.hover(handlerIn, handlerOut) function takes two functions, and those two functions are what you want to happen when the mouse enters something and when the mouse leaves.

.click( handler ) Bind an event handler to the "click" JavaScript event, or trigger that event on an element.

.dblclick() Bind an event handler to the “dblclick” JavaScript event, or trigger that event on an element.

So, that's an example of using the higher-level functions to achieve features and effects that you would normally have to use by using the lower-level event binding functions, and there's a whole bunch of these, check them out: http://api.jquery.com/category/events/mouse-events/

```
$(function() {
   $("#evtTarget").hover(highlight, highlight);

   $("#evtTarget").click(fnClick1);
   $("#evtTarget").dblclick(fnClick2);
});

function highlight(evt) {
   $("#evtTarget").toggleClass("highlighted");
}
function fnClick1() {
   $("#evtTarget").html("Click!");
}
function fnClick2() {
   $("#evtTarget").html("Double Click!");
}
```

# 4.4 Using the jQuery Event Object

- Writing event-handling code is frustrating when it differs across browsers.
- The jQuery event object smoothes these differences and provides a single object with the most important properties that you will use commonly.

The list of the most common properties and methods:

| | |
|---|---|
| type | Type of the event that the event object is representing in a string ("click", e.g.). |
| target | Element that issued the event. |
| data | Data passed to bind function. |
| result | Value returned by the last event handler function. |
| timestamp | Time when event occurred in milliseconds. |
| pageX, pageY | Coordinates of mouse when event happened, relative to document. |

| | |
|---|---|
| preventDefault() | Prevents the browser from executing the default action of the event like following a link after click. |
| isDefaultPreventd() | Returns whether ***preventDefault()*** was ever called on this event object. |
| stopPropagation() | Stops the bubbling of an event to parent elements. |
| isPropagationStopped() | Returns whether ***stopPropagation()*** was ever called on this event object. |

```
$(function() {
   $("div").click(function(evt) {
      $(this).html("pageX: " + evt.pageX + ", pageY: " + evt.pageY + ", type: "
+ evt.type + ", target: " + evt.target);
   });
});
```

**Returns:**

pageX: 19, pageY: 98, type: click, target: [object HTMLDivElement]

# 4.5 Using Miscellaneous Event Features

- For a couple of specialized tasks, jQuery provides some miscellaneous functions.

.one(type, data, handler)
Works the same as ***on()***, but the event handler is only ever executed one time for each one of the matched elements that the event is bound to. So, the event will be bound, and then once the event gets triggered, it will not be responded to again.

```
$(function() {
   $("div").one("click", function() {
      $(this).css({ background: "red", cursor: "auto"
      });
   });
});
```

The ***this*** parameter will refer to the ***div*** that got clicked on. So, we're going to set the CSS for the ***div*** that got clicked on.
So I click on it once, it changes to red and the pointer went away, and if I click on it again, nothing happens anymore.

The other two functions, ***trigger()*** and ***triggerHandler()***, are ways of essentially triggering events from within code without having to wait for the user.

trigger(event, data) function will trigger an event on each one of the matched elements that come back from the selector. And it will also cause the browser to act as though the user caused the event to happen.
So, for example if you trigger the ***click*** event on a link, then not only will the event fire off from within your code, but the browser will act as though the item were clicked on.

triggerHandler(event, data) function is similar. It triggers the event handlers but it does it without having the browser go off, and act as though the event actually happened. This only works on the first matched element in the result set for a selector.

# 4.6 Practical ex. 3: Table Striping and highlighting

Use jQuery to automatically stripe and hover-highlight the contents of a simple HTML table.

```
$(function() {
  $("#theList tr:even").addClass("stripe1");
  $("#theList tr:odd").addClass("stripe2");

   //The hover event takes 2 arguments.
  $("#theList tr").hover(
     function() {
        $(this).toggleClass("highlight");
     },
     function() {
        $(this).toggleClass("highlight");
     }
  );

});
```

## Event Delegation

Suppose I want to have the user, whenever they click on one of the table rows, just show a little alert message that has the text contents of the table row.

There's a couple of ways I could do that. One way to do that would be to

```
 $("#theList tr").on("click", function showText (evt) {
    alert($(this).text());
 })
```

That will work; however, there's a problem. And the problem is that the selector, is going to go through and assign click handlers to every single one of the table ***rows***. Now, for a table that only has 10, 12 ***rows***, that's okay. But imagine I had a table with lots and lots of ***rows***, or imagine this wasn't a table; it was a whole bunch of ***divs***. That's going to create a whole bunch of event handlers on a whole bunch of objects that basically just do the same thing.

What I'd like to be able to do is take advantage of the event bubbling in the browser, and just tell jQuery that whenever an event comes from a certain kind of element but makes its way back up to the element's parent, have the parent handle the event, but only if it came from a certain kind of descendent.

```
$("#theList").on("click", "tr", function showText (evt) {
  alert($(this).text());
})
```

Get the table, and attach the click handler to the table itself. But use the delegate trick to say only handle events when they come from table rows. So, if a click event gets triggered on the table and it came from a table row, then execute this function.
So now I've only got one event handler, and it's sitting on the top-level parent table, and it's listening for click events that came from table rows.
So rather than having a table with a thousand rows in it and having a thousand separate click handlers, I've only got one click handler, and it's listening for clicks that come from within table rows. And browsers handle this automatically. When something like a click event happens on an element and the element doesn't handle that event, such as a click, that click will then bubble up the DOM until it gets to a parent element which does have an event handler. And so by attaching the event handler for click directly to the table itself, I only have one place where I need to listen for it. So, only one event handler, and it's listing for events from that particular table row, and that is what event delegation is all about.

# 5. jQuery Animations and Effects

## 5.1 Hiding and Showing Elements

- jQuery library supplies some basic animation and effects functions to create some common visual effects such as showing and hiding elements, fading elements in and out, moving them around on the screen.
- You can use the basic animation function to build your own animation effects.
- Elements can be shown or hidden immediately or over a specified duration of time.

| | |
|---|---|
| show() | Displays each of the set of matched elements, if they are hidden. |
| show(speed, callback) | Displays all matched elements using graceful animation. Fires an optional callback after completion. |
| hide() | Hides each of the set of matched elements if they are shown |
| hide(speed, callback) | Hides all matched elements using graceful animation. Fires an optional callback after completion. |
| toggle() | Toggles displaying each of the set of matched elements. |
| toggle(switch) | Toggles displaying each of the set of matched elements based upon the switch (true shows all elements, false hides all elements). |
| toggle(switch, callback) | Toggles displaying each of the set of matched elements using a graceful animation and firing an optional callback after completion. |

speed you can either pass a string like slow, normal, or fast or you can pass a number, which is a millisecond duration. So 2000 milliseconds would be 2 seconds.

callback that you can have jQuery call when the animation has become complete. So for example, when an element is finished showing, if you want to do some of the UI changes, you can supply a callback function that will be notified when the effect is done.

```
$(function() {
   $("#show").click(function() {
      $("#theDiv").show(2000, function(){
         alert("Done Showing");
      });
   });
   $("#hide").click(function() {
      $("#theDiv").hide("normal");
   });
   $("#toggle").click(function() {
      $("#theDiv").toggle("slow");
   });
});
```

## 5.2 Fading Elements In And Out

- Elements can be faded in or out either completely or to a predetermined level of opacity so that they are transparent.
- The speed of the fade can be specified as either a string (“slow”, “normal”, or “fast”) or a millisecond duration.

The three routines that perform fading:

fadeIn(speed, callback)
Fades in all matched elements by adjusting their opacity and firing an optional callback after completion.

fadeOut(speed, callback)
Fades out all matched elements by adjusting their opacity to 0 and then setting ***display*** to “***none***” and firing an optional callback after completion.

fadeTo(speed, opacity, callback)
Fades the opacity of all matched elements to a specified opacity and fires an optional callback after completion.
The opacity level that you want the object to ***fadeTo*** is between 0 and 1.0 (the percentage of transparency).

```
$(function() {
   $("#fadein").click(function() {
      $("#theDiv").fadeIn("normal");
   });
   $("#fadeout").click(function() {
      $("#theDiv").fadeOut(4000, function(){
         alert("Done Fading Out");
      });
   });
   $("#fadeto3").click(function() {
      $("#theDiv").fadeTo("slow", 0.3);
   });
   $("#fadeup").click(function() {
      $("#theDiv").fadeTo("slow", 1.0);
   });
});
```

## 5.3 Sliding Elements

- The sliding effect is another way that jQuery gives you to reveal or hide page elements. These functions are not for sliding things around on the screen, just doing some generic movement. This is for showing and hiding elements but using a sliding effect.
- jQuery provides functions for sliding elements up or down, as well as toggling the slide animation based upon whether things are hidden or showing.

slideDown(speed, callback)
Reveals all matched elements by adjusting their height and firing an optional callback after completion.

slideUp(speed, callback)
Hides all matched elements by adjusting their height and firing an optional callback after completion.

slideToggle(speed, callback)
Toggles the visibility of all matched elements by adjusting their height and firing an optional callback after completion.

```
$(function() {
   $("#slideup").click(function() {
      $("#theDiv").slideUp(3000, function(){
         alert("Sliding Done!");
      });
   });
   $("#slidedown").click(function() {
      $("#theDiv").slideDown("normal");
   });
   $("#toggle").click(function() {
      $("#theDiv").slideToggle("slow");
   });
});
```

## 5.4 Creating Custom Animations

- In addition to the pre-built animation functions that jQuery gives you, there is a basic ***animate()*** function that you can use to create custom animation for a wide variety of properties on page elements.
- To stop animations in progress, call the ***stop()*** function.

animate(params, duration, easing, callback)
Creates a custom animation
params : The properties on the elements to animate.
duration : the number of milliseconds the animation should take.
easing : The type of easing function to use (linear or swing built-in). There are plug-ins that do more.
callback : The function to call when the animation is complete.

animate(params, options)
Creates a custom animation
params : The properties to animate.
options : Set of options for the animation to take.

stop()
Stops all the currently running animations on all the specified elements.

I call the ***animate()*** function with an object. Inside the object, I list the properties that I want to have animated.
Using the ***animate()*** function, you can create some pretty complex animations just by specifying the parameters that you want to animate from and animate to.
So the default, when I specify the parameters in the ***animate()*** function, like Width: 500, this is going to be the ending result. This is what the property will look like when the animation is complete. The starting value is whatever the Style Sheet sets it to. So you can see here that the Width starts out at 250, and it's going to end up at 500, and similarly, for the rest of the properties as well.

```
$(function() {
   //The side of the div grows out to the right.
   $("#right").click(function() {
      $("#theDiv").animate({ width: "500px" }, 1000);
   });
   //The text animates up in size.
   $("#text").click(function() {
      $("#theDiv").animate({ fontSize: "24pt" }, 1000);
   });
   //Animates over to the right.
   $("#toggle").click(function() {
      $("#theDiv").animate({ left: "500" }, 1000, "swing");
   });

   //Since we are specifying things as objects, I can have multiple of these
   animations appear all at once.
   $("#multiple").click(function() {
      $("#theDiv").animate({
         left: "500",
         fontSize: "24pt",
         width: "500px"
      }, 2000, "swing");
   });
});
```

# 5.5 Practical example 4: Image rotator

.next()
setInterval(function, duration)

```
$(function() {
   // create the image rotator
   setInterval("rotateImages()", 3000);
});

function rotateImages() {
   var oCurPhoto = $('#photoShow div.current');
   var oNxtPhoto = oCurPhoto.next();
   if (oNxtPhoto.length == 0)
      oNxtPhoto = $('#photoShow div:first');

   oCurPhoto.removeClass('current').addClass('previous');
   oNxtPhoto.css({ opacity: 0.0 }).addClass('current').animate({ opacity: 1.0
}, 1000,
      function() {
         oCurPhoto.removeClass('previous');
      });
}
```

# 6. The jQuery UI Plug-In