

jQuery Fundamentals Training

Effects

Lesson 1, Activity 2: jQuery makes it trivial to add simple effects to your page. Effects can use the built-in settings, or provide a customized duration. You can also create custom animations of arbitrary CSS properties.

For complete details on jQuery effects, visit <http://api.jquery.com/category/effects/>.

Built-in Effects

Frequently used effects are built into jQuery as methods:

- `$.fn.show` - Show the selected element(s). Without a duration parameter, the element is shown immediately. With a duration, the height and width of the element(s) are animated from zero to full-size over that duration.
- `$.fn.hide` - Hide the selected element(s). Behavior is similar to `show`, but shrinking the element instead of expanding.
- `$.fn.fadeIn` - Animate the opacity of the selected element(s) to 100%.
- `$.fn.fadeOut` - Animate the opacity of the selected element(s) to 0%.
- `$.fn.slideDown` - Display the selected element(s) with a vertical sliding motion.
- `$.fn.slideUp` - Hide the selected element(s) with a vertical sliding motion.
- `$.fn.slideToggle` - Show or hide the selected element(s) with a vertical sliding motion, depending on whether the elements are currently visible.
- `$.fn.delay` - simply holds the current state for the specified period

You can run a series of effects on an element. If you invoke several

effect methods on an element, the effects are queued up and run in succession, as opposed to in parallel (which is where `delay` comes in handy). You can also add other behaviors that aren't effects to the queue, like changing a class or CSS.

Basic Use of a Built-in Effects

```
$('h1').show();           // will appear immediately

$('h2').show(1000)        // will appear over 1 second,
.hide(1000);              // and then disappear over another second

$('h3').show(1000);       // the same thing done as
$('h3').hide(1000);       // two separate method calls

$('h4').show(1000)        // will appear over 1 second
.delay(1000)              // remain for 1 second
.hide(1000);             // and then disappear over another second
```

Changing the Duration of Built-in Effects

With the exception of `$.fn.show` and `$.fn.hide`, all of the built-in methods are animated over the course of 400ms by default. Changing the duration of an effect is simple.

Setting the Duration of an Effect

```
$('h1').fadeIn(300);      // fade in over 300ms
$('h1').fadeOut('slow');  // using a built-in speed definition
```

`jQuery.fx.speeds`

jQuery has an object at `jQuery.fx.speeds` that contains the default speed, as well as settings for "slow" and "fast".

```
speeds: {
  slow: 600,
```

```
fast: 200,  
// Default speed  
_default: 400  
}
```

It is possible to override or add to this object. For example, you may want to change the default duration of effects, or you may want to create your own effects speed.

Augmenting jQuery.fx.speeds with Custom Speed Definitions

```
jQuery.fx.speeds.blazing = 100;  
jQuery.fx.speeds.turtle = 2000;
```

Lesson 1, Activity 4: **Limitations on Effects**

Certain effects are limited in their operation by existing styling. For example, `fadeIn` modifies the opacity of an element. If the element's styling specifies `visibility:hidden`, it will never appear (to start initially hidden, you can use `display:none`, although that also has implications for the space taken up by the element).

If an element's style specifies opacity, then `fadeIn` will only take it to that opacity.

Lesson 1, Activity 5: Queuing of Effects with Other Operations

If you want to perform an operation, such as changing a class, during an effects sequence, you can queue the operation as if it were an effect. The

`$.fn.queue(callback)` function receives a callback function to add to the queue, which will be invoked at that point in the sequence.

The function should perform its operation, and then directly or indirectly invoke `$.fn.dequeue()` on the same selection.

```
// Shows the element, changes the class
// and then hides it
// notice the use of delay() to allow you to see each effect on the browser

$(document).ready(function(e) {
  $('h2').delay(1000)
    .show(1000)
    .delay(2000)
    .queue(function() {
      $(this).addClass('blue')
      .dequeue();
    })
    .hide(1000);
});
```

Code Sample:

jqy-effects/Demos/basic-effects.html

```
---- C O D E   O M I T T E D ----
<body>
<div id="div1" class="funtimes">Fun</div>
<div id="div2" class="funtimes">Fun</div>
<script src="../../jqy-lib/jquery.js"></script>
<script>
$('#div1')      // try changing to '.funtimes'
  .fadeIn(1000)
  .delay(1000)
  .fadeOut(1000);
$('#div2')
  .fadeIn(1000)
  .queue(function() { $(this).addClass('blue').dequeue(); })
```

```
.delay(1000)  
.fadeOut(1000);  
</script>  
</body>  
</html>
```

The two animations run in parallel, because they affect different elements. Both fade in, delay, and fade out again, but the `div2` box changes color due to the queued operation in the middle. If you change the first selector to `' .funtimes '`, then the operations will run in succession, since the first effects chain now applies to `div2` as well, so the second set of operations gets added to its queue (`div2` will fade in and out twice, once from the first chain, and then again from the second chain).

Lesson 1, Activity 7: **Callbacks - Doing Something When an Effect is Done**

Often, you'll want to run some code once an animation is done -- if you run it before the animation is done, it may affect the quality of the animation, or it may remove elements that are part of the animation.

Callback functions provide a way to register your interest in an event that will happen in the future. In this case, the event we'll be responding to is the conclusion of the animation. Inside of the callback function, the keyword `this` refers to the element that the effect was called on; as we did inside of event handler functions, we can turn it into a jQuery object via `$(this)`.

A callback will be invoked once for each element that the query retrieves.

Running Code When an Effect is Complete

```
$('#div.old').fadeOut(300, function() { $(this).remove(); });
```

Note that if your selection doesn't return any elements, your callback will never run! You can solve this problem by testing whether your selection returned any elements; if not, you can just run the callback immediately.

Run a Callback Even if There Were No Elements to Animate

```
var $thing = $('#nonexistent');

var cb = function() {
  console.log('done!');
};
```



```
if ($thing.length) {  
  $thing.fadeIn(300, cb);  
} else {  
  cb();  
}
```

Lesson 1, Activity 8: Custom Effects with `$.fn.animate`

jQuery makes it possible to animate arbitrary CSS properties via the `$.fn.animate` method. The `$.fn.animate` method lets you animate to a set value, or to a value relative to the current value.

Custom effects with `$.fn.animate`

Code Sample:

jqy-effects/Demos/fn-animate-custom-effects.html

```
---- C O D E   O M I T T E D ----
```

```
Fun  
Fun
```

Note: Color-related properties cannot be animated with `$.fn.animate` using jQuery out of the box. Color animations can easily be accomplished by including the color plugin. We'll discuss using plugins later in the course.

Easing

Easing describes the manner in which an effect occurs -- whether the rate of change is steady, or varies over the duration of the animation. jQuery includes only two methods of easing: swing and linear. If you want more natural transitions in your animations, various easing plugins are available.

As of jQuery 1.4, it is possible to do per-property easing when using the `$.fn.animate` method.

Per-Property Easing

```
$('#div.funtimes').animate(  
  {  
    left : [ '+=50', 'swing' ],  
    opacity : [ 0.25, 'linear' ]  
  },  
  300  
);
```

For more details on easing options, see <http://api.jquery.com/animate/>.

Lesson 1, Activity 10: **Managing Effects**

- jQuery provides several tools for managing animations.
- `$.fn.stop` - Stop currently running animations on the selected elements.
- `$.fn.delay` - Wait the specified number of milliseconds before running the next animation.

```
$('h1').show(300).delay(1000).hide(300);
```

jQuery.fx.off

If this value is true, there will be no transition for animations; elements will immediately be set to the target final state instead. This can be especially useful when dealing with older browsers; you also may want to provide the option to your users.

Lesson 1, Activity 11: Reveal Hidden Text

Duration: 20 to 30 minutes.

Open the file jqy-effects/Exercises/index.html in your browser. Use the file [blog.js](#). Your task is to add some interactivity to the blog section of the page. The spec for the feature is as follows:

1. Clicking on a headline in the `#blog` div should slide down the excerpt paragraph.
2. Note that the headlines contain hyperlinks; you should ensure the links do not get followed.
3. Clicking on another headline should slide down its excerpt paragraph, and slide up any other currently showing excerpt paragraphs.

Solution:

jqy-effects/Solutions/js/blog.js

```
$(document).ready(function() {  
  var $blog = $('#blog');  
  
  $blog.find('h3').click(function(e) {  
    e.preventDefault();  
  
    var $h3 = $(this);  
    var $p = $h3.next('p.excerpt');  
  
    $p.slideToggle();  
  
    $h3.parent().siblings()  
      .find('.excerpt:visible').slideUp();  
  });  
});
```

Since the existing HTML uses hyperlinks for the blog items, we need to disable the default behavior by calling `e.preventDefault()` in

the click handler.

We slid up only the visible paragraphs - even though it could be made to work by sliding up paragraphs within the chosen items siblings, that seems to perform poorly.

Lesson 1, Activity 12: Create Dropdown Menus

Duration: 20 to 30 minutes.

Open the file jqy-effects/Exercises/index.html in your browser. Use the file [js/navigation.js](http://jqy-effects/js/navigation.js). Your task is to add dropdowns to the main navigation at the top of the page.

1. Hovering over an item in the main menu should show that item's submenu elements, if any. Note that we would not want to show deeply nested elements, just child elements -- otherwise all levels of submenu, sub-submenu, etc., would pop open at once.
2. Exiting an item should hide any submenu items.
3. Also use the `$.fn.hover` method to add and remove a class from the submenu elements to colorize the submenu items. (The file [jqy-css/styles.css](http://jqy-effects/css/styles.css) includes the `hover` class for this purpose.)

Solution:

jqy-effects/Solutions/js/navigation.js

```
$(document).ready(function() {  
  
    $('#nav li')  
        // when a nav item is hovered,  
        // add a class to it and show  
        // any children ULs (dropdowns)  
        .hover(function() {  
            $(this) // list item that was hovered over  
                .addClass('hover')  
                .children('ul').show();  
        }, function() {  
            $(this)  
                .removeClass('hover')  
                .children('ul').hide();  
        });  
});
```

Lesson 1, Activity 14: Create a Slideshow

Duration: 20 to 30 minutes.

Open the file `index.html` in your browser. Use the file jqy-effects/Exercises/js/slideshow.js. Your task is to take a plain HTML page and enhance it with JavaScript by adding a slideshow.

The following steps have already been taken to prepare the slide show:

1. Move the `#slideshow` element to the top of the `main` element.
2. Add the `cycler` class to it. This class uses a fixed size and positioning to avoid having the display become "jumpy".
3. Find all of its `li` elements.
4. Set the `display` of all of the list items to `none`.

Your task is to implement the `fadeIn` function to handle the effects aspect of the script.

1. Fade the slide at the current index in over a few seconds, display it for a second or so, then fade it out.
2. Start the process all over again by using this function as the callback for the fade out at the end of the chain above.
3. The last line of code, in place, will reset the index when you reach the end of the elements.

Solution:

jqy-effects/Solutions/js/slideshow.js

```
$(document).ready(function() {  
  var $slideshow = $('#slideshow');  
  $slideshow  
    .prependTo('#main')  
    .addClass('cycler');  
  var $slides = $slideshow.find('li');  
  $slides.css( { 'display': 'none' } );  
}
```



```

var index = 0;
fadeIn();

function fadeIn() {
    $slides.eq(index).fadeIn(3000).delay(1000).fadeOut(500, fadeIn);
    if (++index >= $slides.length) index = 0;
}
});

```

For this exercise, the setup of the situation was already done. Although it is not overly complex, the exact steps necessary depend on a knowledge of how the fading functions in jQuery work. The absolute and relative positioning defined by the style classes ensures that a constant-size container exists, otherwise the rest of the page could shift up and down as different sized items are displayed. The size should be big enough to hold the largest item.

The items are all set to have `display: none`, to hide them initially, instead of `opacity: 0`, because the `fadeIn` function will stop out at whatever opacity was originally set for the element.

The fading in and out is accomplished with a chain of jQuery's `fadeIn`, `delay`, and `fadeOut`, with a callback to run our `fadeIn` applied at the end. The incrementing of the index and recycling back to the first item can be done after the chain is created - technically that will occur before the fading in takes place, but the fading code will be locked to its item by that point. Therefore the adjusting of the index will occur before the callback is invoked.