How jQuery Works

jQuery: The Basics

This is a basic tutorial, designed to help you get started using jQuery. If you don't have a test page setup yet, start by creating the following HTML page:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Demo</title>
</head>
<body>
    <a href="http://jquery.com/">jQuery</a>
    <script src="jquery.js"></script>
    <script>

    // Your code goes here.

    </script>
</body>
</html>
```

The `src` attribute in the `<script>` element must point to a copy of jQuery. Download a copy of jQuery from the Downloading jQuery page and store the `jquery.js` file in the same directory as your HTML file.

Note: When you download jQuery, the file name may contain a version number, e.g., `jquery-x.y.z.js` . Make sure to either rename this file to `jquery.js` or update the `src` attribute of the `<script>` element to match the file name.

Launching Code on Document Ready

To ensure that their code runs after the browser finishes loading the document, many JavaScript programmers wrap their code in an `onload` function:

```
window.onload = function() {

    alert( "welcome" );

};
```

Unfortunately, the code doesn't run until all images are finished downloading, including banner ads. To run

code as soon as the document is ready to be manipulated, jQuery has a statement known as the ready event:

```
$( document ).ready(function() {

    // Your code here.

});
```

For example, inside the `ready` event, you can add a click handler to the link:

```
$( document ).ready(function() {

    $( "a" ).click(function( event ) {

        alert( "Thanks for visiting!" );

    });

});
```

Copy the above jQuery code into your HTML file where it says `// Your code goes here` . Then, save your HTML file and reload the test page in your browser. Clicking the link should now first display an alert pop-up, then continue with the default behavior of navigating to http://jquery.com.

For `click` and most other events, you can prevent the default behavior by calling `event.preventDefault()` in the event handler:

```
$( document ).ready(function() {

    $( "a" ).click(function( event ) {

        alert( "As you can see, the link no longe

        event.preventDefault();

    });

});
```

c

Try replacing your first snippet of jQuery code, which you previously copied in to your HTML file, with the one

above. Save the HTML file again and reload to try it out.

Complete Example

The following example illustrates the click handling code discussed above, embedded directly in the HTML `<body>`. Note that in practice, it is usually better to place your code in a separate JS file and load it on the page with a `<script>` element's `src` attribute.

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Demo</title>
</head>
<body>
    <a href="http://jquery.com/">jQuery</a>
    <script src="jquery.js"></script>
    <script>

    $( document ).ready(function() {
        $( "a" ).click(function( event ) {
            alert( "The link will no longer take
            event.preventDefault();
        });
    });

    </script>
</body>
</html>
```

Adding and Removing an HTML Class

Important: *You must place the remaining jQuery examples inside the `ready` event so that your code executes when the document is ready to be worked on.*

Another common task is adding or removing a class.

First, add some style information into the `<head>` of the document, like this:

```
<style>
a.test {
    font-weight: bold;
}
</style>
```

c

Next, add the .addClass() call to the script:

```
1  $( "a" ).addClass( "test" );
```

All <a> elements are now bold.

To remove an existing class, use .removeClass():

```
1  $( "a" ).removeClass( "test" );
```

Special Effects

jQuery also provides some handy effects to help you make your web sites stand out. For example, if you create a click handler of:

```
1  $( "a" ).click(function( event ) {
2
3      event.preventDefault();
4
5      $( this ).hide( "slow" );
6
7  });
```

Then the link slowly disappears when clicked.

Callbacks and Functions

Unlike many other programming languages, JavaScript enables you to freely pass functions around to be executed at a later time. A *callback* is a function that is passed as an argument to another function and is executed after its parent function has completed. Callbacks are special because they patiently wait to execute until their parent finishes. Meanwhile, the browser can be executing other functions or doing all sorts of other work.

To use callbacks, it is important to know how to pass them into their parent function.

Callback *without* Arguments

c

If a callback has no arguments, you can pass it in like this:

```
1   $.get( "myhtmlpage.html", myCallBack );
```

When $.get() finishes getting the page `myhtmlpage.html` , it executes the `myCallBack()` function.

> **Note:** The second parameter here is simply the function name (but *not* as a string, and without parentheses).

Callback *with* Arguments

Executing callbacks with arguments can be tricky.

Wrong

This code example will ***not*** work:

```
1   $.get( "myhtmlpage.html", myCallBack( param1, para
```

The reason this fails is that the code executes `myCallBack( param1, param2 )` immediately and then passes `myCallBack()` 's *return value* as the second parameter to `$.get()` . We actually want to pass the function `myCallBack()` , not `myCallBack( param1, param2 )` 's return value (which might or might not be a function). So, how to pass in `myCallBack()` *and* include its arguments?

Right

To defer executing `myCallBack()` with its parameters, you can use an anonymous function as a wrapper. Note the use of `function() {` . The anonymous function does exactly one thing: calls `myCallBack()` , with the values of `param1` and `param2` .

```
1   $.get( "myhtmlpage.html", function() {
2
3       myCallBack( param1, param2 );
4
5   });
```

When `$.get()` finishes getting the page `myhtmlpage.html` , it executes the anonymous function, which executes `myCallBack( param1, param2 )` .